# Resource-based Corruptions and the Combinatorics of Hidden Diversity

Juan Garay* David Johnson* Aggelos Kiayias† Moti Yung‡

September 27, 2012

**Abstract**

In the setting of cryptographic protocols, the corruption of a party has traditionally been viewed as a simple, uniform and atomic operation, where the adversary decides to get control over a party and this party immediately gets corrupted. In this paper, motivated by the fact that different players may require different resources to get corrupted, we put forth the notion of *resource-based corruptions*, where the adversary must invest some resources in order to do so.

If the adversary has full information about the system configuration then resource-based corruptions would provide no fundamental difference from the standard corruption model. However, in a resource "anonymous" setting, in the sense that such configuration is hidden from the adversary, much is to be gained in terms of efficiency and security.

We showcase the power of such *hidden diversity* in the context of secure multiparty computation (MPC) with resource-based corruptions and prove that it can effectively be used to circumvent known impossibility results. Specifically, if $OPT$ is the corruption budget that violates the completeness of MPC (the case when half or more of the players are corrupted), we show that if hidden diversity is available, the completeness of MPC can be made to hold against an adversary with as much as a $B \cdot OPT$ budget, for any constant $B > 1$. This result requires a suitable choice of parameters (in terms of number of players and their hardness to corrupt), which we provide and further prove other tight variants of the result when the said choice is not available. Regarding efficiency gains, we show that hidden diversity can be used to force the corruption threshold to drop from 1/2 to 1/3, in turn allowing the use of much more efficient (information-theoretic) MPC protocols.

We achieve the above through a series of technical contributions:

- The modeling of the corruption process in the setting of cryptographic protocols through *corruption oracles* as well as the introduction of a notion of reduction to relate such oracles;
- the abstraction of the corruption game as a combinatorial problem and its analysis; and, importantly,
- the formulation of the notion of *inversion effort preserving* (IEP) functions which is a type of direct-sum property, and the property of *hardness indistinguishability.* While hardness indistinguishability enables the dissociation of parties' identities and the resources needed to corrupt them, IEP enables the discretization of adversarial work into corruption tokens,

all of which may be of independent interest.

*AT&T Labs – Research, {garay,dsj}@research.att.com.
†University of Connecticut, aggelos@kiayias.com.
‡Google Inc. and Columbia University, moti@cs.columbia.edu.

## Contents

## 1 Introduction

The notion of *computing in the presence of an adversary* which controls or gets access to parts of the system is at the heart of modern cryptography. This paradigm has given rise to cryptosystems and protocols which achieve general tasks in the presence of powerful adversaries. A prime example of the paradigm are "completeness theorems" which show that a distributed cryptographic protocol exists where an adversary controlling any minority of the parties, cannot prevent the secure computation of any efficient functionality defined over their inputs [23]. Similar secure multiparty computation (MPC) results hold over secure channels (and no additional cryptography) with an adversary controlling less than a third of the parties [6, 11]. Furthermore, these results are tight in the sense that when the adversary controls more parties there are functionalities that cannot be securely implemented.

In the works thus far however, the corruption of a party has been viewed as a simple, uniform and atomic operation where the adversary decides to corrupt a party and this party gets corrupted. The only limit the adversary has is on the number of individual party's corruptions and restrictions on when and how it can corrupt.

In this paper we are motivated by the fact that different players may require different resources to get corrupted. Indeed, the price paid for penetrating one organization, measured for example by the amount of money that a corrupt employee might have demanded, may differ from the cost of getting into another organization. Another example of a more cryptographic nature is when two organizations employ different password policies, or utilize VPN's relying on cryptosystems of different strengths. Taking such real world considerations into account gives rise to the notion we put forth of *resource-based corruption*, where the adversary must invest certain resources in order to corrupt a party. The initiation of the study of corruption models which describe situations like the above is our first contribution.

In a setting where the adversary has full information about a system, resource-based corruptions provide no fundamental divergence from the logic of the standard corruption model. The adversary, given an initial corruption budget, will target the largest subset of the system that it can afford and if such subset if large enough, the disruption of various security properties will ensue (as it follows from, e.g., [12]). The interesting case thus arises when the parties act on the system "anonymously" from the point of view of the adversary, in the sense that the association of parties' names and the individual corruption resources they require remains hidden. In this way, it seems plausible that a portion of the corruption budget will have to be wasted to learn the system configuration. Our second contribution is thus the investigation of the quantitative effect of this type of hidden diversity in the context of resource-based corruptions.

The problem the adversary faces can be abstracted as the following combinatorial game. The adversary is given a number of balls ("corruption tokens") and is faced with a sequence of buckets with the objective to fill a certain fraction of them. While it knows all the bucket sizes, it does not know their correspondence to the individual buckets in the sequence. If $OPT$ is the minimum number of balls required to fill a given percentage of the buckets had the adversary been privy to the correspondence between the buckets and their sizes, how many balls as a function of $OPT$ would be required if such hidden diversity is provided? We analyze this setting as "the combinatorics of hidden diversity" and show a number of results concerning the initial partial knowledge given to the adversary and the strategies it can apply. The analysis leads to corruption strategies (algorithms) and corruption impossibilities (lower bounds), under various cases of partial knowledge and size parameters.

Most importantly, we prove that the "value of hidden diversity" can be *unbounded!* Specifically, for any $B$ there are ways to choose buckets' sizes so that the required resources needed by the adversary to reach its target would be $B \cdot OPT$, i.e., an arbitrarily high inflation of the required corruption budget (the cryptographic implications of this are shown below).

To realize the abstractions made above—including the combinatorial game for corruptions—we introduce a formal framework where there is a special entity called a *corruption oracle* that mediates the adversary's corruption capability. We describe a variety of natural corruption oracles and introduce a notion of reduction between them that makes the translation between different corruption games that an adversary might choose feasible. In particular this allows us to capture and quantify the setting where corruptions occur due to the inversion of computationally hard problem instances (what we call "computational corruptions").

Our third contribution is the study of the sufficient conditions under which such computational corruptions can be abstracted as token-based corruptions. First, we formulate the notion of *Inversion Effort Preserving* (IEP) functions, which, at a high level, postulates a complexity lower bound on the problem of solving (inverting) multiple instances simultaneously as a function of the sum of the complexities of solving those instances individually. This notion is a type of generalized direct-sum property. Such direct-sum properties have been studied in other contexts such as communication complexity and complexity of quadratic forms (see, e.g., [18, 29, 17]) and also relate to the notion of hardness amplification [41, 27]. We show that the IEP property holds for random functions as well as exponentiation maps in idealized models such as the random oracle model [4] and the generic group model (e.g., [40]), respectively; we also provide evidence that it holds in the computational model under complexity of factoring assumptions. Second, we formulate the notion of *hardness indistinguishability* which, essentially, expresses the inability of the adversary to distinguish between two or more inversion instances of different complexity. We express this property in the statistical sense, something that facilitates our reductions between corruption models. Effectively, the former notion (IEP) enables the discretization and token abstraction of the computational effort against a sequence of problems, while the latter (indistinguishability) provides the uncertainty the adversary faces while deciding its corruption budget allocation. In turn, these two notions rely on the hardness of the inversion problem of a function when measured in an *exact* sense, a notion we also formalize, relate to past notions and investigate.

Putting everything together, we show how hidden diversity in the context of resource-based corruptions enables us to get around impossibility results. Going back to our motivating example of secure multiparty computation, the impossibility result states that when the adversary has corruption resources that exceed those needed to control a minority of players, many functionalities have to be given up. Specifically we prove the following (informally stated):

Let $OPT$ be the optimal corruption budget for which the completeness of MPC is violated. Assuming hidden diversity, there exist configurations such that:

- For any $B$, the completeness of MPC holds against any adversary with less than $B \cdot OPT$ corruption budget assuming a sufficient number $n$ of players (where $n = \Omega(\log(\frac{1}{\epsilon}) \cdot B)$, and $\epsilon$ is the probability of error).
- The above assumes the hardness of individual corruptions is not bounded. If, on the other hand, a bound $M$ is imposed, the completeness of MPC holds against any adversary with less than $\sim (\frac{\sqrt{M}}{\log(\frac{1}{\epsilon})}) \cdot OPT$ corruption budget assuming $n \geq \sqrt{M}$.

These results are expressed formally in Theorems 3.3 and 3.6, respectively. In addition, we provide evidence that the above results are essentially tight. For example, for the second formulation above, when the adversary's budget reaches an amount $\sim \sqrt{M} \cdot OPT$, there is a strategy that always corrupts half the players (Corollary 3.5).

Another way to exploit hidden diversity that we consider is to improve the efficiency of the implementation of MPC (as opposed to increasing the corruption budget the protocol can tolerate).

We prove that, assuming the same $OPT$ corruption budget, hidden diversity forces the corruption threshold to drop from 1/2 to 1/3, in turn allowing the use of much more efficient (information-theoretic) MPC protocols (see Theorem 3.7).

The above results demonstrate that there exist settings of player diversity that can be quite beneficial from a security standpoint if the diversity is appropriately hidden. This opens up the possibility for further investigation of the benefits of hidden diversity in various settings where a specific player configuration is assumed (or even can be imposed). In addition, we remark that in this paper we focus on corruptions that require effort from the adversary, which capture a wide range of attacks, from cryptographic- (e.g., an offline dictionary attack against a password file) to system-based (e.g., a brute force search against address space layout randomization), since they provide a fertile ground for relevant theoretical analysis (in contrast to settings where corruption happens essentially for free).

**Related work.** To our knowledge, this is the first time that hidden diversity in terms of corruption effort is identified and used as a mechanism to boost security guarantees. This can be juxtaposed with results in the theory of cryptography that showed how other types of adversarial uncertainty can be beneficial, perhaps the closest one of which is (sender) anonymity. For example, it has been shown that variants of an anonymous channel can be used to implement unconditionally secure point-to-point channels and a broadcast channel [19, 28], as well as more efficient natural secure computation tasks, such as private information retrieval (PIR) [28].

With respect to resource-based corruptions, the notion of general adversary structures [25], where the adversary can choose different sets of players to corrupt from a collection of possible choices, not necessarily determined by their size, can be cast as an instance of resource-based corruption for an appropriate adversary budget and players' corruption thresholds. The two models, in fact, can be shown to be equivalent; the resource-based corruption model, however, is the one that enables us to reason about hidden diversity which is our main focus.

There is also related work regarding the "lower-bounding" of adversarial effort, including the notion of "moderately hard functions" to fight spam [16], cryptographic key escrow [39, 2], time-lock puzzles and timed-release cryptography (e.g., [36, 7, 21]), and resource fairness [20]. We note that in most such applications, however, the common theme is relating work to time, and thus crafting problems whose solution is hard to parallelize, which is not the case here. In terms of measuring adversarial effort, our approach is along the same lines as *precise* zero-knowledge [32] where the knowledge gained by a player is measured in terms of its actual computation.

In an independent development, Bellare, Ristenpart and Tessaro [3] recently introduced the notion of *multi-instance* (MI) security, which relates to the notion of IEP functions introduced here. In settings where it is computationally feasible for instances to be compromised ("inverted," in our parlance), such as password-based cryptography, the application where the notion is showcased, it is shown that security can be amplified linearly in the number of instances in the random oracle (RO) model under a proper modeling. Such modeling, called "left-or-right xor" (LORX), is introduced in [3], aiming to capture the level of multi-instance security of a cryptographic primitive (encryption in that paper) with respect to indistinguishability-type challenges. In contrast, and in line with our objectives, the IEP property captures the multi-instance hardness behavior of one-way functions, and, as opposed to LORX, is cast as an intrinsic complexity characteristic that expresses the rate of hardness growth as a function of the number of instances.

As we already mentioned above, the notion of IEP functions also bears some resemblance to hardness amplification and direct-product theorems [41, 27]; we elaborate on such relation in Appendix A.2. Finally, similar notions to the notion of exact hardness that we put forth in this paper have been considered in the literature; see Section 4.1 and in particular Appendix A.1 for a comparison to another "more refined than asymptotic" measure of inversion difficulty considered by Haitner, Harnik and Reingold [24].

**Organization of the paper.** The rest of the paper is organized as follows. In Section 2 we introduce the notion of resource-based corruptions, corruption oracles and reductions between them. In Section 3, we present the combinatorial analysis that allows us to prove the gains of hidden diversity at an abstract, "token" level. Finally, in Section 4 we give the definitions and instantiations of exact hardness, IEP and hardness indistinguishability, which enable the application of the results to the computational-corruption setting. For ease of readability, some of the proofs, comparisons to related notions, candidate functions satisfying the Section 4 definitions, as well as additional analysis and constructions, are presented in the appendix. We first introduce some basic notation.

**Notation.** We use $\lambda \in \mathbb{N}$ to denote the security parameter. All quantities are assumed to be functions of $\lambda$ unless otherwise noted. Let $X, Y$ be random variables with range in $\{0,1\}$. We write $X \overset{\epsilon}{\approx} Y$ if we have that $|\mathbf{Pr}[X = 1] - \mathbf{Pr}[Y = 1]| < \epsilon$.

## 2 Resource-based Corruptions

A main goal of this paper is to formulate the cost for an adversary to "corrupt" parties running a cryptographic protocol, in contrast to the traditional approach where the adversary gains control of parties for free. We will be modelling this process in what nowadays is the widely accepted method to formulate the security of a protocol carrying out a given task: the "trusted-party paradigm" [23]. Recall that in this paradigm, the protocol execution is compared with an ideal process where the outputs are computed by a trusted party that sees all the inputs. A protocol is then said to securely carry out a given task if running the protocol with a realistic adversary amounts to "emulating" the ideal process with the appropriate trusted party. In the perhaps most developed version of the paradigm, due to Canetti [8], the task of distinguishing between the two experiments is assigned to an entity (a polynomial-time interactive Turing machine [ITM]) called *the environment*, and denoted by $\mathcal{Z}$, which is also in charge of producing the inputs for and receiving the outputs from both executions. Besides $\mathcal{Z}$, the other basic entities (also ITMs) involved in the real-world execution are $n$ players $P_1, ..., P_n$, and an adversary $\mathcal{A}$. We will be using a variant of this formulation, similar to [20], meant to capture synchronous communication and exact running-time bounds[1].

### 2.1 Corruption Oracles and Security Definition

We model the corruption process, under different costs, by the addition of a new entity (ITM) to the real-world execution, which we call the *corruption oracle* ($\mathcal{C}$), and whose essential purpose is to interact with adversary $\mathcal{A}$ and manage its corruption capability. In more detail, the two basic principles in $\mathcal{C}$'s operation are as follows:

- $\mathcal{C}$ is initialized with a random tape and the number of players $n \in \mathbb{N}$ involved in the system. It may return some information about the players to the adversary $\mathcal{A}$.
- For each player $P_i$, $\mathcal{A}$ may engage $\mathcal{C}$ in a simple (corruption) protocol, call it $\rho_i$, to determine whether player $P_i$ gets corrupted. The adversary is free to schedule many such corruption protocols concurrently (either statically for a static adversary or dynamically for an adaptive

[1]We note that these formulations are in fact more powerful than needed for this paper, as we will be focusing on stand-alone protocol execution.

one). If the protocol terminates with $\mathcal{C}$ accepting, then player $P_i$ is declared corrupted, meaning that $\mathcal{A}$ has access to $P_i$'s internal state, and is able to impersonate $P_i$ in all of its subsequent interactions (for a malicious $\mathcal{A}$).

We consider communication with the corruption oracle as part of the adversary's basic step of (significant) computation (see Section 4.1). For succinctness, sometimes we will refer to the adversary $\mathcal{A}$ interacting with corruption oracle $\mathcal{C}$ as $\mathcal{A}^{\mathcal{C}}$. Following [8], we will use the notation $\text{EXEC}_{\pi,\mathcal{A}^{\mathcal{C}},\mathcal{Z}}$ to denote the (binary) distribution ensemble describing $\mathcal{Z}$'s output after interacting with adversary $\mathcal{A}$ running with corruption oracle $\mathcal{C}$, and players running protocol $\pi$.

In the ideal process we will model the corruption capability as follows. We consider a "wrapper" functionality $\mathcal{W}^{\mathcal{C}}(\mathcal{F})$ that for any functionality $\mathcal{F}$ and corruption oracle $\mathcal{C}$, "traps" the corruption requests directed to a party participating in it; all other messages and requests it immediately passes to the functionality. This will allow us to provide a corruption interface that is compatible with the corruption-oracle modeling of the real world. In more detail, for each party $P_i$ involved in $\mathcal{F}$ the wrapper $\mathcal{W}^{\mathcal{C}}(\mathcal{F})$ initializes the corruption oracle and considers all parties as "uncorrupted." Mimicking the behavior of the real-world adversary attacking parties, $\mathcal{S}$ can submit corruption messages for party $P_i$ following the protocol $\rho_i$ to $\mathcal{W}^{\mathcal{C}}(\mathcal{F})$, who maintains the state of the corruption oracle. If, at some point, $\mathcal{C}$ terminates the execution of one of the $\rho_i$ protocols and accepts, then $\mathcal{W}^{\mathcal{C}}(\mathcal{F})$ sets the party $P_i$ status to "corrupted," issues a (standard) corruption message for $P_i$, $(\text{Corrupt}, P_i)$, to $\mathcal{F}$, and returns the functionality's response, namely, the party's internal state as kept by the functionality, back to $\mathcal{S}$. In addition, we note that once a party is corrupted, the ideal adversary is typically allowed to "re-write" the corrupted party's internal state; $\mathcal{W}^{\mathcal{C}}(\cdot)$ ignores such messages while the party is uncorrupted. Let $\text{EXEC}_{\mathcal{W}^{\mathcal{C}}(\mathcal{F}),\mathcal{S},\mathcal{Z}}$ denote the (binary) distribution ensemble describing $\mathcal{Z}$'s output after interacting with adversary $\mathcal{S}$ and the ideal protocol for $\mathcal{W}^{\mathcal{C}}(\mathcal{F})$ as specified above.

We now proceed to more formally specify how the two executions—real and ideal—would be indistinguishable to the environment. The typical order of quantifiers in simulation-based security definitions ($\forall\mathcal{A}\exists\mathcal{S}\forall\mathcal{Z}$) allows the ideal-world adversary to depend on the real-world adversary that it simulates, but it should be independent of the environment. Following [20], we use a slight weakening of this definition, which is appropriate for the setting of adversaries restricted to performing a specific number of steps of the computation. Specifically, let $t$ denote the number of steps taken by the adversary, for $t$ a monotonically increasing arithmetic function; we consider the "compound" ITM $\langle\mathcal{Z},\mathcal{A}\rangle$ (namely, two ITMs, interacting with each other and possibly with other ITMs in a well-defined manner, treated as a single entity) as being $t$-bounded. (This refinement will become useful when capturing corruptions as computational effort—cf. Section 4.)

**Definition 2.1** *For a given $t$, a protocol $\pi$ is said to* securely realize a functionality $\mathcal{F}$ against $t$-bounded adversaries and corruption oracle $\mathcal{C}$, *if for all $\mathcal{A}$ there exists an ideal-world adversary $\mathcal{S}$, running in time $t + p$, such that for all $\mathcal{Z}$ with $\langle\mathcal{Z},\mathcal{A}\rangle$ being $t$-bounded,*

$$\text{EXEC}_{\pi,\mathcal{A}^{\mathcal{C}},\mathcal{Z}} \overset{\epsilon}{\approx} \text{EXEC}_{\mathcal{W}^{\mathcal{C}}(\mathcal{F}),\mathcal{S},\mathcal{Z}},$$

*where $\epsilon$ is some negligible function and $p$ some polynomial.*

Next, we introduce the notion of *safety* for a corruption oracle.

**Definition 2.2** *For a given $t$, we say a corruption oracle $\mathcal{C}$ is $t$-*safe *if for all functionalities $\mathcal{F}$ (including those that guarantee fairness and output delivery to all honest parties), there is a protocol $\pi$ that securely realizes $\mathcal{F}$ against $t$-bounded adversaries and $\mathcal{C}$.*

If a corruption oracle is $t$-safe for all $t$, then we call it simply safe. For example, it can be derived from [23, 9] that the standard (traditional) corruption oracle allowing less than 1/2 of the players to be corrupted (to be denoted below as $\mathcal{C}^{\mathsf{std}}_{\frac{1}{2}}$) is safe. Further, in case we want to be explicit about the parameters involved in the proof of safety, we will say that a corruption oracle $\mathcal{C}$ is $(t, \gamma)$-safe if it holds that the above definition is satisfied with a simulation that may fail with probability at most $1 - \gamma$.

## 2.2 Cost Measures and Relations between Corruption Oracles

Next, we will start by capturing two cost measures for corruptions, namely, the traditional one (i.e., no cost) and the "token" based (i.e., discretized) measure, defining corruption oracles for them, which will already enable us to establish the power of hidden diversity at an abstract level; later on (Section 4) we will provide the final "translation" by presenting the computational-based measure. In addition, we will define the "blinded" version of such oracles (which, looking forward, will effectively model the hardness indistinguishability of functions put forth in Section 4.1), and establish the relations between them.

**Standard cryptographic corruption.** In this case the corruption oracle is initialized with a threshold $\alpha \in (0, 1)$, and maintains a counter $\mathsf{ctr}$ initially set to 0. The oracle provides the number of players $n$ to the adversary. The corruption protocol $\rho_i$ consists of a single message $(\textsc{Corrupt}, P_i)$ that is to be transmitted by the adversary to the oracle. Given such a message, the oracle checks whether $\mathsf{ctr} + 1 < \lceil \alpha \cdot n \rceil$, and in this case declares player $P_i$ corrupted (with the effect described above) and increases $\mathsf{ctr}$ by 1. We denote this corruption oracle by $\mathcal{C}^{\mathsf{std}}_{\alpha}$. For secure multi-party computation applications, typical values of $\alpha$ are $\frac{1}{2}$ (i.e., honest majority), $\frac{1}{3}$ and $1 - \frac{1}{n}$ (i.e., at least one honest player).

**Token-based corruption.** In this case the corruption oracle is initialized with a vector $\overline{s} \in \mathbb{N}^n$ ($\overline{s}$ for bucket "sizes") and a threshold $k$ (total number of tokens); the threshold of player $P_i$ is the value $s_i$. The oracle gives to $\mathcal{A}$ the vector $\overline{s}$, and maintains counters $\mathsf{ctr}_1, \ldots, \mathsf{ctr}_n$ initially set to 0. Protocol $\rho_i$ here consists of messages of the form $(\textsc{Corrupt}, P_i, v)$ sent by $\mathcal{A}$. The oracle checks that $v + \sum_i \mathsf{ctr}_i \leq k$, and in this case it increases counter $\mathsf{ctr}_i$ by $v$; if it happens that $\mathsf{ctr}_i \geq s_i$, then player $P_i$ gets corrupted. We will denote the token-based corruption oracle by $\mathcal{C}^{\mathsf{tk}}_{\overline{s},k}$.

**Blinded token-based corruption.** This corruption oracle, denoted by $\mathcal{C}^{\mathsf{btk}}_{\overline{s},k}$, is identical to $\mathcal{C}^{\mathsf{tk}}_{\overline{s},k}$ in operation with the following difference: whenever the adversary submits a $(\textsc{Corrupt}, P_i, v)$ message, the oracle performs the update operations on player $P_{p(i)}$ where $p$ is a secret random permutation that is selected initially and maintained by the corruption oracle. Otherwise, the blinded token-based corruption oracle behaves as the token-based corruption oracle. Given that this oracle has a private state (permutation $p$), for technical reasons (to be revealed later) we will also consider a "leaky" version of this corruption oracle that is parameterized by an ITM $L$ and operates on the private permutation $p$. In this leaky version, the adversary $\mathcal{A}$ may submit a special request given which the corruption oracle will run $L$ on its internal state and return its output to $\mathcal{A}$. We will denote the leaky version of this corruption oracle by $\mathcal{C}^{\mathsf{btk},L}_{\overline{s},k}$.

We now set out to study the relations between corruption oracles defined above. Our main tool is the following definition.

**Definition 2.3** *Fix some $\epsilon > 0$ and arithmetic function $t$. Given two corruption oracles $\mathcal{C}_1$ and $\mathcal{C}_2$, we say that $\mathcal{C}_2$* dominates $\mathcal{C}_1$ with error $\epsilon$ and complexity $t$, *denoted $\mathcal{C}_1 \leq^t_\epsilon \mathcal{C}_2$, provided the* *following holds: for $n \in \mathbb{N}$ and any $n$-party protocol $\pi$, for any $\mathcal{A}$, there is an adversary $\mathcal{B}$, running in time $t+p$, such that for all $\mathcal{Z}$ with $\langle \mathcal{Z}, \mathcal{A} \rangle$ being $t$-bounded,*

$$\mathrm{EXEC}_{\pi,\mathcal{A}^{\mathcal{C}_1},\mathcal{Z}} \overset{\epsilon}{\approx} \mathrm{EXEC}_{\pi,\mathcal{B}^{\mathcal{C}_2},\mathcal{Z}},$$

*where $p$ is some polynomial.*

The main intuition behind this definition is that if $\mathcal{C}_2$ dominates $\mathcal{C}_1$ and an adversary is given a choice between the two, it may always opt for the former. The way we will employ the above relation in our exposition is that if $\mathcal{C}_2$ dominates $\mathcal{C}_1$ and it happens that $\mathcal{C}_2$ is safe, then $\mathcal{C}_1$ is also safe. More specifically, if $\mathcal{C}_2$ is $(t,\gamma)$-safe and we have that $\mathcal{C}_1 \leq^t_\epsilon \mathcal{C}_2$, then we have that $\mathcal{C}_1$ is $(t, \gamma-\epsilon)$-safe. It is easy to see that the relation $\leq^t_\epsilon$ is reflexive and transitive and thus constitutes a preorder for any choice of the parameters $t, \epsilon$.

Another easy observation is that $\mathcal{C}^{\mathsf{std}}_{\gamma_1} \leq^t_\epsilon \mathcal{C}^{\mathsf{std}}_{\gamma_2}$ provided that $\gamma_1 \leq \gamma_2$, independently of the values $t, \epsilon$. This stems from the fact that the power of the adversary can only potentially increase in the corruption oracle of the right-hand side. When the relation $\leq^t_\epsilon$ holds for any function $t$, we will write $\leq_\epsilon$.

**From token-based corruptions to standard corruptions.** Let $\mathcal{A}$ be any adversary that interacts with $\mathcal{C}^{\mathsf{X}}_{\overline{s},k}$ for some $\overline{s} = (s_1, \ldots, s_n) \in \mathbb{N}^n, k \in \mathbb{N}$ and $\mathsf{X} \in \{\mathsf{tk}, \mathsf{btk}\}$. For some $\alpha \in (0,1)$ and complexity $t$, we denote by $\mathsf{bad}^t_{\mathsf{X}}[\overline{s}, k, \alpha]$ the supremum of the probability of the event that the number of corrupted players reaches $\lceil \alpha \cdot n \rceil$ in the execution of $M^{\mathcal{C}^{\mathsf{X}}_{\overline{s},k}}$, where $M$ is any $t$-bounded ITM. We have the following :

**Lemma 2.4** *For any $n, k \in \mathbb{N}, \overline{s} \in \mathbb{N}^n, \alpha \in (0,1)$ and arithmetic function $t$, we have that $\mathcal{C}^{\mathsf{X}}_{\overline{s},k} \leq^t_\epsilon \mathcal{C}^{\mathsf{std}}_\alpha$, where $\epsilon \leq \mathsf{bad}^t_{\mathsf{X}}[\overline{s}, k, \alpha]$ and $\mathsf{X} \in \{\mathsf{tk}, \mathsf{btk}\}$.*

*Proof.* The description of $\mathcal{B}$ is based on $\mathcal{A}$. Specifically, $\mathcal{B}$ will simulate the corruption oracle $\mathcal{C}^{\mathsf{X}}_{\overline{s},k}$ for $\mathcal{A}$ and whenever a player is corrupted according to the corruption oracle it will pass the corresponding corruption message to its own corruption oracle $\mathcal{C}^{\mathsf{std}}_\alpha$. The only problem that may occur in the simulation is when the corruption oracle of $\mathcal{B}$ rejects its corruption request for a certain player while the corresponding player is expected to be corrupted by $\mathcal{A}$. Using the fact that $\mathbf{Pr}[X|\neg B] = \mathbf{Pr}[Y|\neg B]$ implies that $|\mathbf{Pr}[X] - \mathbf{Pr}[Y]| \leq \mathbf{Pr}[B]$ for any three events $X, Y, B$ over the same probability space yields the proof, by taking $X$ to be the output of $\mathcal{A}$'s execution, $Y$ to be the output of $\mathcal{B}$'s execution, and $B$ the probability that $\mathcal{A}$ is able to successfully corrupt a player while $\mathcal{B}$ is denied. $\square$

This lemma would be most useful in case $\epsilon$ is negligible (cf. next section).

## 3 The Combinatorics of Hidden Diversity

In this section we will use the relations we established between the corruption oracles in the previous section in order to derive cryptographic safety bounds at a purely combinatorial level. In particular we will provide bounds for the "bad" event (Lemma 2.4) and negative results—for the adversary—showing how the blinded version of corruption oracles remains safe for ranges of parameters that are unsafe in the regular case, hence demanding from the adversary a substantially higher "budget" to achieve its goal.

We defined two types of token-based corruption oracles, regular and blinded, which are specified by two parameters: $\overline{s}, k$. Let $OPT_\alpha(\overline{s})$ be the minimal number of tokens that need to be invested in order to corrupt a set of players of size at least $\lceil \alpha n \rceil$ in the token-based corruption oracle model, i.e., $OPT_\alpha(\overline{s}) = \min\{k : \exists C \subseteq \{1, \dots, n\}$ with $|C| \geq \lceil \alpha n \rceil$ and $\sum_{i \in C} s_i = k\}$. Based on this definition, Lemma 2.4, and our observation that $\mathcal{C}^{\mathsf{std}}_{1/2}$ is safe, we have the following.

**Theorem 3.1** *For any $n \geq 2$, and $\overline{s} \in \mathbb{Z}^n$, the corruption oracle $\mathcal{C}^{\mathsf{tk}}_{\overline{s},k}$ is safe for any $k < OPT_{1/2}(\overline{s})$ and unsafe for any $k \geq OPT_{1/2}(\overline{s})$.*

*Proof.* First, regarding cryptographic safety, we observe that as long as $k < OPT_{\frac{1}{2}}(\overline{s})$, it holds that an honest majority of players is always guaranteed. Due to the results of [23], it follows that a protocol can be constructed for any functionality $\mathcal{F}$. On the flip side, if $k > OPT_{\frac{1}{2}}(\overline{s})$ this means that there exists a set of players $C$ for which it holds that $k \geq \sum_{i \in C} s_i$ and $|C| \geq \lceil \frac{n}{2} \rceil$. It follows that by corrupting this set of players it is impossible to realize all functionalities (this follows from the impossibility result of Cleve [12]). $\square$

Next, we demonstrate how the blinded token-based corruption oracle remains safe for ranges of parameters that are unsafe in the regular case.

**Balls and buckets.** The problem at hand can be rephrased as the following game, which is further elaborated on in Appendix B. An adversary wishes to distribute balls (corresponding to corruption tokens) to $n$ buckets, so as to fill at least $\lceil \alpha n \rceil$ of them for some given $\alpha \in (0, 1)$. The sizes of the buckets are given in the form of a vector $\overline{s} \in \mathbb{Z}^n$. If the adversary has full information about the buckets, it can achieve its goal by investing $OPT_\alpha(\overline{s})$ balls. Given the characteristics of the $\mathcal{C}^{\mathsf{btk}}_{\overline{s},k}$ oracle, we are interested in the case where the adversary does not know the correspondence between buckets and $\overline{s}$ so it may have to waste a certain number of balls to reach the $\lceil \alpha n \rceil$ threshold. Specifically, we assume that there is a hidden random permutation $\pi$ that re-labels the buckets so the adversary knows $\overline{s}^\pi = (s_{\pi(1)}, \dots, s_{\pi(n)})$. We note that it should be the case that $\overline{s}$ includes at least 2 different sizes, since otherwise the hidden permutation would not stall the adversary in any way. Specifically, the cardinality of the set $\{s \mid \exists i : s = s_i\}$ is bigger than 1.

## 3.1 Increased Security from Hidden Diversity

We first consider the case when there are no restrictions on the number of sizes or their values.

**Theorem 3.2** *For any $\alpha$, $0 < \alpha < 1$, constants $B > 1$ and $\epsilon > 0$, and for any $n \geq \log(1/\epsilon) \cdot \max\{1/\alpha, (4B-2)/(1-\alpha)\}$, there exists a vector $\overline{s} \in \mathbb{Z}^n$ such that any adversary that is given $\overline{s}^\pi$ for a random permutation $\pi$, and has fewer than $B \cdot OPT_\alpha(\overline{s})$ balls, has probability less than $\epsilon$ of filling $\lceil \alpha n \rceil$ buckets.*

*Proof.* We may assume without loss of generality that $n > 8B/(1-\alpha)$. For a $c$, $0 < c \leq \alpha$, to be specified later, our instance will have $\lceil cn \rceil$ buckets of size $\lceil \alpha n \rceil + 1$, $\lceil \alpha n \rceil - \lceil cn \rceil$ buckets of size 1, and $n - \lceil \alpha n \rceil$ buckets of size $(\lceil cn \rceil + 2)B\lceil \alpha n \rceil$. An optimal solution will consist of the $\lceil \alpha n \rceil$ smallest buckets, and will have total size $(\lceil cn \rceil + 1)\lceil \alpha n \rceil$. This implies that we cannot afford to fill any of the largest buckets if we are to use fewer than $B \cdot OPT_\alpha(\overline{s}) = B(\lceil cn \rceil + 1)\lceil \alpha n \rceil$ balls, and any algorithm that fills $\lceil \alpha n \rceil$ buckets must find and fill all the buckets of size $\lceil \alpha n \rceil + 1$.

But now note that the only way the adversary can tell whether a bucket has this size or one of the larger ones is to place $\lceil \alpha n \rceil + 1$ balls in the bucket. Thus, if the adversary is to use no more than $B(\lceil cn \rceil + 1)\lceil \alpha n \rceil$ balls, it can test no more than $B(\lceil cn \rceil + 1)$ buckets with size exceeding 1, of which there are $n - \lceil \alpha n \rceil + \lceil cn \rceil$. The probability of finding all $\lceil cn \rceil$ of the mid-size buckets is then at most

$$\binom{n - \lceil \alpha n \rceil}{B(\lceil cn \rceil + 1) - \lceil cn \rceil} \Big/ \binom{n - \lceil \alpha n \rceil + \lceil cn \rceil}{B(\lceil cn \rceil + 1)}.$$

Setting $X = n - \lceil \alpha n \rceil$ and $Y = B(\lceil cn \rceil + 1) - \lceil cn \rceil$, this is equal to

$$\binom{X}{Y} \Big/ \binom{X + \lceil cn \rceil}{Y + \lceil cn \rceil} \quad = \quad \frac{(Y+1)(Y+2)\cdots(Y+\lceil cn \rceil)}{(X+1)(X+2)\cdots(X+\lceil cn \rceil)},$$

which we will be able to argue is sufficiently small if we can chose $c > 0$ such that, say, $Y + \lceil cn \rceil \le (X + \lceil cn \rceil)/2$. This requires that $B\lceil cn \rceil + B < (n - \lceil \alpha n \rceil + \lceil cn \rceil)/2$, or $(2B-1)\lceil cn \rceil \le n - \lceil \alpha n \rceil - 2B$. which will be true so long as $(2B-1)(cn+1) \le n(1-\alpha) - 1 - 2B$, or $(2B-1)(cn) \le n(1-\alpha) - 4B$, or $c \le (1-\alpha)/(2B-1) - 4B/(n(2B-1)$. By our assumption that $n > 8B/(1-\alpha)$, this means that all we need for $Y + \lceil cn \rceil < (X + \lceil cn \rceil)/2$ is that $c \le (1-\alpha)/(4B-2)$. By our construction, we also need $c \le \alpha$, so it suffices to set $c = \min\{\alpha, (1-\alpha)/(4B-2)\}$.

Given that $Y + \lceil cn \rceil < (X + \lceil cn \rceil)/2$, we then have that $(Y+i)/(X+i) \le 1/2$ for $1 \le i\lceil cn \rceil$, and hence the probability of success is at most

$$\frac{(Y+1)(Y+2)\cdots(Y+\lceil cn \rceil)}{(X+1)(X+2)\cdots(X+\lceil cn \rceil)} < (1/2)^{\lceil cn \rceil} \le (1/2)^{cn}.$$

Setting $(1/2)^{cn} = \epsilon$ and solving for $n$ yields the claimed result. □

The above theorem implies the following result.

**Corollary 3.3** *For any $B > 1$ and $\epsilon > 0$, and for any $n \ge \log(1/\epsilon)(8B-4)$, there exists a vector $\overline{s} \in \mathbb{Z}^n$ such that the corruption oracle $\mathcal{C}^{\mathsf{btk}}_{\overline{s},k}$ is $(1-\epsilon)$-safe provided that $k < B \cdot OPT_{1/2}(\overline{s})$.*

*Proof.* We show that $\mathcal{C}^{\mathsf{btk}}_{\overline{s},k} \le_\epsilon \mathcal{C}^{\mathsf{std}}_{n,\frac{1}{2}}$ for the choice of parameters $n, \overline{s}, k$ of the previous theorem. Cryptographic safety will follow then immediately due to Theorem 3.1.

In order to prove the reduction between the corruption oracles we need to provide an adversary $\mathcal{B}$ operating with $\mathcal{C}^{\mathsf{std}}_{n,1/2}$ for any adversary $\mathcal{A}$ operating with $\mathcal{C}^{\mathsf{btk}}_{\overline{s},k}$. $\mathcal{B}$ will simply simulate the corruption oracle of $\mathcal{A}$ and submit the corruption requests to $\mathcal{C}^{\mathsf{std}}_{n,1/2}$ whenever a corruption with $\mathcal{C}^{\mathsf{btk}}_{\overline{s},k}$ takes place. The simulator will fail with probability at most $\mathsf{bad}_{\mathsf{btk}}[\overline{s}, k, 1/2]$ from Lemma 2.4. Due to the previous theorem we can bound the probability by $\epsilon$, which completes the proof. □

The above results require arbitrarily large bucket sizes (components of $\overline{s}$). In real applications, it is plausible that there might be some upper bound $M$ on the maximum size. We now show how an adversary can exploit this (and other restrictions), and what level of safety may remain.

Consider the following algorithm $H2$ that is a hybrid between two simple bucket-filling strategies: one that continuously picks an empty bucket and fills it, and one that layers balls horizontally across all buckets. Let $M' = \lfloor \sqrt{M} \rfloor$. Algorithm $H2$ proceeds in two basic steps: (1) While less than $\lceil \alpha n \rceil$ buckets are full and there is an unfull bucket containing fewer than $M'$ balls, choose one with the fewest balls and place a ball in it. (2) While less than $\lceil \alpha n \rceil$ buckets are full, pick an unfull bucket and add balls until it is filled to capacity.

Let $H2_\alpha(\overline{s})$ denote the worst-case number of balls that $H2$ must place in order to fill $\lceil \alpha n \rceil$ buckets. The following upper bound on the behavior of $H2$ is proved in Section B.3.4 of the Appendix.

**Theorem 3.4** *For $0 < \alpha < 1, n \in \mathbb{N}$ and any $\overline{s} \in \mathbb{Z}^n$ with maximum size $M$, $\frac{H2_\alpha(\overline{s})}{OPT_\alpha(\overline{s})} \le 1 + \frac{\sqrt{M}}{\alpha}$.*

**Corollary 3.5** *For any $n > 1$, $M > 0$, and any $\overline{s} \in \mathbb{Z}^n$ with maximum level $M$, the corruption oracle $\mathcal{C}^{\mathsf{btk}}_{\overline{s},k}$ is unsafe whenever $k \geq (1 + 2\sqrt{M}) \cdot OPT_{1/2}(\overline{s})$.*

*Proof.* The proof follows as a direct application of the previous theorem. Specifically, we have that due to the previous theorem there is an adversarial strategy that needs at most $1 + 2\sqrt{M}$ balls to fill $\lceil \frac{1}{2} \cdot n \rceil$ buckets. It follows that with this strategy an adversary can corrupt $\lceil \frac{1}{2} \cdot n \rceil$ players always and as we argued in Theorem 3.1, it follows that the corruption oracle is unsafe. □

The above relation between the maximum bucket size and unsafety is essentially tight—once the number of corruption tokens drops below $\sqrt{M}$ by a fixed fraction there exist $\overline{s}$ with maximum size $M$ where with high probability the adversary will fail to fill the target number of buckets. Theorem B.8 in the Appendix implies the following.

**Theorem 3.6** *For any $\epsilon > 0$ and any $M \geq 8$, $n > 4$ with $\lceil \sqrt{M} \rceil$ dividing $n$, there is an $\overline{s} \in \mathbb{Z}^n$ with maximum size $M$, such that the corruption oracle $\mathcal{C}^{\mathsf{btk}}_{\overline{s},k}$ is $(1 - \epsilon)$-safe provided that $k < \delta \cdot \sqrt{M} \cdot OPT_{1/2}(\overline{s})$ where $\delta = \min\big(2/45, (4/15)/\log(1/\epsilon)\big)$.*

It is worth notingthat this theorem becomes trivial if $\delta < 1/\sqrt{M}$, since in that case we would be given fewer than $OPT_{1/2}(\overline{s})$ balls and so even the unblinded corruption oracle would be safe. This in turn implies that the theorem is only meaningful for $\epsilon > 2^{-(4/15)\sqrt{M}}$.

The above results about the cryptographic safety of the blinded token-based corruption oracle are all proved using systems $\overline{s}$ that have three distinct sizes. The fact that we use more than two sizes turns out to be necessary. For systems with just two sizes, Theorem B.4 in the Appendix implies that there is an adversarial strategy that violates cryptographic safety given just $2 \cdot OPT_{1/2}(\overline{s})$ corruption tokens. This bound turns out to be tight, as shown in Theorem B.5(B).

## 3.2 Increased Efficiency from Hidden Diversity

Next, we explore the question whether hidden diversity can be used to relax the computational effort in secure multiparty computation (MPC). Consider an adversary that is given $k < OPT_{1/2}(\overline{s})$ corruption tokens. Recall that the corruption oracle $\mathcal{C}^{\mathsf{tk}}_{\overline{s},k}$ is safe for exactly this range of adversarial resources, i.e., fully secure MPC can be achieved under computational assumptions [23, 33, 9]. The problem we investigate in this section is what potential benefits can be reaped in the hidden diversity setting assuming the same level of adversarial resources. Theorem B.15 in the Appendix implies the following result.

**Theorem 3.7** *For any $\beta > 0$, there are constants $N > 1$ and $a < 1$, such that for any $n > N$, there is a vector $\overline{s} \in \mathbb{Z}^n$ with $\mathcal{C}^{\mathsf{btk}}_{\overline{s},k} \leq_{a^n} \mathcal{C}^{\mathsf{std}}_{1/4+\beta}$, provided that $k \leq OPT_{1/2}(\overline{s})$.*

Thus, if we choose $\beta = 1/3 - 1/4 = 1/12$ and an instance $\overline{s}$ with sufficiently large $n$, an adversary will have probability less than $\epsilon$ of corrupting $1/3$ or more participants, where "sufficiently large" here again grows proportionally with $\log(1/\epsilon)$. This result can be used to extend the application domain of information-theoretic protocols for fully secure MPC such as those of [6, 14], which are typically much more efficient than the cryptographic ones, but that in the regular corruption model only tolerate a rate of corruptions less than $1/3$.

Appendix B contains additional results in the blinded balls-and-buckets model, including adversarial strategies that bound how much the previous result can be extended and an examination of the case where the number of buckets $n$ is bounded but not the maximum size $M$.

## 4 Computational Corruptions

In this last section we turn to capturing and quantifying the setting where corruptions occur due to the inversion of computationally hard problem instances (what we call "computational corruptions"), and study the sufficient conditions under which such corruptions can be abstracted out as token-based corruptions. We start by defining the corresponding corruption oracles. First, the computational corruption oracle captures the setting where the adversary can corrupt a player by "breaking in," specifically by solving an instance of a computational problem that is otherwise unrelated to the cryptographic protocol (for example, this would be the case when the adversary's offline attack against a cryptographic authentication protocol succeeds). The *blinded* computational corruption oracle extends the above to the setting where the adversary cannot determine the correspondence between players and the instances that need to be solved for a break-in to occur.

**Computational corruption oracle.** The oracle is initialized with the description of functions $f_1, \ldots, f_n$ and samples $x_1, \ldots, x_n$ from the functions' respective domains. Subsequently, it provides to the adversary the vector $(y_1, \ldots, y_n)$, where $y_i = f_i(x_i)$. The corruption protocol $\rho_i$ here consists of messages of the form $(\textsc{Corrupt}, P_i, x)$ provided by $\mathcal{A}$; if it happens that $y_i = f_i(x)$, then player $P_i$ gets corrupted[2]. We denote this oracle by $\mathcal{C}^{\mathsf{cc}}_{\overline{f}}$, where $\overline{f} = (f_1, \ldots, f_n)$.

**Blinded computational corruption oracle.** This case is similar to computational corruption with the difference that in addition, the corruption oracle is initialized with a random permutation $p$. The oracle provides to $\mathcal{A}$ the vector $(y_{p(1)}, \ldots, y_{p(n)})$. The corruption protocol messages are as in the case of $\mathcal{C}^{\mathsf{cc}}_{\overline{f}}$. We denote this corruption oracle by $\mathcal{C}^{\mathsf{bcc}}_{\overline{f}}$. We will be considering this definition in the setting when the functions $\overline{f}$ are hardness indistinguishable.

Next, we introduce the two notions that play a fundamental role in the complexity interpretation of resource-based corruptions.

### 4.1 Hardness Indistinguishability and Inversion-Effort-Preserving Functions

Both notions are related to the hardness of the inversion problem of a function when measured in an *exact* sense. We start by explaining this notion of hardness first.

#### 4.1.1 Exact hardness

Consider any function $f : X \to Y$, where $X = \cup_{\lambda \in \mathbb{N}} X_\lambda, Y = \cup_{\lambda \in \mathbb{N}} Y_\lambda$. In what follows, if $\lambda$ is clear from the context, we may denote $X_\lambda$ and $Y_\lambda$ by simply $X$ and $Y$, respectively.

An inversion algorithm for $f$ with success $p(\lambda)$ in time $t(\lambda)$, is a non-uniform algorithm $\mathcal{A}$ that for any $\lambda$, receives input $f(x)$ where $x$ is uniformly distributed over $X_\lambda$ and returns a value that belongs in $f^{-1}(f(x))$ with probability at least $p(\lambda)$ while being restricted to read at most $t(\lambda)$ symbols of its advice string and run for at most $t(\lambda)$ steps. We write $\mathcal{A}_t$ to denote $\mathcal{A}$ restricted on performing only $t$ steps of computation and reading only $t$ symbols of its advice string (for any $\lambda$). We then denote the success probability of $\mathcal{A}$, namely, $\mathbf{Pr}[\mathcal{A}_t(f(x)) \in f^{-1}(f(x))]$, by $p_{\mathcal{A},t}(\lambda)$. For simplicity, in the sequel we may drop $\lambda$ from $p_{\mathcal{A},t}(\lambda)$.

[2]As stated, the corruption protocol is an abstract version of a potentially more complex interaction where, for example, $y_i$ is the public-key of player $P_i$ and corruption takes place by getting ahold of the secret key $x_i$ via running some algorithm against $y_i$ and then authenticating on behalf of $P_i$.

We remark that the restriction on the number of steps may only bind a specific operation, which would be computational significant compared to the others (such as, for example, modular exponentiation). At times we may specify the input distribution to be other than the uniform distribution over $X$; e.g., we may consider some $X' \subseteq X$ and let $x$ be distributed uniformly over $X'$. Unless otherwise noted we will assume that the uniform over $X$ is used.

We are now ready to introduce *exact hardness*, a notion which will later on allow us to compare functions according to their inversion difficulty. In a nutshell, the exact hardness of a function, parameterized by $\epsilon$, is the number of steps that necessarily needs to be surpassed in order to achieve probability of success at least $\epsilon$. More formally:

**Definition 4.1** *For any $\epsilon \in (0,1)$ and a function $f : X \to Y$, we define the* exact hardness of $f$ with probability $\epsilon$ *to be the maximum $H \in \mathbb{N}$ such that for any $\mathcal{A}$ and $t \leq H$, it holds that $p_{\mathcal{A},t} < \epsilon$. For each $\lambda \in \mathbb{N}$, we denote the maximum such $H$ by $H_{f,\epsilon}(\lambda)$.*

Notions of similar nature have been considered in the literature; a salient difference is that exact hardness fixes $\epsilon$, and then calls for the largest possible $t$ for which the function remains hard to invert. Also, observe that, by definition, there is a (non-uniform) algorithm $\mathcal{A}$ that performs a number of steps $t = H_{f,\epsilon}(\lambda) + 1$ with a similarly sized advice string and satisfies $p_{\mathcal{A},t}(\lambda) \geq \epsilon$ for all $\lambda$. This holds since if there is no such algorithm, then $H_{f,\epsilon}$ would not be the maximum possible for that particular value of $\lambda$.

Similar notions to the notion of exact hardness that we put forth in this paper have been considered in the literature. Notably, Nissan and Wigderson [35] define the hardness of a Boolean function as the largest size of the circuit whose bias is still bounded by the inverse of its size. In our setting, we are interested in fixing the bound of success to a certain level $\epsilon$ and maximizing with respect to that (as opposed to allowing $\epsilon$ to vary as the inverse of the circuit size). Bellare and Rogaway [5] define a function to be $(t, \epsilon)$-secure if there is no "$t$-inverter" (an algorithm bounded by $t$ in number of steps and size) with success probability at least $\epsilon$. Put in this latter context, our notion of exact hardness can be defined by fixing $\epsilon$ and calling for the largest possible $t$ for which the function remains $(t, \epsilon)$-secure. Another "more refined than asymptotic" measure of inversion difficulty has been considered by Haitner, Harnik and Reingold [24] and we explore its relation to exact hardness in Appendix A.1.

Next, we show two basic properties of our notion of exact hardness. Specifically, we show that exact hardness is monotonically increasing in the probability of inversion success $\epsilon$, and that there is a natural upper bound for exact hardness that stems from the fact that any function over a finite domain can be subjected to a brute-force attack.

**Proposition 4.2** *Let $f : X \to Y$ be any function. For any $\lambda \in \mathbb{N}$ we have:*

1. *For any $0 < \epsilon \leq \epsilon'$, it holds that $H_{f,\epsilon}(\lambda) \leq H_{f,\epsilon'}(\lambda)$.*
2. *For any $\epsilon > 0$, $H_{f,\epsilon}(\lambda) \leq \lceil \epsilon \cdot |X_\lambda| \rceil$.*

*Proof.* We drop $\lambda$ for notational simplicity. For the first property, assume for the sake of contradiction that $\epsilon \leq \epsilon'$ and $H_{f,\epsilon} > H_{f,\epsilon'}$. Consider now an algorithm $\mathcal{A}$ running in $t \leq H_{f,\epsilon}$ steps. It follows that it has probability of success less than $\epsilon \leq \epsilon'$. As a result any algorithm $\mathcal{A}$ running in time at most $H_{f,\epsilon}$ has probability of success less than $\epsilon'$. This contradicts that $H_{f,\epsilon'}$ is the maximum integer with this property.

For the second property, we consider the basic step to be the operation of reading a pair $(x, y)$ and an element $y'$ from the respective tapes they reside in, and testing whether $y = y'$. Now let $z \geq 1$ be an integer function and consider a family of advice strings that contain pairs of the type

$(x_i, y_i)$, $i = 1, \ldots, z$, that belong to the graph of $f$ following a lexicographic ordering. Consider now the algorithm $\mathcal{A}$ that, given $y$, scans the advice string, and if it finds $(x_i, y_i)$ such that $y = y_i$ it returns $x_i$. It is easy to see that the number of steps that $\mathcal{A}$ takes is $z$ (in the worst case—without loss of generality we can assume that $\mathcal{A}$ always takes that many steps). The success probability of $\mathcal{A}$ is $\frac{z}{|X|}$, since the desired output $x$ is selected at random from all inputs.

Recall now that no algorithm running $H_{f,\epsilon}$ steps can have probability of success at least $\epsilon$. If, for the sake of contradiction, we assume that for sufficiently large values of the security parameter it holds that $\lceil \epsilon \cdot |X| \rceil < H_{f,\epsilon}$, it follows that $H_{f,\epsilon} \geq \lceil \epsilon \cdot |X| \rceil + 1$; thus, setting $z = \lceil \epsilon \cdot |X| \rceil + 1$, we have that $\mathcal{A}$ performs $z \leq H_{f,\epsilon}$ operations and has success probability of success $\frac{z}{|X|} > \epsilon$, which is a contradiction. □

It is worth noting that in the specification of $H_{f,\epsilon}$ there is no presumption of a "minimal hardness" for the inversion problem of the function $f$; rather, $H_{f,\epsilon}$ is meant to capture the intrinsic property of $f$ that corresponds to the minimum computational effort (measured in basic operations depending on the computational model) that is required to invert the function with a certain probability of success. If one assumes that this value is sufficiently high, then the function would be presumed to be one-way. To this end, we now show the basic relations between exact hardness and one-wayness. Specifically, for one-wayness we have that for any $\epsilon$ that is bounded away from 0 by an inverse polynomial, the exact hardness of the function to beat $\epsilon$ should exceed any polynomial function. Similarly, for *weak* one-way functions, we have that there is some threshold, which is an inverse polynomial away from 1, that in order to be reached the exact hardness should exceed any polynomial function.

**Proposition 4.3** *Let $f : X \to Y$ be a polynomial-time computable function. Then:*

1. *$f$ is a one-way function if and only if $\forall c_1, c_2 : \epsilon = \lambda^{-c_1}$ implies $H_{f,\epsilon}(\lambda) > \lambda^{c_2}$ for sufficiently large $\lambda$.*
2. *$f$ is a weak one-way function if and only if $\exists c_1\ \forall c_2 : \epsilon = 1 - \lambda^{-c_1}$ implies $H_{f,\epsilon}(\lambda) > \lambda^{c_2}$ for sufficiently large $\lambda$.*

*Proof.* The proofs of the two statements are similar so we only prove the first one. For the forward direction, we assume that the function is one-way and $\exists c_1, c_2$ for which if $\epsilon = \lambda^{-c_1}$, then $H_{f,\epsilon} \leq \lambda^{c_2}$. Then we have that there is some algorithm that runs in $\lambda^{c_2} + 1$ steps (which is a polynomial in $\lambda$) and has success probability at least $\lambda^{-c_1}$ for infinitely many $\lambda$. This contradicts one-wayness.

For the backward direction, consider an algorithm $\mathcal{A}$ that attempts to invert $f$ and runs in $\lambda^{c_2}$ steps. Also let $\lambda^{c_1}$ be any polynomial. Now suppose that the probability of inversion success is at least $\lambda^{-c_1}$ for infinitely many choices of $\lambda$ (i.e., the function $f$ fails to be one-way). This contradicts the fact that $H_{f,\epsilon} > \lambda^{c_2}$, which states that in order to reach probability of success $\lambda^{-c_1}$ one has to exceed $\lambda^{c_2}$ steps. □

It is natural to ask how easy is to calculate $H_{f,\epsilon}$ for a function $f$. Naturally, any inversion algorithm for $f$ provides an upper bound for exact hardness, while any lower bound argument on the complexity of the inversion problem of $f$ is a lower bound for $H_{f,\epsilon}$. While finding a formula for $H_{f,\epsilon}$ might be hard for a given function $f$, for certain functions in idealized computational models, such as random functions [4] or exponentiation maps in the generic group model (e.g., [40]), obtaining exact formulae is in fact possible; we provide such results in Appendix A.3. Furthermore, under cryptographic assumptions, reasonable ranges for $H_{f,\epsilon}$ can be stated; we do so for factoring-related assumptions also in Appendix A.3.

### 4.1.2 The IEP property

Next, we consider the setting where instead of just one, a *set* of functions are to be inverted, and would like, in particular, to have a measure for their "combined" hardness, as a function of the functions' individual hardnesses.

**Definition 4.4** *Let $\epsilon > 0, n \in \mathbb{N}$ and $\tau$ be a monotonically increasing function. We say a sequence of functions $\{f_i\}_{i=1,\ldots,n}$ is $\tau$-*inversion effort preserving ($\tau$-IEP) *if for any subset $S = \{i_1, \ldots, i_m\} \subseteq [n]$, it holds that $H_{f^S,\epsilon} \geq \tau(\sum_{i \in S} H_{f_i,\epsilon})$, where $f^S(x_1, \ldots, x_m) \overset{\text{def}}{=} \langle f_{i_1}(x_1), \ldots, f_{i_k}(x_m) \rangle$.*

Note that, trivially, $H_{f^{[n]},\epsilon} \geq \max_i H_{f_i,\epsilon}$, since any algorithm that inverts a sequence of instances will have to spend at least as much time as the time needed to invert the most difficult one. Nevertheless, it is not guaranteed that the work performed for the solution of any single instance cannot be used to speed up the solution of other instances. Essentially, the IEP property above says that the speed-up that can occur is lower bounded by a value that is determined by the sum of the individual exact-hardness functions, calibrated by a given function ($\tau$). In the extreme case, if $\tau$ is the identity function, then the instances have to be solved entirely independently and there is no algorithm that can proceed towards solving a subset of the instances simultaneously with a joint strategy.

Our IEP property is related to the notions of direct sum and direct product in complexity-theoretic models. A direct sum results holds in a model for a problem if solving one instance $x$ of a problem costs $c$, then when given $k$ independent instances $x_1, ..., x_k$, no significant gain is achievable and the model requires about $ck$ cost to solve all instances. A direct product result, on the other hand, holds if a fixed probability $p$ to solve a single instance correctly is given, then the probability of solving a $m$-vector of instances drops exponentially with $m$ (as in, for example, Yao's "concatenation lemma" [41]). We provide further context on this relation in Appendix A.2.

The IEP property can be proven to hold in idealized models such as the random oracle model or the generic group model (we demonstrate this in Appendix A.3); furthermore, it is reasonable to assume that it holds for standard cryptographic functions such as multiplication of primes, provided that a suitably function $\tau$ is chosen (also Appendix A.3).

### 4.1.3 Hardness indistinguishability

To introduce this notion, we define first the notion of *indistinguishability* between two functions. At a high level, it is the realization of this property that will provide the hiding of the functions' hardness, "blinding" the adversary as to what functions to attack first.

**Definition 4.5** *For $\epsilon > 0$, two functions $f_1 : X_1 \to Y_1, f_2 : X_2 \to Y_2$ are statistically indistinguishable if the random variables $f_1(x_1), f_2(x_2)$ have statistical distance less than $\epsilon$ when $x_i$ is uniformly drawn from $X_i$, for $i = 1, 2$.*

We observe that for all but at most an $\epsilon$ fraction of $y_2 \in f_2(X_2)$ it holds that there is some $x_1$ with $f_1(x_1) = y_2$. The above definition is particularly interesting to us in the setting where, say, $H_{f_1,\epsilon} < H_{f_2,\epsilon}$ for some $\epsilon$; i.e., the functions behave differently with respect to the exact hardness of the inversion problem. In such case we will talk about *hardness indistinguishability.* Given that an instance can be solved almost always in more than one way, hardness indistinguishability ensures that the hardness level can be *equivocated*[3].

[3]This property becomes handy when designing simulators for reductions between corruption models (cf. Section 4.2). We note that the reason for considering statistical as opposed to computational indistinguishability is that

The definition extends to the case of a sequence of functions in a straightforward way. We will call a sequence of functions indistinguishable when every pair of functions is indistinguishable. We give two constructions of such functions one generic and one slightly more efficient that depends on dense public-key cryptosystems [38]. Note that since indistinguishability is required in the statistical sense, there is no need to consider how the decision problem amortizes over a sequence of indistinguishable instances (as it is unattainable except with probability $\epsilon$).

Our first construction is general and relies on the existence of regular one-way functions see e.g., [22].

**Construction #1.** Let $f_1 : X_1 \to Y_1$ and $f_2 : X_2 \to Y_2$ be any two regular functions with exact hardness $H_{f_i,\epsilon}$ for $i = 1, 2$. We define the functions $f_1', f_2'$ as follows $f_1' : X_1 \times Y_2 \to Y_1 \times Y_2$ and $f_2' : Y_1 \times X_2 \to Y_1 \times Y_2$ so that $f_1'(x_1, y_2) = (f_1(x_1), y_2)$ and $f_2'(y_1, x_2) = (y_1, f_2(x_2))$. Observe that $H_{f_i',\epsilon} = H_{f_i,\epsilon}$ for $i = 1, 2$. Moreover the domains of $f_1', f_2'$ are efficiently sampleable (assuming those of $f_1, f_2$ are). The proof of the following proposition is straightforward.

**Proposition 4.6** *For any regular $f_1, f_2$, the functions $f_1', f_2'$ defined as above are (perfectly) hardness indistinguishable.*

The above approach generalizes easily to a sequence of functions (at the expense of increasing linearly the length of the domain and range elements).

**Construction #2.** A more efficient way to construct indistinguishable functions is using *dense one-way functions* along the lines of dense public-key cryptosystems of [38, 37]. Specifically, a one-way function is called dense if its output is statistically indistinguishable from a random string of a certain length. More formally, if $f$ is the function, it holds that $f(x)$ is statistically close to $\{0,1\}^k$ for some suitable $k$ when $x$ is uniformly distributed. In order to show hardness indistinguishability, given a sequence of dense one-way functions, each function output can be padded with random bits so that all of them match the length of the longest one. Based on the density property, the functions modified as above are pairwise statistically indistinguishable.

We now briefly sketch how a dense one-way function can be constructed. Given a function $f : X \to Y$ (for simplicity assume it is an injection), a dense one way function can be derived by applying a strong randomness extractor that is also collision resistant (see [15]). Specifically, if $\mathsf{Ext}$ is such an extractor, we have that $f'(r, x) = (r, \mathsf{Ext}(r, f(x)))$ is a dense one-way function. Indeed, since $\mathsf{Ext}$ is a strong extractor and $r$ is a uniformly random seed, it holds that the output of $f'$ is almost uniformly distributed. On the other hand, given the collision resistance property of $\mathsf{Ext}$ any inversion algorithm against $f'$ can be turned to an algorithm inverting $f$, hence, $f'$ is a one-way function of exact hardness not much less than that of $f$. We omit further details.

## 4.2 From Computational Corruptions to Token-based Corruptions

We now have the tools to consider the relation between the computational corruption oracle and the token-based corruption oracle. Connecting the two relies on whether the computation effort invested by the adversary against corrupting players can be abstracted as discrete token investments. The notion of IEP functions introduced above plays a crucial role here.

the former provides to the simulator the ability to "equivocate" even when the adversary is able to invert some of the functions, as we allow in our corruption model.

**Theorem 4.7** *Let $\epsilon > 0$ and $\tau$ a monotonically increasing invertible function. Given a $\tau$-IEP sequence of functions $f_1, \ldots, f_n$, we have that for any $t$ there exist $\overline{s}, k, L$ such that*

$$\mathcal{C}^{\mathsf{cc}}_{\overline{f}} \leq^t_\epsilon \mathcal{C}^{\mathsf{tk}}_{\overline{s},k} \quad \text{and} \quad \mathcal{C}^{\mathsf{bcc}}_{\overline{f}} \leq^t_\epsilon \mathcal{C}^{\mathsf{btk},L}_{\overline{s},k},$$

*where $\overline{s} = (s_1, \ldots, s_n)$ is such that $s_i = H_{f_i,\epsilon}$ for $i = 1, \ldots, n$ and $k = \lceil \tau^{-1}(t) \rceil$, and where $L(p)$ operates so that it returns $(y_{p(1)}, \ldots, y_{p(n)})$ with $y_i = f(x_i)$ and $x_i$ a randomly selected input of $f_i$.*

*Proof.* We first consider the relation between computational corruption and token-based corruption. The adversary $\mathcal{B}$ will simulate $\mathcal{A}$ as well as the corruption oracle $\mathcal{C}^{\mathsf{cc}}_{\overline{f}}$. During the simulation, $\mathcal{B}$ operates exactly as $\mathcal{A}$ with the following modification: when $\mathcal{A}$ submits a corruption request $(\textsc{Corrupt}, P_i, x)$, $\mathcal{B}$ submits $(\textsc{Corrupt}, P_i, s_i)$ to the corruption oracle.

Corruption requests $(\textsc{Corrupt}, P_i, x)$ for which it holds that $f_i(x) \neq y_i$ are ignored by $\mathcal{B}$.

The only divergence in the simulation is when it may happen that $\mathcal{A}$ issues a corruption request that is granted while the corresponding corruption request of $\mathcal{B}$ is denied. Suppose this happens with probability at least $\epsilon$. This means that $\mathcal{B}$ has used all its tokens $\lceil \tau^{-1}(t) \rceil$, i.e., the set of players $C$ corrupted by $\mathcal{A}$ satisfies that $\sum_{i \in C} s_i > \lceil \tau^{-1}(t) \rceil \geq \tau^{-1}(t)$. Note that since $\mathcal{A}$ manages to corrupt the set of players $C$ with probability at least $\epsilon$ it follows that it runs for at least $\tau(\sum_{i \in C} H_{f_i,\epsilon}) + 1$ steps due to the $\tau$-IEP property, i.e., $t \geq \tau(\sum_{i \in C} H_{f_i,\epsilon}) + 1 > \tau(\sum_{i \in C} s_i)$, which is a contradiction.

Regarding the relation of blinded computational corruption and leaky blinded token-based corruption we construct $\mathcal{B}$ as follows. When $\mathcal{B}$ receives $\overline{s}$ from its corruption oracle $\mathcal{C}^{\mathsf{btk},L}_{\overline{s},k}$ it requests to obtain the leak $(y_{p(1)}, \ldots, y_{p(n)})$ and returns this to the adversary $\mathcal{A}$. Otherwise the simulation and proof proceeds in the same fashion as before. □

We next consider the potential advantage that is given by the leaking capability of the leaky blinded-token corruption oracle. We have the following:

**Theorem 4.8** *Let $\epsilon > 0, n, k \in \mathbb{N}, \overline{s} \in \mathbb{N}^n$ be parameters. Given any sequence of statistically indistinguishable functions $\overline{f} = \langle f_1, \ldots, f_n \rangle$ we have $\mathcal{C}^{\mathsf{btk},L}_{\overline{s},k} \leq_{n \cdot \epsilon} \mathcal{C}^{\mathsf{btk}}_{\overline{s},k}$, where $L$ is defined as in Theorem 4.7 and $\epsilon$ is an upper bound on the pairwise statistical distances for the sequence $\overline{f}$.*

*Proof.* Basically, $\mathcal{B}$ will simulate $\mathcal{A}$ as well as the leak that is received by $\mathcal{A}$. The only issue in the simulation is that $\mathcal{B}$ is not privy to the permutation of its corruption oracle and it will still need to simulate the leak. $\mathcal{B}$ utilizes the sampler for the *hardest* of the $f_1, \ldots, f_n$ functions to obtain $n$ instances $y_1, \ldots, y_n$. The simulator then provides $(y_1, \ldots, y_n)$ to $\mathcal{A}$. The simulator proceeds as follows: whenever the adversary submits a corruption message to the corruption oracle with a certain number of tokens the simulator passes through this request to its own corruption oracle. Due to the fact that the sampling done is at a distance at most $\epsilon$ away from regular operation for each pair of functions the behavior of $\mathcal{A}$ cannot result in a change of more than $n \cdot \epsilon$ in the execution experiment. □

With the above results we conclude that under the proper assumptions (IEP and hardness indistinguishability) the (blinded) token-based corruption oracle is an accurate abstraction of the (blinded) computational corruption oracle. It then follows from the results of Section 3 that the blinded computational corruption oracle remains $t$-safe even for values of $t$ that far exceed the computational cost needed to corrupt a majority of player instances, thus establishing the value of the hidden diversity approach put forth in this paper.

**Acknowledgements:** The authors are grateful to Ran Gelles, Arjen Lenstra and Alexander May for valuable comments.

## References

[1] Paul Beame, Toniann Pitassi, Nathan Segerlind, and Avi Wigderson. A direct sum theorem for corruption and the multiparty nof communication complexity of set disjointness. In *IEEE Conference on Computational Complexity*, pages 52–66. IEEE Computer Society, 2005.

[2] Mihir Bellare and Shafi Goldwasser. Verifiable partial key escrow. In Richard Graveman, Philippe A. Janson, Clifford Neumann, and Li Gong, editors, *ACM Conference on Computer and Communications Security*, pages 78–91. ACM, 1997.

[3] Mihir Bellare, Thomas Ristenpart, and Stefano Tessaro. Multi-instance security and its application to password-based cryptography. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 312–329. Springer, 2012.

[4] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[5] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures - how to sign with rsa and rabin. In *EUROCRYPT*, pages 399–416, 1996.

[6] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In Janos Simon, editor, *STOC*, pages 1–10. ACM, 1988.

[7] Dan Boneh and Moni Naor. Timed commitments. In Mihir Bellare, editor, *CRYPTO*, volume 1880 of *Lecture Notes in Computer Science*, pages 236–254. Springer, 2000.

[8] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Cryptology ePrint Archive, Report 2000/067*, December 2005. Latest version at `http://eprint.iacr.org/2000/067/`.

[9] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In Miller [33], pages 639–648.

[10] Amit Chakrabarti, Yaoyun Shi, Anthony Wirth, and Andrew Chi-Chih Yao. Informational complexity and the direct sum problem for simultaneous message complexity. In *FOCS*, pages 270–278. IEEE Computer Society, 2001.

[11] David Chaum, Claude Crépeau, and Ivan Damgard. Multiparty unconditionally secure protocols. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, STOC '88, pages 11–19, New York, NY, USA, 1988. ACM.

[12] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In Juris Hartmanis, editor, *STOC*, pages 364–369. ACM, 1986.

[13] Don Coppersmith. Modifications to the number field sieve. *J. Cryptology*, 6(3):169–180, 1993.

[14] Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In Henri Gilbert, editor, *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 445–465. Springer, 2010.

[15] Yevgeniy Dodis. On extractors, error-correction and hiding all partial information. In *Theory and Practice in Information-Theoretic Security, 2005. IEEE Information Theory Workshop on*, pages 74 –79, oct. 2005.

[16] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In Ernest F. Brickell, editor, *CRYPTO*, volume 740 of *Lecture Notes in Computer Science*, pages 139–147. Springer, 1992.

[17] Tomás Feder, Eyal Kushilevitz, Moni Naor, and Noam Nisan. Amortized communication complexity. *SIAM J. Comput.*, 24(4):736–750, 1995.

[18] Ephraim Feig and Shmuel Winograd. On the direct sum conjecture. *Linear Algebra and its Applications*, 63(0):193 – 219, 1984.

[19] Matthias Fitzi, Juan A. Garay, Ueli M. Maurer, and Rafail Ostrovsky. Minimal complete primitives for secure multi-party computation. *J. Cryptology*, 18(1):37–61, 2005.

[20] Juan A. Garay, Philip D. MacKenzie, Manoj Prabhakaran, and Ke Yang. Resource fairness and composability of cryptographic protocols. *J. Cryptology*, 24(4):615–658, 2011.

[21] Juan A. Garay and Carl Pomerance. Timed fair exchange of standard signatures [Extended Abstract]. In Rebecca N. Wright, editor, *Financial Cryptography*, volume 2742 of *Lecture Notes in Computer Science*, pages 190–207. Springer, 2003.

[22] Oded Goldreich, Hugo Krawczyk, and Michael Luby. On the existence of pseudorandom generators. *SIAM J. Comput.*, 22(6):1163–1175, 1993.

[23] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In Alfred V. Aho, editor, *STOC*, pages 218–229. ACM, 1987.

[24] Iftach Haitner, Danny Harnik, and Omer Reingold. Efficient pseudorandom generators from exponentially hard one-way functions. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *ICALP (2)*, volume 4052 of *Lecture Notes in Computer Science*, pages 228–239. Springer, 2006.

[25] Martin Hirt and Ueli M. Maurer. Player simulation and general adversary structures in perfect multiparty computation. *J. Cryptology*, 13(1):31–60, 2000.

[26] W. Hoeffding. Probability inequalities for sums of bounded random variables. *J. American Stat. Ass.*, 58:13–30, 1963.

[27] Russell Impagliazzo, Ragesh Jaiswal, Valentine Kabanets, and Avi Wigderson. Uniform direct product theorems: Simplified, optimized, and derandomized. *SIAM J. Comput.*, 39(4):1637–1665, 2010.

[28] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography from anonymity. In *FOCS*, pages 239–248. IEEE Computer Society, 2006.

[29] Mauricio Karchmer, Ran Raz, and Avi Wigderson. Super-logarithmic depth lower bounds via the direct sum in communication complexity. *Computational Complexity*, 5(3/4):191–204, 1995.

[30] Arjen K. Lenstra, Hendrik W. Lenstra Jr., Mark S. Manasse, and John M. Pollard. The number field sieve. In Harriet Ortiz, editor, *STOC*, pages 564–572. ACM, 1990.

[31] Hendrik W. Lenstra. Factoring integers with elliptic curves. *The Annals of Mathematics*, 126(3):649–673, Nov. 1987.

[32] Silvio Micali and Rafael Pass. Local zero knowledge. In Jon M. Kleinberg, editor, *STOC*, pages 306–315. ACM, 2006.

[33] Gary L. Miller, editor. *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996.* ACM, 1996.

[34] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press, Cambridge, UK, 2005.

[35] Noam Nisan and Avi Wigderson. Hardness vs randomness. *J. Comput. Syst. Sci.*, 49(2):149–167, 1994.

[36] Ron Rivest, Adi Shamir, and David Wagner. Timed-lock puzzles and timed-release crypto. In *MIT/LCS/TR-684*, 1996.

[37] Alfredo De Santis, Giovanni Di Crescenzo, and Giuseppe Persiano. Necessary and sufficient assumptions for non-iterative zero-knowledge proofs of knowledge for all np relations. In Ugo Montanari, José D. P. Rolim, and Emo Welzl, editors, *ICALP*, volume 1853 of *Lecture Notes in Computer Science*, pages 451–462. Springer, 2000.

[38] Alfredo De Santis and Giuseppe Persiano. Zero-knowledge proofs of knowledge without interaction (extended abstract). In *FOCS*, pages 427–436. IEEE Computer Society, 1992.

[39] Adi Shamir. Partial key escrow: A new approach to software key escrow. In *Key Escrow Conference*, 1995.

[40] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT*, pages 256–266, 1997.

[41] Andrew Chi-Chih Yao. Theory and applications of trapdoor functions (extended abstract). In *FOCS*, pages 80–91. IEEE Computer Society, 1982.

## A Hardness Indistinguishability and Inversion-Effort-Preserving Functions (cont'd)

### A.1 Exact Hardness vis-à-vis $\mu$-Hardness

We relate the notion of exact hardness to another "more refined than asymptotic" measure of inversion difficulty that has been considered in the literature, namely, the notion of *one-wayness with hardness* $\mu$ by Haitner, Harnik and Reingold [24]. According to this definition, a function $f$ is one-way with hardness $\mu$ if for any $\mathcal{A}$ it holds that $p_{\mathcal{A}} < t_{\mathcal{A}} \cdot \mu$, where $t_{\mathcal{A}}$ is the expected running time of $\mathcal{A}$ and $p_{\mathcal{A}}$ is the probability of success without bounding the running time of $\mathcal{A}$ (here both $\mu$ and $p_{\mathcal{A}}$ are functions of $\lambda$).

At a high level the following proposition suggests that exact hardness has an inverse multiplicative relation with $\mu$-hardness.

**Proposition A.1** *Let $f : X \to Y$.*

1. *Suppose that for some $T(\lambda)$, $H_{f,\epsilon}(\lambda) \geq \epsilon \cdot T(\lambda)$ for all $\epsilon, \lambda$. Then $f$ is one-way with hardness $\frac{1}{T}$.*
2. *Suppose $f$ is one-way with hardness $\mu$. Then, for any $\lambda$, $\epsilon > \mu(\lambda)$, $H_{f,\epsilon}(\lambda) \geq \lfloor \frac{\epsilon}{\mu} \rfloor$.*

*Proof.* 1. Let $\mathcal{A}$ be any non-uniform algorithm that inverts $f$, and $\epsilon = p_{\mathcal{A},t}$ for some $t$. We know that if $t \leq H_{f,\epsilon}$ then $p_{\mathcal{A},t} < \epsilon$, so necessarily $t > H_{f,\epsilon}$, from which by the first condition of the theorem we obtain $t > \epsilon \cdot T$, i.e., $t > p_{\mathcal{A},t} \cdot T$, which implies $p_{\mathcal{A},t} < t \cdot \frac{1}{T}$. Now we have that

$$p_{\mathcal{A}} = \mathbf{Pr}[\mathcal{A} \text{ succeeds}] = \sum_t \mathbf{Pr}[\mathcal{A} \text{ succeeds in } t \text{ steps}] \cdot \mathbf{Pr}[\mathcal{A} \text{ runs in } t \text{ steps}] <$$

$$\frac{1}{T} \sum_t t \cdot \mathbf{Pr}[\mathcal{A} \text{ runs in } t \text{ steps}] = t_{\mathcal{A}} \cdot \frac{1}{T} \ .$$

This completes the proof of part 1.

2. Now suppose that $f$ is one-way with hardness $\mu$. We show that for any $\epsilon$ it holds that $H_{f,\epsilon} \geq \lfloor \epsilon \cdot \mu^{-1} \rfloor$.

Suppose, for the sake of contradiction, that there is some $\epsilon$ for which $H_{f,\epsilon} < \lfloor \frac{\epsilon}{\mu} \rfloor$ for sufficiently large $\lambda$. Let $\epsilon' = \epsilon - \mu > 0$ based on the statement of the second part of the proposition. Now observe that $H_{f,\epsilon} \leq \frac{\epsilon}{\mu} - 1 = \frac{\epsilon'}{\mu}$. Based on the property of $H_{f,\epsilon}$, there is an algorithm $\mathcal{A}$ that runs in $H_{f,\epsilon} + 1$ steps and its probability of success is at least $\epsilon$. Given that $H_{f,\epsilon} \leq \frac{\epsilon'}{\mu}$, we have that $t_{\mathcal{A}} \leq \frac{\epsilon'}{\mu} + 1$. By the property of one-wayness with hardness $\mu$ we obtain that $\epsilon \leq p_A < t_{\mathcal{A}} \cdot \mu \leq \epsilon' + \mu$; i.e., $\epsilon < \epsilon' + \mu$, which is a contradiction. □

## A.2 IEP vis-à-vis Hardness Amplification

The notion of inversion-effort-preserving sequence of functions with a certain exact hardness bears some resemblance to hardness amplification and direct-product theorems [41, 27], which essentially show that if there is a problem which is hard to solve on average, then solving multiple instances of the problem becomes even harder. In order to clarify this relation we prove in this section a lower bound on the exact hardness of $f^{[n]}$, defined as $n$ copies of a function $f$; the bound is a function of the exact hardness of a single copy of $f$. The main idea of the proof is derived from Yao's hardness amplification theorem, which highlights the relation between weak one-way and one-way functions.

At a high level, what we show is that the exact hardness of $f^{[n]}$ for a certain (low) inversion success probability $\alpha$ cannot drop much lower than a linear function of the exact hardness of a single copy of $f$ for a much higher inversion success probability $1-\delta$, with a factor which explicitly depends on $\frac{1}{\alpha}$ and $\frac{1}{\delta}$. This result by itself immediately implies the derivation of a one-way function from a sufficient number of copies of a weak one-way function. On the other hand, this type of hardness amplification cannot show an inversion-effort-preserving property for the function $f$. This is the case since for IEP we would require a provable increase of the inversion effort as the instances of the function accumulate if one wants to maintain a fixed success ratio (which is essentially a type of a generalized *direct sum* statement akin to direct sum results in communication complexity [17, 29, 10, 1]).

**Theorem A.2** *Let $0 < \alpha, \delta < \frac{1}{2}$. For any $f$, $f : X \to Y$, let $f^{[n]}$ denote its $n$-direct product. Then*

$$H_{f^{\langle n \rangle},\alpha} \geq \lfloor \frac{H_{f,1-\delta}}{m} \rfloor - n,$$

*where $n = \Omega(\frac{\log \frac{1}{\alpha}}{\delta})$ and $m = \Omega(\frac{\ln \frac{1}{\alpha} \cdot \ln \frac{1}{\delta}}{\alpha\delta})$, in the computational model where sampling from $X$ and testing whether $f(x) = y$ for a given candidate pair $(x, y)$ costs one step.*

*Proof.* Let $f^{[n]}$ denote the parallel repetition of $f$, $n$ times where $n$ is a parameter to be determined.

Consider any algorithm $\mathcal{A}$ inverting $f^{[n]}$ and without loss of generality suppose that $\mathcal{A}$ returns a vector $(x_1, \ldots, x_n)$ always. Let $S_{\mathcal{A}}$ be the event that $\mathcal{A}$ succeeds and $\alpha = \mathbf{Pr}[S_{\mathcal{A}}]$. For each $i = 1, \ldots, n$, we define a subset $G_i$ of $X$ as follows: $x \in G_i$ iff $\mathbf{Pr}[S_{\mathcal{A}} | x_i = x] \geq \frac{\alpha}{2n}$ (so one can think of $G_i$ as the "good elements" of the $i$-th coordinate). Now suppose that for all $j = 1, \ldots, n$ it holds that $|G_j| < (\frac{\alpha}{2})^{1/n} \cdot |X|$. We observe that $\mathbf{Pr}[S_{\mathcal{A}} | x_i \notin G_i] \leq \max_{x \notin G_i} \mathbf{Pr}[S_{\mathcal{A}} | x_i = x] < \frac{\alpha}{2n}$ (due to the fact that $\mathbf{Pr}[A | B \cup C] \leq \max\{\mathbf{Pr}[A|B], \mathbf{Pr}[A|C]\}$ for mutually exclusive $B, C$). Given this we have,

$$\begin{aligned} \mathbf{Pr}[S_{\mathcal{A}}] &\leq \mathbf{Pr}[S_{\mathcal{A}} \wedge (\wedge_{i=1}^{n} x_i \in G_i)] + \sum_{i=1}^{n} \mathbf{Pr}[S_{\mathcal{A}} \wedge x_i \notin G_i] \\ &\leq \mathbf{Pr}[\wedge_{i=1}^{n} x_i \in G_i] + \sum_{i=1}^{n} \mathbf{Pr}[S_{\mathcal{A}} | x_i \notin G_i] \\ &< ((\frac{\alpha}{2})^{1/n})^n + \frac{\alpha}{2} = \alpha \end{aligned}$$

The above is a contradiction, thus we can deduce that there is a $j$ such that $|G_j| \geq (\frac{\alpha}{2})^{1/n} |X|$.

It follows that there exists a (non-uniform) algorithm $\mathcal{A}'$ that inverts $f$: the advice of $\mathcal{A}'$ is the value $j$. Suppose $m$ is a parameter to be determined later. Given $y$, $\mathcal{A}'$ proceeds by randomly sampling $x_i^l$ for $i \in [n] \setminus \{j\}$ and $l = 1, \ldots, m$, and forms instance $y^l = (y_1^l, \ldots, y_n^l)$ by setting $y_j^l = y$

and $y_i^l = f(x_i^l)$ for $i \neq j$. Then, $\mathcal{A}'$ simulates $\mathcal{A}(y^l)$, for $l = 1, \ldots, m$, and returns the value $x_j^l$ if $l$ is found with $f(x_j^l) = y$. We use $S_{\mathcal{A}'}$ to denote the success probability of $A'$ overall, and $S_{\mathcal{A}'}^l$ the success probability in experiment $l = 1, \ldots, m$. We have:

$$\mathbf{Pr}[S_{\mathcal{A}'}] \geq \mathbf{Pr}[\vee_{l=1}^m S_{\mathcal{A}'}^l | x_j \in G_j] \cdot \mathbf{Pr}[x_j \in G_j] \geq (1 - (1 - (\frac{\alpha}{2n})^m) \cdot (\frac{\alpha}{2})^{1/n}$$

To make the above probability greater than or equal to $1 - \delta$, we proceed as follows:

1. We choose $n$ such that $(\frac{\alpha}{2})^{1/n} \geq 1 - \frac{\delta}{2}$, which follows from the fact that $\frac{\alpha}{2} \geq e^{-\delta n/2}$, i.e., $\ln \frac{\alpha}{2} \geq -\frac{\delta n}{2}$, which in turn implies $n \geq \frac{\ln(4\alpha^{-2})}{\delta}$, and
2. we choose $m$ such that $(1 - (1 - (\frac{\alpha}{2n})^m) \geq 1 - \frac{\delta}{2}$, which is equivalent to $(1 - \frac{\alpha}{2n})^m \leq \frac{\delta}{2}$ and implied by $e^{-\alpha \cdot m/2n} \leq \frac{\delta}{2}$, i.e., $m \geq \frac{2n \ln \frac{2}{\delta}}{\alpha}$.

As a result from the above arguments we have that for any algorithm $\mathcal{A}$ that inverts $f^{[n]}$ in $q$ steps with probability $\alpha$, we can construct an algorithm $\mathcal{A}'$ that inverts $f$ in $mq + nm$ steps with probability $1 - \delta$ where $n = \Omega(\frac{\log \frac{1}{\alpha}}{\delta})$ and $m = \Omega(\frac{\ln \frac{1}{\alpha} \cdot \ln \frac{1}{\delta}}{\alpha \delta})$. Note that a single"step" is assumed to be sufficient to sample an input from the domain of $f$.

By definition we know that there is a (non-uniform) algorithm that inverts $f^{[n]}$ in $H_{f^{[n]},\alpha}+1$ steps with probability $\alpha$. It then follows that there is an algorithm that inverts $f$ in $m \cdot H_{f^{[n]},\alpha} + nm.$ steps with probability $1 - \delta$, i.e., $H_{f,1-\delta} < m \cdot H_{f^{[n]},\alpha} + nm$. We conclude that $H_{f^{[n]},\alpha} > H_{f,1-\delta}/m - n - 1$ and the theorem follows. ☐

### A.3 Candidate Functions

In this section we study and give bounds for the exact hardness, the IEP property and hardness indistinguishability of several candidates, including random functions, discrete logarithm in the generic group model, and factorization. We note that all the functions we present here can be used in conjunction with constructions #1 and #2 given above to derive hardness indistinguishable functions.

#### A.3.1 Random functions

We first consider the notion of exact hardness and argue that for the inversion problem of a function that is modeled as a random oracle [4], we can calculate exactly the expression for exact hardness. In this model, we only count queries to the random oracle as the basic computational operation, and thus all complexity measures are expressed in terms of such queries.

**Proposition A.3** *Let $\epsilon > 0$ and $f : [2^\lambda] \to [2^\lambda]$ be modelled as a random oracle. Then $H_{f,\epsilon}(\lambda) = \epsilon \cdot 2^\lambda - 1$.*

*Proof.* When $f$ is modelled as a random oracle the probability $\mathbf{Pr}[\mathcal{A}_t(f(x)) \in f^{-1}(f(x))]$ is taken over all possible choices of $x$ (uniformly selected from $[2^\lambda]$) and the choices of $f$ (thought as a random table with $2^\lambda$ entries). Any algorithm that queries $f$ will have probability $(q+1)2^{-\lambda}$ of returning the inverse of the input value $y = f(x)$ if it is allowed $q$ queries. To see this consider the following: if $\mathcal{A}$ asks $q$ distinct queries and finds the inverse among them it can return it for a probability of success equal to 1; otherwise, any $\mathcal{A}$ no matter which strategy it follows will have at best a $\frac{1}{n-q}$ chance of succeeding. Note that if $\mathcal{A}$ is repeating some queries the probability of success would be worse, but this is something we can assume without loss of generality that it does not happen given that there is no restriction in space for $\mathcal{A}$. The statement of the proposition now follows since the probability of success is $q2^{-\lambda} + (1 - q2^{-\lambda})(1/(2^{\lambda} - q)) = (q+1)2^{-\lambda}$ by requiring this probability to be less than $\epsilon$ and considering a query to $f$ as the basic computational step. $\square$

We now study the inversion-effort-preserving property of random oracles. We show that if the functions are modeled as independent random oracles, then no speed up is possible for such set of instances.

**Proposition A.4** *For $n > 0$, the sequence of functions $\{f_i\}_{i=1,\dots,n}$ with $f_i : [2^{\lambda}] \to [2^{\lambda}]$ is IEP assuming each one of them is modelled as a random oracle.*

*Proof.* We will use the fact that querying function $f_i$ gives no information about function $f_j$ for $j \neq i$. As in Proposition A.3, the inversion of function $f_i$ can succeed with probability $(q_i + 1)2^{-\lambda}$ as long as $q_i$ queries are made. Also as before, any algorithm inverting $f_i$ with probability $\epsilon$ requires $\epsilon \cdot 2^{\lambda} - 1$ queries to $f_i$. We will show that any algorithm running in $t = n\epsilon \cdot 2^{\lambda} - n$ steps cannot invert all of the functions with probability greater than $\epsilon$. We argue as follows. At the end of any execution at the specified number of steps $t$ we can build a table of size $q_i$ for each function $f_i$; we have that $\sum_{i=1}^{n} q_i \leq t$. By conditioning on the subset $A$ of the functions for which a good query has been made, we obtain that the probability of success can be upper-bounded by the expression

$$\sum_{A \subseteq [n]} \prod_{i \in A} \frac{q_i}{2^{\lambda}} \cdot \prod_{i \notin A} (1 - \frac{q_i}{2^{\lambda}}) \cdot \prod_{i \notin A} \frac{1}{2^{\lambda} - q_i} = \sum_{A \subseteq [n]} \frac{\prod_{i \in A} q_i}{2^{\lambda n}} = \frac{\prod_{i=1}^{n} (q_i + 1)}{2^{\lambda n}} \leq \left( \frac{n + \sum_{i=1}^{n} q_i}{n \cdot 2^{\lambda}} \right)^{n}$$

where the last inequality follows from the arithmetic and geometric means inequality. Now applying the bound $\sum_{i=1}^{n} q_i \leq t = n\epsilon \cdot 2^{\lambda} - n$ and the fact that $\epsilon^n \leq \epsilon$ we bound the probability by $\epsilon$. The result on $H_{f^S,\epsilon}$ (cf. Definition 4.4) now immediately follows.

$\square$

### A.3.2 Discrete logarithm

We now consider the exact hardness of discrete logarithms in the generic group model (see, e.g., [40]). In this setting, any algorithm $\mathcal{A}$ solving the discrete logarithm is given the encodings of two group elements, $\sigma(1)$ and $\sigma(x)$, where $\sigma : \mathbb{Z}_q \to S$ is a random permutation. Here $q$ is a prime number that is $\lambda$-bits.

$\mathcal{A}$ aims to discover $x$ but has no way to internally emulate the group operation. Instead, the algorithm operates with access to an oracle that takes input $(r, s)$ and returns the value $\sigma(r + sx)$; we measure those oracle calls as the basic steps taken by the algorithm.

We denote all the queries of $\mathcal{A}$ to the oracle by $Q_{\mathcal{A}}$. A key observation used in [40] is that as long as it holds that $r + sx = r' + s'x \bmod q$ for two queries $(r, s), (r', s')$ to the oracle, it is possible to reconstruct $x$. The second key observation is that if this event does not occur then $\mathcal{A}$'s behavior is independent of $x$. It follows that the success of algorithm $\mathcal{A}$ in solving the discrete logarithm problem is bounded by the probability that $r + sx = r' + s'x$ or $y = x$ for some $(r, s), (r', s') \in Q_{\mathcal{A}}$, where $x$ is uniformly distributed over $\mathbb{Z}_q$ and $y$ is some arbitrarily distributed value (in this setting, $y$ would capture the output of $\mathcal{A}$). If the algorithm performs $k$ queries then we have that the probability of success is bounded by $(\binom{k}{2} + 1)/q$ (this stems from the fact that the $k$ queries can be thought of lines over a plane that determine $\binom{k}{2}$ cut points—the probability of equality amounts to hitting one of those cut points). Based on this we state the following.

**Proposition A.5** *Let $\epsilon > 0$ and $q$ be a $\lambda$-bit prime number. Suppose $f : \mathbb{Z}_q \to S$ is the exponentiation function over a generic multiplicative group. Then $H_{f,\epsilon} \geq \sqrt{2q\epsilon}$.*

*Proof.* The proof follows from the argument presented above (due to [40]). Suppose $H_{f,\epsilon} < T = \sqrt{2q\epsilon}$. This means that there is an algorithm that in $T$ steps achieves a probability of success $\epsilon$. But we observe that $T$ is such that $(\binom{T}{2} + 1)/q < \epsilon$, hence a contradiction following the above arguments (assuming $T \geq 2$). $\square$

Next, we consider the IEP property in the same setting. We consider the solution of $n$ independent instances of the discrete-logarithm problem over $n$ groups. Each group is assumed to have a separate encoding of elements $\sigma_i$ that is independently selected. Specifically, an algorithm $\mathcal{A}$ is given the pairs $(\sigma_i(1), \sigma(x_i))$ for $i = 1, \ldots, n$ and aims to produce the vector $(x_1, \ldots, x_n)$. $\mathcal{A}$ is allowed to access an oracle that accepts queries of the form $(i, r, s)$ and returns $\sigma_i(r + sx_i)$.

**Proposition A.6** *Let $\epsilon > 0$, $n > 0$ and $q$ be a prime $\lambda$-bit number. Suppose $f^{[n]} : (\mathbb{Z}_q)^n \to S_1 \times \ldots \times S_n$ is the coordinate-wise exponentiation function over a sequence of $n$ generic multiplicative groups. Then $H_{f^{[n]},\epsilon} \geq \sqrt{2n \cdot \epsilon^{1/n} \cdot q}$.*

*Proof.* Suppose that $H_{f^{(n)},\epsilon} < T = \lceil \sqrt{2n\epsilon^{1/n}q} \rceil$. This means that there is an algorithm $\mathcal{A}$ that in $T$ steps has probability of success at least $\epsilon$. Define by $t_i$ the number of queries posed to the $i$-th oracle by $\mathcal{A}$. Then it holds that $\sum_{i=1}^{n} t_i = T$ since we count only oracle queries in the running time of $\mathcal{A}$. We let $C_i$ be the event that a collision has occurred at the $i$-th oracle. Now let $B \subseteq [n]$ be the set of groups over which the event $C_i$ takes place, i.e., for all $i$, $i \in B$ if and only if $C_i$ is true. Assuming this conditioning we know that the probability of success of algorithm $\mathcal{A}$ is at most $(1/q)^{n-|B|}$ since it essentially has to guess all the other queries. Furthermore, the probability of event $B$ happening is the probability that we get a collision; given the independence of the group encodings this is $\prod_{i\in B} \binom{t_i}{2}/q$. So overall we have that the probability is bounded by

$$q^{-n} \cdot \sum_{B\subseteq[n]} \prod_{i\in B} \binom{t_i}{2} = q^{-n} \cdot \prod_{i=1}^{n} (\binom{t_i}{2} + 1) \leq (n \cdot q)^{-n} (n + \sum_{i=1}^{n} \binom{t_i}{2})^n,$$

where the last inequality follows from the geometric-arithmetic means inequality. Observe now that $\sum_{i=1}^{n} \binom{t_i}{2} = \frac{1}{2} \cdot (\sum_{i=1}^{n} t_i^2 - T) < T(T-1)/2$. From this we obtain that the probability of success $\epsilon$ is strictly bounded by $(T^2/2qn)^n$. This is a contradiction since $(T^2/2qn)^n \geq \epsilon$ by definition of $T$. $\square$

**Corollary A.7** *Let $\epsilon > 0, n > 0$, and $q$ be a $\lambda$-bit prime number. Suppose $f_i : \mathbb{Z}_q \to S_i$ is the exponentiation function over a generic multiplicative group $S_i$. Then the set of functions $\{f_i\}_{i=1,\ldots,n}$ is $\tau$-IEP for $\tau(\cdot) = (\cdot)^{1/2}$.*

*Proof.* By definition we need to show that for any subset $S \subseteq [n]$ we have that $H_{f^S,\epsilon} \geq \tau(\sum_{i\in S} H_{f_i,\epsilon})$. Using Propositions A.5 and A.6 we have that

$$H_{f^S,\epsilon} \geq \sqrt{|S|} \cdot \sqrt{2q\epsilon^{1/n}} \geq \sqrt{|S|} \cdot \max_{i\in S} H_{f_i,\epsilon} \geq (\sum_{i\in S} H_{f_i,\epsilon})^{1/2},$$

which completes the proof. $\square$

### A.3.3 Factoring

The complexity to factor an integer $N$ with the best known algorithm (the number field sieve [30]) takes time on the order of $L[1/3, 1.9229]$, where

$$L[\alpha, \beta] = \exp((\beta + o(1))(\log N)^{\alpha}(\log \log N)^{1-\alpha}),$$

and was improved to $L[1/3, 1.902]$ by Coppersmith [13]. Assuming that this result matches the complexity of the factoring problem, this will provide bounds for the exact hardness of the multiplication function. Specifically, if $\mathsf{P}_\lambda$ is the set of all $\lambda$-bit prime numbers, we have that if $f_{\mathsf{mult}} : \mathsf{P}_\lambda \times \mathsf{P}_\lambda \to \mathbb{N}$ with $f_{\mathsf{mult}}(p, q) = p \cdot q$, then for any $\epsilon$,

$$\epsilon \cdot L(1/3, 1.902) \leq H_{f,\epsilon} \leq \epsilon \cdot L(1/2, \sqrt{2}) \tag{1}$$

According to the above statement, the exact hardness of factoring is a subexponential function that is bounded from below by a time complexity that is derived from the running time of the number field sieve algorithm. We note that a similar assumption was made in [5] when stating the "exact security" of the RSA function. The upper bound in the statement is derived from Lenstra's elliptic curve factorization algorithm [31]. This algorithm is suitable for expressing an upper bound in the form above since it is an algorithm that repetitively picks an elliptic curve and a point on it, and then tests whether the group law holds by calculating a scalar of the point. The complexity of the algorithm is determined by the fact that after picking around $L(1/2, \sqrt{2})$ curves, then with very high probability a "bad" curve will be found. As a result, if we wish to succeed with probability $\epsilon$, we should sample about $\epsilon \cdot L(1/2, \sqrt{2})$ curves.

Regarding the IEP property, it was shown in [13] that it is possible to amortize cost when factoring $n$ integers and there exists an algorithm with expected time complexity $L[1/3, 2.0068] + n \cdot L[1/3, 1.6386]$. Under the assumption that Coppersmith's algorithm (and small optimizations thereof) is the best possible when trying to factor simultaneously a sequence of moduli, one might be willing to assume that if $f^{[n]}$ is the function that multiplies $n$ pairs of primes to the corresponding moduli, the following lower bound would hold true:

$$H_{f^{[n]},\epsilon} \geq \epsilon \cdot n \cdot L[1/3, \sqrt{2}]. \tag{2}$$

From the above two assumptions, the $\tau$-IEP property for factoring is satisfied by letting $\tau(x) = e^{(\ln x)^{2/3}}$. This holds due to the fact that for any $n, \epsilon$, we have that $\epsilon \cdot n \cdot L(1/3, \sqrt{2}) \geq \tau(\epsilon \cdot n \cdot L(1/2, \sqrt{2}))$.

## B The Combinatorics of Hidden Diversity (cont'd)

### B.1 Preliminaries

In this Appendix, we study the following "balls in buckets" problem. Suppose there is a sequence $B_1, B_2, \ldots, B_n$ of buckets having integer sizes $s_1, s_2, \ldots, s_n$, respectively. For a given *target fraction* $\alpha$, $0 < \alpha < 1$, our goal is to sequentially place balls in buckets until at least $\lceil \alpha n \rceil$ buckets are full, so as to minimize the number of balls used, which we shall denote by $OPT_\alpha(I)$ for a given instance $I$. The purpose of the combinatorial game is to capture the objective of an adversary trying to corrupt an $\lceil \alpha \cdot n \rceil$ number of players against the token-based corruption oracle that implements the instance $I$. For conveniencein our exposition below we will write from the point of view of the adversary.

If we knew the size of each bucket, we could obtain an optimal assignment, simply by filling the buckets in order of increasing size until the desired number had been filled. Here we consider the case where, although we know $n$ and $\alpha$, we do not know the specific bucket sizes $s_i$, and when we place a ball in bucket $B_j$, we only learn whether or not the bucket $B_j$ is now full. (If a bucket has size 0, then it is full to begin with, and we know this before we start placing balls.)

Here we study what can be done under four variants of incomplete information. In order of increasing knowledge, these are:

1. We know nothing at all about the bucket sizes [*No-Information*].
2. We know the maximum bucket size [*Max-Only*].
3. We know the sizes $s_1 \leq s_2 \leq \cdots \leq s_m$ that occur in the instance [*Sizes-Only*].
4. We know the *profile* of the sizes: the size list as above, and, for each size, $s_i$, the number $k_i$ of buckets that have that size [*Known-Profile*].

The terminology in this appendix differs from that in the body of the paper, in that here we speak of an instance $I$ having "profile" $\overline{s} = s_1, s_2, \ldots, s_n$, rather than simply letting the vector $\overline{s}$ itself be the instance. The blinded-token model in the body of the paper corresponds to the Known-Profile situation here, when the buckets undergo an initial random permutation before the algorithm (adversary) begins placing balls.

In the results that follow, we provide both algorithmic performance guarantees and lower bounds on the best that any algorithm can achieve. The algorithmic results, although positive in the ball-packing context, are actually negative when viewed in the context of our cryptographic application, as they show the power of a would-be corrupter. Similarly, our lower bounds can be viewed as positive results for our cryptographic application, corresponding to proofs of security.

We also note that all the algorithms we study apply in the No-Information, Max-Only, or Sizes-Only situations, but match, to varying extents, our lower bounds, which all hold even in the Known-Profile case. This suggests that the general algorithmic advantage to knowing the full profile may not be substantial.

In what follows, we begin with the case where there are no restrictions on the input (and little in the way of good algorithmic performance), and then consider the effects of placing reasonable on the number of different item sizes, the number of buckets, and the maximum bucket size. We shall then consider alternative objective functions, such as filling as many buckets as possible for a given number of balls (in the presence of hidden diversity), and maximizing the number of balls needed subject to a budget when increasing the size of a bin has a cost.

## B.2 Unrestricted Instances

Our first observation is that, even when we know the full profile, no fixed multiple of the optimal number of balls suffices to guarantee even a small probability of success if there are no restrictions on the input.

**Theorem B.1** *For any $\alpha$, $0 < \alpha < 1$, and constants $B > 1$ and $\epsilon > 0$, there exists an instance $I$ such that, assuming we first use a random permutation to relabel the buckets, any algorithm that knows nothing more than the profile of the instance, and has fewer than $B \cdot OPT_\alpha(I)$ balls, has probability less than $\epsilon$ of filling $\lceil \alpha n \rceil$ buckets.*

**Proof.** Our instance will have $n = \left\lceil ((2B/\epsilon) + 1)/(1-\alpha) \right\rceil$ buckets. Of these buckets, the $\lceil \alpha n \rceil$ with smallest capacity will consist of $\lceil \alpha n \rceil - 1$ buckets of size 1 and one bucket of size $X = \lceil \alpha n \rceil + 1$. An optimal solution will thus be simply to fill these $\lceil \alpha n \rceil$ buckets, requiring $OPT_\alpha(I) = 2\lceil \alpha n \rceil$ balls. The remaining $n - \lceil \alpha n \rceil$ buckets will all have capacity $M = 2B\lceil \alpha n \rceil$.

Now let $A$ be any algorithm that has access to fewer than $B \cdot OPT_\alpha(I) = 2B\lceil \alpha n \rceil$ balls. Clearly the algorithm cannot fill any of our size-$M$ buckets, so it will have to fill all the $\lceil \alpha n \rceil$ buckets of the optimal solution, and, in particular, it must fill the bucket of size $X$. Consider the buckets of size $X$ or greater, and let $m$ be the number of such buckets. Given the hidden depth of the buckets, the only way the algorithm can tell whether one of these buckets is the one that has size $X$ is to place $X$ balls in it. So it can test at most $2B-1$ buckets. Given that the identities of the buckets have been randomly permuted, the probability that the algorithm succeeds in finding the bucket of size $X$, and hence has any chance of filling $\lceil \alpha n \rceil$ buckets, is no more than $(2B-1)/m$, which will be less than $\epsilon$ so long as $m > (2B-1)/\epsilon$. But now note that

$$m = n - \lceil \alpha n \rceil \geq n(1-\alpha) - 1 = \left\lceil \frac{\frac{2B}{\epsilon}+1}{1-\alpha} \right\rceil (1-\alpha) - 1 \geq \frac{2B}{\epsilon}$$

As desired. □

Using essentially the same examples, parameterized by $n$ instead of $\epsilon$, we get the following alternative statement of the theorem.

**Corollary 1** *For any $\alpha$, $0 < \alpha < 1$, and constant $B > 1$, there exists a sequence of instances $I_n$ such that, assuming we first use a random permutation to relabel the buckets, any algorithm that knows nothing more than the profile of the instance, and has fewer than $B \cdot OPT_\alpha(I_n)$ balls, has $O(1/n)$ probability of filling $\lceil \alpha n \rceil$ buckets.*

By setting $\epsilon = 1/2$ and replacing "$B$" by "$2B$"in the proof of Theorem B.1, we obtain the following.

**Corollary 2** *For any $\alpha$, $0 < \alpha < 1$, and constant $B > 0$, there exists an instance $I$ of our balls-in-buckets problem such that, assuming we first use a random permutation to relabel the buckets, the expected number of balls used by any algorithm that knows nothing more than the profile of the instance is at least $B \cdot OPT(I)$ balls.*

The proof of Theorem B.1 also implies that, if we are willing to settle for small penalties and non-minuscule $\epsilon$, neither $n$ nor the maximum bucketsize need be impractically large. For instance, suppose $\alpha = 1/2$ and we only want hidden diversity to cause us a factor-of-two ball penalty with probability 1/3. Then the relevant instance is $I_{2,1/3}$, for which $n = \lceil (2 \cdot 2 \cdot 3 + 1)/(1/2) \rceil = 26$ and $M = \lceil n/2 \rceil + 1 = 14$.

In the proof of Theorem B.1, our constructions only let us claim that, for fixed $B$ and $\alpha$, the error probability is $O(1/n)$. Ideally, we would like error probabilities that decline exponentially in $n$. A modification of our constructions allows us to do this. (The theorem was already presented in Section 3; here we repeat statement and proof for convenience.)

**Theorem 3.2** *For any $\alpha$, $0 < \alpha < 1$, and constant $B > 1$, there exists a constant $a < 1$ and instances $I_n$, $n > 8B/(1-\alpha)$, such that, assuming we first use a random permutation to relabel the buckets, any algorithm that knows nothing more than the profile of the instance, and has fewer than $B \cdot OPT_\alpha(I)$ balls, has probability less than $a^n$ of filling $\lceil \alpha n \rceil$ buckets.*

**Proof.** For a $c$, $0 < c \leq \alpha$, to be specified later, instance $I_n$ will have $\lceil cn \rceil$ buckets of size $\lceil \alpha n \rceil + 1$, $\lceil \alpha n \rceil - \lceil cn \rceil$ buckets of size 1, and $n - \lceil \alpha n \rceil$ buckets of size $(\lceil cn \rceil + 2)B\lceil \alpha n \rceil$. An optimal solution will consist of the $\lceil \alpha n \rceil$ smallest buckets, and will have total size $(\lceil cn \rceil + 1)\lceil \alpha n \rceil$. This implies that we cannot afford to fill any of the largest buckets if we are to use fewer than

$B \cdot OPT(I_n) = B(\lceil cn \rceil + 1)\lceil \alpha n \rceil$ balls, and any algorithm that fills $\lceil \alpha n \rceil$ buckets must find and fill all the buckets of size $\lceil \alpha n \rceil + 1$.

But now note that the only way the algorithm can tell whether a bucket has this size or is one of the larger ones is to place $\lceil \alpha n \rceil + 1$ balls in the bucket. Thus, if our algorithm is to use no more than $B(\lceil cn \rceil + 1)\lceil \alpha n \rceil$ balls, it can test no more than $B(\lceil cn \rceil + 1)$ buckets with size exceeding 1, of which there are $n - \lceil \alpha n \rceil + \lceil cn \rceil$. The probability of finding all $\lceil cn \rceil$ of the mid-size bucket is then at most

$$\binom{n - \lceil \alpha n \rceil}{B(\lceil cn \rceil + 1) - \lceil cn \rceil} \Big/ \binom{n - \lceil \alpha n \rceil + \lceil cn \rceil}{B(\lceil cn \rceil + 1)}$$

Setting $X = n - \lceil \alpha n \rceil$ and $Y = B(\lceil cn \rceil + 1) - \lceil cn \rceil$, this is equal to

$$\binom{X}{Y} \Big/ \binom{X + \lceil cn \rceil}{Y + \lceil cn \rceil} \quad = \quad \frac{(Y+1)(Y+2)\cdots(Y+\lceil cn \rceil)}{(X+1)(X+2)\cdots(X+\lceil cn \rceil)},$$

which we will be able to argue is sufficiently small if we can chose $c > 0$ such that, say, $Y + \lceil cn \rceil \le (X + \lceil cn \rceil)/2$. This requires that $B\lceil cn \rceil + B < (n - \lceil \alpha n \rceil + \lceil cn \rceil)/2$, or $(2B-1)\lceil cn \rceil \le n - \lceil \alpha n \rceil - 2B$. which will be true so long as $(2B-1)(cn+1) \le n(1-\alpha) - 1 - 2B$, or $(2B-1)(cn) \le n(1-\alpha) - 4B$, or $c \le (1-\alpha)/(2B-1) - 4B/(n(2B-1)$. By our assumption that $n > 8B/(1-\alpha)$, this means that all we need for $Y + \lceil cn \rceil < (X + \lceil cn \rceil)/2$ is that $c \le (1-\alpha)/(4B-2)$. By our construction, we also need $c \le \alpha$, so it suffices to set $c = \min\{\alpha, (1-\alpha)/(4B-2)\}$.

Given that $Y + \lceil cn \rceil < (X + \lceil cn \rceil)/2$, we then have that $(Y+i)/(X+i) \le 1/2$ for $1 \le i \lceil cn \rceil$, and hence

$$\frac{(Y+1)(Y+2)\cdots(Y+\lceil cn \rceil)}{(X+1)(X+2)\cdots(X+\lceil cn \rceil)} < \left(\frac{1}{2}\right)^{\lceil cn \rceil} \le \left(\frac{1}{2}\right)^{cn}$$

which is less than $a^n$ for $a = (1/2)^c < 1$, as desired. □

From the above results, we know that, if we place no restrictions on our instances, then hidden diversity imposes a penalty factor that is unbounded. Thus the only situations in which the performance penalty for hidden diversity can be bounded will be ones in which we restrict the instances, for example by bounding $n$ or the maximum bucket size, or by limiting the number of distinct bucket sizes. Note, however, that the proofs of Theorems B.1 and 3.2 imply that we cannot avoid unbounded performance penalty as soon as there are three or more distinct sizes.

## B.3 Restricted Instances and No-Information Algorithms

In what follows, we shall assume that the greatest common divisor $\Delta$ of the bucket sizes in our instances is 1 (as for instance will happen when there are buckets of size 1 themselves). It is easy to see that if $\Delta > 1$, then any algorithm can be modified to place $\Delta$ balls at once and will have the same relative performance in comparison to an optimal algorithm (and that there is no advantage to doing anything differently). We shall denote the maximum bucket size under this assumption as $M$.

In what follows, we shall consider not only lower bounds on the performance penalties for *all* algorithms, but also how well specific natural algorithms perform. When talking about specific algorithms, we will be interested in two worst-case measurements. For a given $\alpha$, an algorithm $A$, and an $n$-bucket instance $I$, let $A_\alpha(I)$ be the number of balls used by algorithm $A$ to fill $\lceil \alpha n \rceil$

buckets in instance $I$. Suppose $0 < \alpha < 1$, $M \geq 1$, and $n > 0$. If $dA$ is a deterministic algorithm. define

$$R^{dA}_{\alpha,M,n} = \max\left\{\frac{A_\alpha(I)}{OPT_\alpha(I)} : I \text{ is an instance with } n \text{ buckets, target fraction } \alpha\text{, and } s_m \leq M\right\}$$

If $rA$ is a randomized algorithm, define

$$R^{rA}_{\alpha,M,n} = \max\left\{\frac{E[A_\alpha(I)]}{OPT_\alpha(I)} : I \text{ is an instance with } n \text{ buckets, target fraction } \alpha\text{, and } s_m \leq M\right\}$$

Then define

$$\begin{aligned} R^{A}_{\alpha,\infty,n} &= \limsup_{M\to\infty} R^{A}_{\alpha,M,n} \\ R^{A}_{\alpha,M,\infty} &= \limsup_{n\to\infty} R^{A}_{\alpha,M,n} \end{aligned}$$

In this section, we first discuss a sequence of algorithms and how they perform when either $n$ or $M$ is bounded, including a hybrid of the two tailored to the case when neither is bounded, but we are guaranteed that there are at most two distinct sizes. We begin with algorithms that operate in the No-Information domain and, are building blocks for later, more effective algorithms that follow and exploit the additional information available in our other cases. We present our lower bound proofs immediately following the algorithmic results they match.

### B.3.1 The *Fill* Algorithm

The algorithm "Fill" works by repeatedly choosing an empty bucket and adding balls to that bucket until it is full. Let $dFill$ represent any version of the algorithm that chooses the next bucket according to some deterministic policy, and $rFill$ be the version that chooses the next bucket randomly from the set of currently empty buckets, with all choices equally likely. It is easy to see that the following holds.

**Theorem B.2**

1. *For all $M > 0$,*

$$\begin{aligned} R^{dFill}_{\alpha,M,\infty} &= \begin{cases} M, & 0 < \alpha \leq \frac{1}{2} \\ \frac{1-\alpha}{\alpha}M + 2 - \frac{1}{\alpha}, & \frac{1}{2} < \alpha < 1 \end{cases} \\ R^{rFill}_{\alpha,M,\infty} &= \alpha + (1-\alpha)M, \quad 0 < \alpha < 1 \end{aligned}$$

2. *For all $\alpha$, $0 < \alpha < 1$, and all $n > 1/(1-\alpha)$,*

$$R^{dFill}_{\alpha,\infty,n} = R^{rFill}_{\alpha,\infty,n} = \infty.$$

Note that for $\alpha = 1/2$, claim (1) reduces to $R^{dFill}_{\alpha,M,\infty} = M$ and $R^{rFill}_{\alpha,M,\infty} = (M+1)/2$.

**Proof.** For the cases where $M$ is bounded, it is easy to see that a worst-case example $I_n$ for $n$ buckets consists of $\lceil \alpha n \rceil$ buckets of size 1 and the rest of the buckets having capacity $M$. The bounded-$M$ result for $\alpha \leq 1/2$ is trivial. For $\alpha > 1/2$, $OPT_\alpha(I_n) = \lceil \alpha n \rceil$ and

$$dFill_\alpha(I_n) = (n - \lceil \alpha n \rceil)M + \lceil \alpha n \rceil - (n - \lceil \alpha n \rceil) = (n - \lceil \alpha n \rceil)M + 2\lceil \alpha n \rceil - n.$$

Dividing by $OPT_\alpha(I_n)$ and taking the limit as $n \to \infty$ gives the claimed bound. The result for $rFill$ follows from the fact that the expected size of a randomly chosen bucket is $\lceil \alpha n \rceil / n + (n - \lceil \alpha n \rceil) M/n$.

For the cases where $n$ is bounded, we can use the same examples. The assumption that $n > 1/(1-\alpha)$ implies that $n > \lceil \alpha n \rceil$, so there will be at least one bucket with size $M$. Thus we have that in the worst-case, $dFill(I) \geq M$ for these instances, and the expected value of the solution generated by $rFill$ is at least $M/n$. Since $OPT_\alpha(I) = \lceil \alpha n \rceil < n$ and $n$ is fixed, the claim follows. □

Thus both $dFill$ and $rFill$ can be unboundedly bad if $M$ can be arbitrarily large, even if $n$ is bounded, but have bounded worst-case behavior if $M$ is bounded, even if $n$ is unbounded.

### B.3.2 The *Lowest Level* Algorithm

A complementary algorithm is "Lowest Level" ($LL$), which has bounded worst-case behavior if *either* $M$ or $n$ is bounded. In this algorithm, as long as we haven't yet filled our quota of buckets, we place the next ball in an unfilled bucket that currently contains the fewest balls. Here let $dLL$ denote any version of the algorithm where ties are broken in some deterministic fashion, and $rLL$ denote the version where ties are broken randomly, with all choices equally likely. We then have

**Theorem B.3** *For all $\alpha$, $0 < \alpha < 1$,*

1. *For all $M \geq 2$, $R^{dLL}_{\alpha,M,\infty} = R^{rLL}_{\alpha,M,\infty} = \dfrac{1-\alpha}{\alpha}(M-1)+1$.*
2. *For all $n > 1/(1-\alpha)$, $R^{dLL}_{\alpha,\infty,n} = R^{rLL}_{\alpha,\infty,n} = n - \lceil \alpha n \rceil + 1$.*

(Note that for $\alpha = 1/2$, these claims reduce to $R^{dLL}_{\alpha,M,\infty} = R^{dLL}_{\alpha,M,\infty} = M$, the same as for $rFill$ and $dFill$, and $R^{dLL}_{\alpha,\infty,n} = R^{rLL}_{\alpha,\infty,n} = n/2+1$ (for even $n$) and $n/2 + 1/2$ (for odd $n$).

**Proof.** We first show the upper bounds for (1) and (2). Let $I$ be an instance with $n > 1/(1-\alpha)$ buckets, and let $s_k$ be the size of the largest bucket filled in an optimal solution. Note that $s_k/s_1 \leq M$. We will have

$$OPT_\alpha(I) \geq (\lceil \alpha n \rceil) s_1 + s_k - s_1 \geq \max(\alpha n, s_k).$$

As to the performance of our algorithms, and suppose there are $y$ buckets of size $s_k$ in an optimal placement. In this case, the algorithms will be finished as soon as they put the $s_k$'th ball into the $y$'th size-$s_k$ bucket, leaving $n - \lceil \alpha n \rceil$ buckets with just $M-1$ balls. Thus

$$dLL_\alpha(I), rLL_\alpha(I) \quad \leq \quad OPT_\alpha(I) + (n - \lceil \alpha n \rceil)(s_k - 1)$$

and, hence, for $LL \in \{dLL, rLL\}$, we have

$$\begin{aligned}\frac{LL_\alpha(I)}{OPT_\alpha(I)} \quad &\leq \quad 1 + \min\left(\frac{1-\alpha}{\alpha}(s_k-1), \left(n - \lceil \alpha n \rceil\right)\frac{s_k-1}{s_k}\right) \\ &\leq \quad 1 + \min\left(\frac{1-\alpha}{\alpha}(M-1), \left(n - \lceil \alpha n \rceil\right)\frac{M-1}{M}\right).\end{aligned}$$

The overall limiting upper bounds follow.

For the lower bounds, consider instances $I_{M,n}$ in which there are $\lceil \alpha n \rceil - 1$ buckets of size 1 and all the remaining buckets have size $M$. Here

$$\begin{aligned} OPT_\alpha(I_{M,n}) &= \lceil \alpha n \rceil + M - 1 \le \alpha n + M, \\ dLL_\alpha(I) = rLL_\alpha(I) &= OPT_\alpha(I_{M,n}) + (n - \lceil \alpha n \rceil)(M-1). \end{aligned}$$

Fixing $M$ and letting $n \to \infty$ yields the matching lower bound for (1), while fixing $n$ and letting $M \to \infty$ yields the matching lower bound for (2). □

Note that, although $LL$ dominates *Fill* in terms of worst-case behavior when $M$ is unbounded, the situation is different when $M$ is fixed. Now *Fill* actually can have better worst-case behavior in many situations. So long as $M > 1$ and $\alpha \le 1/2$, we have

$$R^{dFill}_{\alpha,M,\infty} = M \le ((1-\alpha)/\alpha)(M-1) + 1 = R^{dLL}_{\alpha,M,\infty}$$

with a strict inequality if $\alpha < 1/2$. The advantage of $R^{rFill}_{\alpha,M,\infty}$ over $R^{rLL}_{\alpha,M,\infty}$ in this situation is even more substantial. Both these algorithms, however, are beaten for fixed $M$ by a Size-Only algorithm we shall be describing shortly.

In the meantime, we observe that the worst-case examples that we provided for both *Fill* and $LL$ only had two distinct bucket sizes. Interestingly, there is a No-Information algorithm that does much better than either on instances of this type.

### B.3.3 A Hybrid No-Information Algorithm for the 2-Size Case and Matching Lower Bound

The algorithm, which we shall call "Hybrid1" ($H1$), works on the *assumption* that there are just two sizes, and so can be about as bad as *Fill* when there are three sizes, in particular, having $R^{dH1}_{\alpha,\infty,n} = R^{rH1}_{\alpha,\infty,n} = \infty$ for any deterministic and randomized implementations $dH1$ and $rH1$ of $H1$. However such implementations do surprisingly well when there are just two sizes. The algorithm proceeds as in $LL$ until we fill a bucket. Let $d$ be the number of balls in this bucket. We add balls to all the remaining buckets to bring them up to level $d$. Then, while we have not yet filled more than $\alpha n$ buckets, repeatedly pick an unfull bucket and add balls to it until it is full. Note that if there are only two item sizes, all implementations of $H1$, randomized or deterministic, will require the same number of balls.

**Theorem B.4** *Suppose* $0 < \alpha < 1$. *Let* $H$ *represent any deterministic or randomized implementation of* $H1$. *Then for any instance* $I$ *with at most two distinct bucket capacities, we are guaranteed to have*

$$H_\alpha(I) \le (1/\alpha) OPT_\alpha(I).$$

**Proof.** First note that if $k = 1$, then the claim holds trivially, since $H_\alpha(I) = OPT_\alpha(I)$. Thus we may assume that $k = 2$. Suppose the two sizes are $s_1 < s_2$, and there are $k_1$ buckets of size $s_1$ and $k_2 = n - k_1$ buckets of size $s_2$. There are two cases.

(a) If $k_1 > \alpha n$, then $OPT_\alpha(I) = \lceil \alpha n \rceil s_1 > \alpha n s_1$ and $H_\alpha(I) \le n s_1 < (1/\alpha) OPT_\alpha(I)$, as desired.

(b) If $k_1 \le \alpha n$, then $OPT_\alpha(I) = k_1 s_1 + (\lceil \alpha n \rceil - k_1) s_2 = \lceil \alpha n \rceil s_1 + (\lceil \alpha n \rceil - k_1)(s_2 - s_1)$ and $H_\alpha(I) = n s_1 + (\lceil \alpha n \rceil - k_1)(s_2 - s_1) < (1/\alpha) OPT_\alpha(I)$, again as desired. □

Matching this guarantee, we have the following lower bounds.

**Theorem B.5** *Suppose* $0 < \alpha < 1$.

*(A) For any deterministic algorithm dA, even one that knows the profile, there exist instances* $I_n$ *for each* $n > 1/(1-\alpha)$*, with maximum size bounded independently of n, such that*

$$\lim_{n\to\infty} \frac{dA_\alpha(I_n)}{OPT_\alpha(I_n)} = \frac{1}{\alpha}.$$

*(B) For any* $\alpha$, $0 < \alpha < 1$ *and any* $\delta$, $\alpha < \delta < 1$, *there is a constant* $a < 1$ *and a sequence of instances* $I_n$*, such that the following holds for all sufficiently large n: Assume we first use a random permutation to relabel the buckets. Then any algorithm A that knows nothing more than the profile of the instance, and has no more than* $\delta(1/\alpha)OPT_\alpha(I_n)$ *balls, has probability less than* $a^n$ *of filling* $\lceil \alpha n \rceil$ *buckets.*

**Proof.** For (A), let $I_n$ be an instance with $n$ buckets, $\lceil \alpha n \rceil$ of them with size 1, the remainder with size $\lceil 1/\alpha \rceil$. Note that this implies that $OPT_\alpha(I_n) = \lceil \alpha n \rceil$. Our adversary arranges things so that the first $n - \lceil \alpha n \rceil$ buckets into which the algorithm places balls all have size $\lceil 1/\alpha \rceil$. If $dA$ completes the filling $\lceil \alpha n \rceil$ buckets before it has placed a ball in more than $n - \lceil \alpha n \rceil$ buckets, then we will have $dA_\alpha(I_n) \geq \lceil \alpha n \rceil \cdot \lceil 1/\alpha \rceil$ and hence $dA_\alpha(I_n)/OPT_\alpha(I_n) \geq 1/\alpha$, as desired. If, on the other hand, $dA$ places balls in $n - \lceil \alpha n \rceil$ or more buckets, then every one of the required $\lceil \alpha n \rceil$ filled buckets will entail the spending at least one ball in excess of the first balls into the $n - \lceil \alpha n \rceil$ size-$\lceil 1/\alpha \rceil$ buckets, for a total of at least $n$ balls. This implies $dA_\alpha(I_n)/OPT_\alpha(I_n) \geq n/\lceil \alpha n \rceil$, with the desired limiting ratio.

For (B), our desired instances $I_n$ are constructed by increasing the size of the size-$\lceil 1/\alpha \rceil$ buckets in the examples used for (B) to $n$. Note that the number of balls we are given is at most $p = (\delta/\alpha)\lceil \alpha n \rceil$. Although this is greater than $\lceil \alpha n \rceil$, since by assumption $\delta > \alpha$, it is less than $\gamma n$ for $\gamma = (\delta+1)/2 < 1$, so long as $n > \frac{2\delta}{(1-\delta)\alpha}$. In what follows, we assume that $n$ is at least this large. Thus, our algorithm cannot afford to fill any of the size-$n$ buckets and so must identify all $\lceil \alpha n \rceil$ size-1 buckets, which are in random locations.

First, suppose it makes its choices non-adaptively. In other words, it picks $p$ buckets ahead of time, and places one ball in each, succeeding if it manages to have chosen all $\lceil \alpha n \rceil$ size-1 buckets. The probability of success will be

$$\frac{\binom{p}{\lceil \alpha n \rceil}}{\binom{n}{\lceil \alpha n \rceil}} = \frac{p(p-1)\cdots(p-\lceil \alpha n \rceil+1)}{n(n-1)\cdots(n-\lceil \alpha n \rceil+1)}$$

But now, since $(p-i)/(n-i) \leq p/n \leq \gamma$ for $1 \leq i \leq p$, and $\lceil \alpha n \rceil < p$, this is at most $\gamma^{\lceil \alpha n \rceil} \leq \gamma^{\alpha n}$. Thus the probability of success declines as $a^n$ for $a = \gamma^\alpha < 1$, as desired.

So (B) holds if our algorithm is non-adaptive. But note that adaptivity yields no advantage for this particular task. At each step, all the as-yet-untouched buckets all have the same probability of being size-1, given our initial random permutation of the buckets. □

### B.3.4 A Hybrid Max-Only Algorithm for the Bounded-$M$ Case and Matching Lower Bounds

We can do better than either Fill or SLL in the case when $M$ is bounded. This involves a second hybrid algorithm, which we shall call Hybrid2 (H2). The algorithm works in the Max-Only situation and proceeds as follows. Suppose we are given an upper bound $M$ on the maximum bucket size.

1. Let $M' = \lfloor\sqrt{M}\rfloor$.
2. Apply $LL$ until either $\lceil\alpha n\rceil$ buckets are full or all non-full buckets contain $M'$ balls.
3. While less than $\lceil\alpha n\rceil$ buckets are full, pick an unfull bucket and add balls until it is filled to capacity.

Recall that in the Max-Only situation, we may assume that $M$ *is* the maximum bucket size; the following result holds even if it is only an upper bound.

**Theorem B.6** *For $0 < \alpha < 1$, any deterministic implementation dH2 of H2, and any instance $I$ with maximum bucket size no greater than $M$, we have*

$$\frac{dH2_\alpha(I)}{OPT_\alpha(I)} \le 1 + \frac{\sqrt{M}}{\alpha},$$

*and consequently $R^{dH2}_{\alpha,M,\infty}$ obeys the same bound.*

**Proof.** Let $I$ be any instance, and suppose $I$ contains $n$ buckets. Let $M''$ be the largest capacity of a bucket filled in an optimal solution, and let $x$ be the number of buckets of this size in the optimal solution. Then we have

$$OPT_\alpha(I) \ge \lceil\alpha n\rceil + x(M''-1) \ge \alpha n + x(M''-1)$$

There are two cases to consider. First, suppose that $M'' \le M'$. Then dH2 simply constructs an LL packing and we have

$$dH2_\alpha(I) \le OPT_\alpha(I) + (n - \lceil\alpha n\rceil)M'' \le OPT_\alpha(I) + n(1-\alpha)M''.$$

Consequently, since by assumption $M'' \le M' \le \sqrt{M}$, we have

$$\frac{dH2_\alpha(I)}{OPT_\alpha(I)} \le 1 + \frac{1-\alpha}{\alpha}M'' < 1 + \frac{\sqrt{M}}{\alpha},$$

as required.

Suppose, on the other hand, that $M'' > M'$. Then, by the definition of $M'$, we must have $M'' > \sqrt{M}$. In this case, we will have

$$dH2_\alpha(I) \le OPT_\alpha(I) + x(M - M'') + n(1-\alpha)M'.$$

Consequently, given that $M'' > \sqrt{M} \ge M'$, we have

$$\frac{dH2_\alpha(I)}{OPT_\alpha(I)} < 1 + \frac{M - M''}{M''-1} + \frac{1-\alpha}{\alpha}M' < 1 + \frac{M}{M''} + \frac{1-\alpha}{\alpha}M' < 1 + \frac{\sqrt{M}}{\alpha},$$

again as required. □

We do not have precisely matching lower bound examples for Theorem B.6. However, the next two theorems, to be proved below, show that no algorithm, deterministic or randomized, can do qualitatively better, even if it knows the full profile.

**Theorem B.7** *Suppose $0 < \alpha \le 1$. If $dA$ is any deterministic algorithm that knows nothing more than the profile of the instance, then for all $M \ge \max\{8, 1/\alpha^2\}$,*

$$R^{dA}_{\alpha,M,\infty} \ge \frac{\min(.79, 1-\alpha)}{1+\alpha}\sqrt{M}$$

*and*

$$\lim_{M\to\infty} \frac{R^{dA}_{\alpha,M,\infty}}{\sqrt{M}} \ge \frac{1-\alpha}{1+\alpha}.$$

**Theorem B.8** *Suppose we are given $\alpha$, $0 < \alpha < 1$, and $\epsilon > 0$, and let $\delta = (1-\alpha)/\big(1.25(1+\alpha)\max(6, \log(1/\epsilon))\big)$. For any $M \ge 8$, and $n > \max(2/(1-\alpha), 1/\alpha^2)$ that is divisible by $\lceil\sqrt{M}\rceil$, there is an instance $I_n$ with $n$ buckets and maximum bucket size $M$ such that, assuming we first use a random permutation to relabel the buckets, any algorithm that knows nothing more than the profile of the instance, and has fewer than $(\delta\sqrt{M})OPT_\alpha(I_n)$ balls, has probability less than $\epsilon$ of filling $\lceil\alpha n\rceil$ buckets.*

Note that Theorem B.8 does not provide as strong a lower bound as does Theorem B.5(C), since $\delta$ is not independent of $\epsilon$, and increasing $n$ does not decrease the success probability. We do not currently know whether there exist examples where this does happen – both $n$ and $M$ cancel out in the analysis of our current proof. The theorem is, however, strong enough to imply this easy corollary obtained by setting $\epsilon = 1/64$.

**Corollary 3** *Suppose we are given $\alpha$, $0 < \alpha < 1$. Then, for any randomized algorithm $rA$ that knows nothing more than the profile of the instance and any $M \ge 8$,*

$$R^{rA}_{\alpha,M,\infty} \ge \frac{1-\alpha}{7.5(1+\alpha)}\sqrt{M}.$$

**Proof of Theorem B.7.** The proof is based on examples $I_n$ with $n$ buckets for $n$ an integral multiple of $K = \lceil\sqrt{M}\rceil$. This will suffice since $R^{dA}_{\alpha,M,\infty}$ is defined as a lim sup. Let us first consider the bound that is claimed for $M \ge 8$, in which case we have $\sqrt{M}/\lceil\sqrt{M}\rceil > 0.79$.

Our instances have three sizes: 1, $K+1$ and $M$. Noting that, since $M \ge 1/\alpha^2$, we have $n/K \le \lceil\alpha n\rceil$, we have $\lceil\alpha n\rceil - n/K \ge 0$ buckets with size 1, $n/K$ buckets with size $K+1$, and $n - \lceil\alpha n\rceil$ buckets with size $M$. An optimal placement fills all the buckets of the first two types, yielding

$$OPT_\alpha(I_n) \;=\; \lceil\alpha n\rceil + \frac{n}{K}(K) \le n(\alpha+1)+1.$$

Our deterministic algorithm $dA$ will confront an adversary that arranges things so that no bucket of size $K+1$ receives its second ball until all buckets of size $M$ have received $K+1$ balls – recall that we may assume that if our algorithm puts a second ball in a bucket, it will keep placing balls there until it has reached $K+1$, the next valid size. There are two cases to consider.

(a) Suppose our algorithm ends up filling at least one bucket of size $K+1$. Then, by our assumption about the adversary, we must place at least $K+1$ balls in every bucket of size $M$. Thus, since $M - (\lceil\sqrt{M}\rceil + 1) \ge (\lceil\sqrt{M}\rceil + 1)$ whenever $M \ge 8$, we must in addition have enough balls to fill up all the buckets with size 1 or $K+1$, for a total of at least

$$\begin{aligned}\lceil \alpha n\rceil + n + (n - \lceil \alpha n\rceil)(K+1) &= 2n + (n - \lceil \alpha n\rceil)\lceil \sqrt{M}\rceil \\ &\geq 2n + (n(1-\alpha) - 1)\sqrt{M} \\ &\geq n(1-\alpha)\sqrt{M},\end{aligned}$$

the last inequality holding as soon as $n \geq \lceil \sqrt{M}\rceil/2$.

(b) Suppose our algorithm fills *none* of the size-$K+1$ buckets. Then it must fill at least $n/K$ buckets of capacity $M$ entirely, for a total number of balls at least

$$\frac{n}{\lceil \sqrt{M}\rceil} M \geq (.79)n\sqrt{M}.$$

Combining the conclusions of cases (a) and (b), we must use $\min(.79, (1-\alpha))n\sqrt{M}$ balls, and hence

$$\limsup_{n\to\infty} \frac{dA_\alpha(I_n)}{OPT_\alpha(I_n)} \geq \frac{\min(.79,(1-\alpha))n\sqrt{M}}{n(\alpha+1)+1} = \frac{\min(.79,(1-\alpha))\sqrt{M}}{\alpha+1},$$

as claimed for the case of $M \geq 8$.

For the second claimed bound, we note that it is already satisfied in case (a), while in case (b), it follows since $\lim_{M\to\infty}(\sqrt{M}/\lceil\sqrt{M}\rceil) = \sqrt{M}$. $\square$

**Proof of Theorem B.8.** We will restrict attention to the instances $I_n$ introduced in the previous proof. Recall that, for these instances, we have $OPT_\alpha(I_n) \leq n(\alpha+1)+1$. Thus, by hypothesis, our algorithm will be allowed

$$\left(\delta\sqrt{M}\right) OPT_\alpha(I_n) \leq \delta(n(\alpha+1)+1)\sqrt{M}$$

balls for instance $I_n$. One implication of this is that our algorithm cannot fill more than $\delta(n(\alpha+1)+1)/\sqrt{M}$ size-$M$ buckets in such an instance. To understand the significance of this number, recall that our hypothesis that $M \geq 8$ implies $\sqrt{M}/\lceil\sqrt{M}\rceil > 0.79$, and that we are also assuming that $n > 1/\alpha^2$. Thus the maximum number of size-$M$ buckets we can fill is

$$\begin{aligned}\delta\frac{(\alpha+1)n+1}{\sqrt{M}} &= \left(\frac{1-\alpha}{1.25(1+\alpha)\max(6,\log(1/\epsilon)}\right)\left(\frac{(\alpha+1)n+1}{\sqrt{M}}\right) \\ &\leq \left(\frac{(1-\alpha)}{7.5(1+\alpha)}\right)\left(\frac{(\alpha+1)n+1}{\sqrt{M}}\right) \\ &\leq \frac{(\alpha+1)n+1-\alpha-\alpha(\alpha+1)/\alpha^2}{7.2(1+\alpha)\sqrt{M}} = \frac{(\alpha+1)n-\alpha-1/\alpha^2}{7.5(1+\alpha)\sqrt{M}} \\ &\leq \frac{(\alpha+1)n}{7.5(1+\alpha)\sqrt{M}} = \frac{n}{7.2\sqrt{M}} < \frac{n}{7.5(.79)\lceil\sqrt{M}\rceil} < \frac{n}{5\lceil\sqrt{M}\rceil}.\end{aligned}$$

Thus, in order to fill our desired $\lceil\alpha n\rceil$ buckets, we will have to fill more than $4/5$ of the $n/\lceil\sqrt{M}\rceil$ buckets of size $\lceil\sqrt{M}\rceil+1$. In order to tell whether a bucket with size exceeding 1 has this size or size $M$, we must place $\lceil\sqrt{M}\rceil+1$ balls in the bucket. Therefore, the number of such buckets we can test is at most

$$\frac{\delta((\alpha+1)n+1)\sqrt{M}}{\lceil\sqrt{M}\rceil+1} \quad < \quad \delta((\alpha+1)n+1).$$

Given our initial random permutation of the buckets, every bucket tested must be chosen randomly from the as yet untested buckets with size exceeding 1. Thus the probability of us finding at least 13/14 of the size-$(\lceil\sqrt{M}\rceil+1)$ buckets, i.e, $\left\lceil 13n/14\lceil\sqrt{M}\rceil\right\rceil$ such buckets, is no more than that of finding at least that many black balls when picking $\lfloor\delta((\alpha+1)n+1)\rfloor$ balls, without replacement, from an urn containing $n/\lceil\sqrt{M}\rceil+n-\lceil\alpha n\rceil$ balls, only $n/\lceil\sqrt{M}\rceil$ of which are black. Hoeffding [26] has shown that this probability is no more than that implied by Chernoff bounds for the sum of $\lfloor\delta((\alpha+1)n+1)\rfloor$ independent random 0-1 variables, which are 1 with probability $(n/\lceil\sqrt{M}\rceil)/(n/\lceil\sqrt{M}\rceil+n-\lceil\alpha n\rceil)$.

The particular form of the Chernoff bound we use is from [34, p.64, Theorem 4.4(3)]:

**Chernoff Bound Lemma.** *Suppose $(X_1, X_2, \ldots, X_q)$ is a sequence of independent Poisson trials with probability of success p. Let $X=\sum_{i=1}^{q} X_i$ and $\mu = E[X]$. Then, for all $R \geq 6\mu$,*

$$Pr(X \geq R) \leq 2^{-R}.$$

Now, in our case, we have

$$\begin{aligned}
\mu = qp* &= \lfloor\delta((\alpha+1)n+1)\rfloor\left(\frac{\frac{n}{\lceil\sqrt{M}\rceil}}{\frac{n}{\lceil\sqrt{M}\rceil}+n-\lceil\alpha n\rceil}\right) < \big(\delta((\alpha+1)n\big)\left(\frac{\frac{n}{\lceil\sqrt{M}\rceil}}{n(1-\alpha)}\right)\\
&= \frac{1-\alpha}{1.25(1+\alpha)\max(6,\log(1/\epsilon)}\Big((\alpha+1)n\Big)\left(\frac{\frac{n}{\lceil\sqrt{M}\rceil}}{n(1-\alpha)}\right)\\
&\leq \frac{n}{1.25\max(6,\log(1/\epsilon)\lceil\sqrt{M}\rceil}.
\end{aligned}$$

Thus the ratio of the number of size-$(\lceil\sqrt{M}\rceil+1)$ buckets needed to the expected number found is at least

$$R = \left(\frac{4n}{5\lceil\sqrt{M}\rceil}\right)\Big/\left(\frac{n}{1.25\max(6,\log(1/\epsilon)\lceil\sqrt{M}\rceil}\right) = \max(6,\log(1/\epsilon).$$

Thus the Lemma applies for this $R$, and we can conclude that we fill the required number of buckets with probability less than $\min(1/64,\epsilon) \leq \epsilon$, as desired. □

(Note that there is some numeric slack in the above proof, introduced to simplify the arguments. We did not precisely optimize the value of $\delta$, nor did we take into account the fact that if some size-$M$ buckets were filled, we would have fewer balls left for filling size-$(\lceil\sqrt{M}\rceil+1)$ buckets. Consequently, the bounds of Theorem B.8 and its corollary could be improved slightly at the cost of a more complicated proof.)

### B.3.5 Algorithms Exploiting the Sizes-Only Case

Knowing the possible bucket sizes $s_1, s_2, \ldots, s_m$ occurring in our instance would seem to offer significant advantages. First, note that we may now assume that our algorithm operates in stages, as follows. Let $s_0 = 0$. At the beginning of each stage, every bucket contains $s_i$ balls for some $i$, $0 \le i \le m$. This trivially holds at the beginning of the first stage, when all buckets contain $s_0 = 0$ balls. In each stage, we pick an unfull bucket and add $s_{i+1} - s_i$ balls to it, where $s_i$ is the number of balls the bucket contained at the beginning of the stage.

It is easy to see that any algorithm that does not operate in stages can be converted to one that does and never uses more balls. Suppose not. Then the algorithm must contain ball placements that have no possibility of providing us with new information. For example, suppose that buckets are of size 2 and 5. We learn no new information by placing a single ball in an empty bucket, or fewer than three additional balls in a bucket that contains 2 balls and is not full. Thus, given an algorithm that makes such "informationless" placements, we can obtain a staged algorithm that does no worse by omitting such placements if the algorithm never places enough balls into the given bucket to reach the next allowed capacity, and otherwise postponing the placements until just before the placement of the ball that filled the bucket to that next capacity.

Let "Staged Lowest Level" (SLL) be the algorithm that, when we have not yet filled our quota of buckets, picks an unfilled bucket containing the fewest balls and adds to it enough balls to fill it up to the next possible bucket size. By the argument of the previous paragraph, SLL will perform no worse than LL, and it can clearly outperform LL in many situations. For example, in contrast to the results for LL, SLL can match the behavior of H1 on instances with just two capacities.

**Theorem B.9** *Suppose* $0 < \alpha < 1$. *For any variant dSLL of SLL based on a deterministic tie-breaking rule, and for any instance I with at most two distinct bucket capacities, we are guaranteed to have*

$$dSLL_\alpha(I) \le (1/\alpha) OPT_\alpha(I).$$

(This result is proved in exactly the same way as Theorem B.4.)

Unfortunately, as soon as there are three or more sizes, knowing them no longer buys us much under SLL, as we can essentially recreate the same lower bounds as for LL in Theorem B.3 by using sizes 1, $M-1$, and $M$ in the lower bound examples, as the reader can easily verify.

SLL can outperform LL with respect to a different metric, however – the asymptotic ratio $R^A_{\alpha,M,\infty}$ holding when the number $n$ of buckets is fixed but the maximum buckets size $M$ is not. For this metric, the best result we have seen so far is that of Theorem B.3 for the Lowest Level algorithm, which says that $R^{LL}_{\alpha,\infty,n} = n - \lceil \alpha n \rceil + 1$.

Consider the randomized version rSLL of SLL that at each step chooses a bucket randomly (and uniformly) from among the least filled unfull buckets, and adds balls to it to bring it up to the next possible size.

**Theorem B.10** *For* $0 < \alpha < 1$ *and* $n > 1/(1-\alpha)$, $R^{rSLL}_{\alpha,\infty,n} \le (n - \lceil \alpha n \rceil)/2 + 1$.

Moreover, this bound is tight and no algorithm can do better, even if it knows the full profile.

**Theorem B.11** *Let A be any algorithm that knows the full profile, but only begins packing after the buckets have been randomly permuted. Then, for* $0 < \alpha < 1$ *and* $n > 1/(1-\alpha)$, $R^A_{\alpha,\infty,n} \ge (n - \lceil \alpha n \rceil)/2 + 1$.

**Proof of Theorem B.10.** Suppose we are given an instance $I$ with $n > 1/(1-\alpha)$ and hence $n - \lceil \alpha n \rceil > 0$. The optimal solution for $I$ consists of the $\lceil \alpha n \rceil$ smallest buckets, i.e., those with sizes $s_1 \le s_2 \le \cdots \le s_{\lceil \alpha n \rceil}$. Thus we know that $OPT_\alpha(I) \ge s_{\lceil \alpha n \rceil}$. Let $\delta = OPT_\alpha(I)/\lceil \alpha n \rceil - 1$, so that $OPT_\alpha(I) = (1+\delta) s_{\lceil \alpha n \rceil}$.

If $\delta \geq 1$ we are done. For note that $rSLL_\alpha(I) \leq OPT_\alpha(I) + (n - \lceil \alpha n \rceil)s_{\lceil \alpha n \rceil}$, and so

$$\frac{rSLL_\alpha(I)}{OPT_\alpha(I)} \leq 1 + \frac{(n - \lceil \alpha n \rceil)s_{\lceil \alpha n \rceil}}{2s_{\lceil \alpha n \rceil}} = \frac{n - \lceil \alpha n \rceil}{2} + 1,$$

as claimed.

So assume $\delta < 1$. Then we must have $s_{\lceil \alpha n \rceil - 1} \leq \delta s_{\lceil \alpha n \rceil} < s_{\lceil \alpha n \rceil}$, and so at some point the algorithm will have reached a state in which each bucket contains a number of balls equal to the minimum of its size and $s_{\lceil \alpha n \rceil - 1}$, for a total of $\lceil \alpha n \rceil - 1$ full buckets and $OPT_\alpha(I) - s_{\lceil \alpha n \rceil} + (n - \lceil \alpha n \rceil + 1)s_{\lceil \alpha n \rceil - 1}$ balls. Algorithm rSLL will then randomly choose buckets from among the $n - \lceil \alpha n \rceil + 1$ currently unfull buckets containing $s_{\lceil \alpha n \rceil - 1}$ balls, adding $s_{\lceil \alpha n \rceil} - s_{\lceil \alpha n \rceil - 1}$ balls to each, until it encounters one for which the result is a full bucket. At this point it will have filled $\lceil \alpha n \rceil$ buckets and will halt. Elementary calculations show that the expected number of buckets that will have received these additional balls when that full bucket is obtained is $(n - \lceil \alpha n \rceil + 2)/2$. Thus the expected total number of balls used by rSLL is

$$\begin{aligned} OPT_\alpha(I) &- s_{\lceil \alpha n \rceil} + (n - \lceil \alpha n \rceil + 1)s_{\lceil \alpha n \rceil - 1} + \frac{(n - \lceil \alpha n \rceil + 2)(s_{\lceil \alpha n \rceil} - s_{\lceil \alpha n \rceil - 1})}{2} \\ &= OPT_\alpha(I) + \frac{(n - \lceil \alpha n \rceil)s_{\lceil \alpha n \rceil - 1}}{2} + \frac{(n - \lceil \alpha n \rceil)(s_{\lceil \alpha n \rceil})}{2} \\ &\leq OPT_\alpha(I) + \frac{(n - \lceil \alpha n \rceil)(1+\delta)s_{\lceil \alpha n \rceil})}{2}. \end{aligned}$$

Then, since $OPT_\alpha(I) = (1+\delta)s_{\lceil \alpha n \rceil}$, we have

$$\frac{rSLL_\alpha(I)}{OPT_\alpha(I)} \leq 1 + \frac{(n - \lceil \alpha n \rceil)(1+\delta)s_{\lceil \alpha n \rceil}}{2(1+\delta)s_{\lceil \alpha n \rceil}} = \frac{n - \lceil \alpha n \rceil}{2} + 1,$$

as claimed. □

**Proof of Theorem B.11.** Our lower bound is based on $n$-bucket instances with $OPT_\alpha(I_n) = n - \lceil \alpha n \rceil + X$ for a constant $X$ to be determined later. The instance consists of $\lceil \alpha n \rceil - 1$ buckets of size 1, one bucket of size $X+1$, and $n - \lceil \alpha n \rceil$, much larger, buckets having size $OPT_\alpha(I_n)\big((n - \lceil \alpha n \rceil)/2 + 1\big)$. Filling any one of these latter buckets would use the number of balls claimed by our proposed lower bound, so we can assume that the algorithm $A$ does not fill any of them. Therefore, in order to fill $\lceil \alpha n \rceil$ buckets, it must fill the bucket of size $X+1$. The algorithm cannot, however, distinguish the size-$(X+1)$ bucket from a bucket with the larger size without putting $X+1$ balls in the bucket in question. Given that the buckets were initially randomly permuted, the expected number of buckets that must be so tested before the bucket of size $X+1$ is found is again $(n - \lceil \alpha n \rceil + 2)/2$, of which one is filled and the remaining $(n - \lceil \alpha n \rceil)/2$ are not. Thus the expected number of balls used is at least $OPT_\alpha(I_n) + (n - \lceil \alpha n \rceil)(X+1)/2$. We thus have that

$$\frac{rSLL_\alpha(I)}{OPT_\alpha(I)} \geq 1 + \frac{(n - \lceil \alpha n \rceil)(X+1)}{2(X + \lceil \alpha n \rceil)}$$

Taking the limit as $X \to \infty$ gives the desired result. □

Note that this lower bound result, although tight, is weaker than our earlier ones, since it only refers to expected values. The best probabilistic bound we can currently prove derives from Theorem 3.2, by fixing $n$ in that result rather than $B$. We then obtain (roughly, and for large $n$), that there is a constant $a < 1$ and instances $I'_n$ such that, assuming we randomly permute the buckets at the beginning, any algorithm that only knows the profile and uses fewer than $OPT_\alpha(I'_n)(n - \lceil \alpha n \rceil)/8$ balls has probably less than $\alpha^n$ of success. This can probably be improved a bit, but it is not clear that there is a similar result when the denominator 8 is replaced by something arbitrarily close to the 2 of our lower bound on expected behavior.

As a final remark about sizes-only algorithm, we note that, analogously to our modifying LL to the staged version SLL, we can also adapt the H2 algorithm of the previous section to a staged version SH2 that might perform better in practice. Here, instead of setting $M' = \lfloor \sqrt{M} \rfloor$, we can let it be the maximum bucket size $s_i$ that is less than $\sqrt{M}$. The proof of Theorem B.6 will still apply.

### B.3.6 Algorithms that Exploit the Known-Profile Case

In light of the lower bound Theorems B.1, B.5, B.8, and B.11, there would seem to be little opportunity to exploit the extra information available in a known profile to obtain a general qualitative improvement. We ourselves have not yet found one, and leave this as an open problem for the reader.

## B.4 A Bucket-Based Metric

In this section we consider a metric based on bucket counts rather than ball counts, and in particular how much hidden diversity reduces the number of buckets we can fill, given that we have enough balls to fill our target number of buckets. As before, let us fix an $\alpha$, $0 < \alpha < 1$. For an algorithm A, an instance $I$ with $n$ buckets, and a number $b$ of balls, let $A(I, b)$ be the number of buckets filled when A has placed $b$ balls – we assume that the algorithm assumes it has an unbounded supply of balls and does not know $b$ in advance. Let $OPT_\alpha(I)$ be the number of balls in an optimal placement for filling $\lceil \alpha n \rceil$ buckets. For a deterministic algorithm dA, define

$$X^{dA}_{\alpha,n} = \min\left\{ \frac{A(I, OPT_\alpha(I))}{\lceil \alpha n \rceil} : \ I \text{ is an instance with } n \text{ buckets} \right\}.$$

For a randomized algorithm rA, define

$$X^{rA}_{\alpha,n} = \min\left\{ \frac{E\left[A(I, OPT_\alpha(I))\right]}{\lceil \alpha n \rceil} : \ I \text{ is an instance with } n \text{ buckets} \right\}.$$

In both cases, define

$$X^{A}_{\alpha,\infty} = \liminf_{n \to \infty} X^{A}_{\alpha,n}.$$

Note that we are not parameterizing by the maximum bucket capacity $M$ here. A maximum bucket size of $M$ automatically guarantees that $X^{dFill}_{\alpha,\infty} = 1/M$, but getting exact results for sophisticated algorithm when $M$ is bounded is likely to be complicated. Consequently, we shall leave such questions for later. The results for unbounded $M$ are interesting in their own right.

With respect to this metric, best algorithm we currently know is a randomized one which we shall called "$d$-Sampled Lowest Level" algorithm (*$d$-Sample*), where $0 < d < 1$).

1. Randomly pick $\lfloor d \lceil \alpha n \rceil \rfloor$ buckets.

2. Perform LL on just these buckets until there are no more balls left.

**Theorem B.12** *For all $\alpha$, $0 < \alpha < 1$, and $d$, $0 < d \le (1-\sqrt{1-\alpha})/\alpha$,*

$$d\left(\frac{1-d}{1-d\alpha}\right)\alpha \le X^{d-Sample}_{\alpha,\infty} \le d\alpha,$$

*where the lower bound is always at least $d^2\alpha$ and is maximized when $d$ equals its upper bound, which itself exceeds $1/2$.*

(Note that, as $\alpha \to 1$, the ratio between the upper bound and the best lower bound goes to 1, but, as $\alpha \to 0$, the ratio goes to $1-d$. Thus there is clearly still work to do in analyzing the algorithm. For $\alpha = 1/2$, the upper bound on $d$ is approximately 0.58579 and, with $d$ set to this value, we have $0.17157 < X^{d-Sample}_{\alpha,\infty} < 0.2929$.)

**Proof.** First, let us deal with the numerical claims. The upper bound will be greater than $1/2$ so long as $\sqrt{1-\alpha} < 1-\alpha/2$, which is equivalent to $1-\alpha < 1-\alpha+\alpha^2/4$, which is true since $\alpha > 0$. For future reference, also note that our upper bound $(1-\sqrt{1-\alpha})/\alpha$ is always less than 1, since $\alpha < 1$, which implies $1-\alpha < \sqrt{1-\alpha}$.

To see that $(1-d)/(1-d\alpha) \ge d$, we must show that $1-d \ge d(1-d\alpha)$ or $d^2\alpha - 2d + 1 \ge 0$. For the latter, we have equality when $d = \big(1 \pm \sqrt{1-\alpha}\big)/\alpha$, with the negative option the only one that yields a $d$ in the allowed range. Note that because both $d$ and $\alpha$ are less than 1, the function $d^2\alpha - 2d + 1$ has a negative derivative and so is a decreasing function of $d$. Thus the claimed bound $(1-d)/(1-d\alpha) \ge d$ holds for all allowed values of $d$.

To see that the bound $d\left((1-d)/(1-d\alpha)\right)\alpha$ is maximized when $d$ attains its maximum allowed value of $(1-\sqrt{1-\alpha})/\alpha$, we take a derivative with respect to $d$, obtaining

$$\frac{(1-2d)(1-d\alpha)-(d-d^2)(-\alpha)}{(1-d\alpha)^2} = \frac{d^2\alpha-2d+1}{(1-d\alpha)^2}.$$

Our original function will be either minimized or maximized when the numerator equals 0, which we already know happens when $d = 1-\sqrt{1-\alpha})/\alpha$. Since the derivative is positive for $d = 1/2$ and the smallest zero for the derivative is the one given above, this means that the function is maximized at this value.

Now let us turn to the asymptotic bounds for the algorithm, starting with the upper bound, which is straightforward. It is based on the following instances $I_n$, $n > 1/(d\alpha)$. In $I_n$, we have $\lceil \alpha n \rceil$ buckets of size 1, and the remaining buckets all have size $K = \lceil \alpha n \rceil + 1$. Thus we have $OPT_\alpha(I_n) = \lceil \alpha n \rceil$ and our algorithm has this many balls to play with. The expected number of size-1 buckets in our sample is $\lfloor d\lceil \alpha n\rceil \rfloor (\lceil \alpha n \rceil/n)$ which approaches the desired upper bound $(d\alpha)OPT_\alpha(I_n)$ on filled buckets as $n \to \infty$. And note that we do not have enough balls to fill any other buckets.

For the lower bound, let us warm up by fixing $d = 1/2$ and proving a weaker but easier bound $d\alpha/2$. Let $B_1, B_2, \ldots, B_n$ be an indexing of the buckets by increasing size, ties broken arbitrarily. Recall that this means that $OPT_\alpha(I) = \sum_{i=1}^{\lceil \alpha n \rceil} s_i$. Let $t = \lceil \lceil \alpha n \rceil/2 \rceil$, and call all the buckets with indices $i \le t$ "good". Note that we will have filled all good buckets by the time $LL$ has brought every bucket in the sample up to the minimum of its size and $s_t$, which will require at most $s_t\lfloor d\lceil \alpha n \rceil \rfloor$ balls. But now note that

$$OPT_\alpha(I) \ge \sum_{i=1}^{t} s_i + s_t(\lceil \alpha n \rceil - t) \ge \sum_{i=1}^{t} s_i + s_t\lceil \alpha n \rceil/2 \ge s_t\left\lfloor d\lceil \alpha n \rceil \right\rfloor.$$

Consequently we have enough balls to fill all the good buckets in our sample, of which the expected number is $d\lceil \alpha n\rceil(t/n) = (d\lceil \alpha n\rceil\lfloor\lceil \alpha n\rceil/2\rfloor)/n$, which approaches $(d\alpha/2)\lceil \alpha n\rceil$ as $n \to \infty$, proving our lower bound claim.

There is considerable slack in the above argument, even when $d = 1/2$. This is because we can use the balls corresponding to the unused $\sum_{i=1}^{t} s_i$ component of the lower bound on $OPT$ to fill the good buckets, so that the balls corresponding to the $s_t(\lceil \alpha n\rceil - t)$ component only need to be used to fill non-good buckets. This is what we exploit in the general lower bound of the Theorem, which we now prove.

Let $d^* = (1-d)/(1-d\alpha)$. For all $\epsilon > 0$, we will show that $X_{\alpha,\infty}^{d-Sample} \geq (1-\epsilon)dd^*\alpha$, which will imply the lower bound. We may assume without loss of generality that $\epsilon < 1$. Suppose $I$ is an arbitrary instance with $n$ buckets. Let $t$ now equal $\lfloor (1-\epsilon)d^*\lceil \alpha n\rceil\rfloor$, and call a bucket $B_i$ "good" if $i \leq t$, and "bad" otherwise. Let $r = \lfloor d\lceil \alpha n\rceil\rfloor$, the number of buckets we sample. Then the expected number $g$ of good buckets in our sample of $r$ buckets is

$$E[g] = r\left(\frac{t}{n}\right) \sim d\alpha t \sim (1-\epsilon)dd^*\alpha\lceil \alpha n\rceil,$$

where we use "$\sim$" to mean essentially equal as $n \to \infty$. Note that this limiting value is just the number of buckets we need to fill for the claimed lower bound on $X_{\alpha,\infty}^{d-Sample}$ to hold.

To complete the proof, we need to show that, in asymptotic expectation, our algorithm will be guaranteed to fill all the good buckets in our sample as soon as it has placed a number of balls that is no more than $OPT_\alpha(I)$. Let $b$ denote the number of bad buckets in our sample. All of the good buckets in our sample are guaranteed to be filled as soon as LL has placed enough balls in the buckets of the sample for each to contain the minimum of its size and $s_t$. This number is clearly no more than $\sum_{i=1}^{t} s_i + bs_t$. On the other hand, the number of balls we will have available is at least $OPT_\alpha(I) \geq \sum_{i=1}^{t} s_i + s_t(\lceil \alpha n\rceil - t)$. Thus we will have enough balls so long as $b \leq \lceil \alpha n\rceil - t$.

Now note that

$$E[b] = d\lceil \alpha n\rceil - E[g] \sim d\lceil \alpha n\rceil - d\alpha t \sim d\alpha n\big(1-(1-\epsilon)d^*\alpha\big)$$

and thus

$$\begin{array}{rcl} (\lceil \alpha n\rceil - t) - E[b] & = & \lceil \alpha n\rceil - \lfloor (1-\epsilon)d^*\lceil \alpha n\rceil\rfloor - E[b] \\ & \sim & \alpha n\big(1-d-(1-\epsilon)d^*(1-d\alpha)\big) \\ & = & \alpha n\big(1-d-(1-\epsilon)(1-d)\big) = (1-d)\alpha n\epsilon. \end{array}$$

Thus, we will fill all the good buckets in our sample unless we have

$$b - E[b] > (1-d)\alpha n\epsilon \sim \frac{(1-d)\alpha n\epsilon E[b]}{d\alpha n\big(1-(1-\epsilon)d^*\alpha\big)} > \left(\frac{\epsilon(1-d)}{d(1-(1-\epsilon)d^*\alpha)}\right)E[b].$$

This is a constant fraction $\beta$ of $E[b]$, and so another Chernoff bound theorem ([34, p.64, Theorem 4.4(2)], applicable again via the arguments of [26]) implies that the probability that this holds is no more than $e^{-(\beta^2/3)E[b]} = a^n$ for some constant $a < 1$ since $E[b]$ is itself linear in $n$.

Let $f$ be the number of good buckets actually filled after we have placed $OPT_\alpha(I)$ balls (a lower bound on the total number of buckets filled). Then we have

$$E[f] \quad \geq \quad \sum_{i=1}^{d\lceil \alpha n\rceil} (i \cdot Pr[g = i] \cdot Pr[\text{all good buckets filled}])$$

Now, as observed above, all good buckets will be filled unless $b > E[b] + (1-d)\alpha n\epsilon$ or, equivalently, $g < E[g] - (1-d)\alpha n\epsilon$, and this happens with probability less than $a^n$. Thus, in the above expression for $E[f]$, the only summands which can take on values less than the full count $i$ are ones whose cumulative probability is less than $a^n$, which drops exponentially in $n$, whereas the maximum value for $i$ is no more than $d\lceil \alpha n \rceil$, which grows only linearly. Thus, as $n \to \infty$, the contribution of these terms to $E[f]$ goes to 0, and, in the limit, the expected number of buckets we fill is at least the expected number of good buckets in our sample, which we have already seen is precisely what is needed to prove our lower bound on $X_{\alpha,\infty}^{d-Sample}$.

(The above argument, rendered informal by the use of "$\sim$", can be made rigorous by interpreting $A \sim B$ to mean $B \in [(1-\delta)A, (1+\delta)A]$ where $\delta > 0$ is an arbitrarily small constant, and the inclusion holds for all $n > n_\delta$, where $n_\delta$ is a constant depending only on $\delta$. All the statements involving "$\sim$" can then be restated in terms of such $\delta$, with our claims holding by taking yet another limit, this one for $\delta \to 0$.) $\square$

In contrast, we have the following general upper bound theorems.

**Theorem B.13** *For all $\alpha$, $0 < \alpha < 1$, if dA is any deterministic algorithm, even one that knows the profile, we have*

$$X_{\alpha,\infty}^{dA} = 0.$$

**Theorem B.14** *For any randomized algorithm rA,*

$$X_{\alpha,\infty}^{rA} \le \alpha.$$

**Theorem B.15** *Suppose we are given $\alpha$, $0 < \alpha < 1$, and $\delta > 0$. Then there is a constant $a < 1$ and a sequence of instances $I_n$ such that the following holds for sufficiently large n: Assuming we first use a random permutation to relabel the buckets, any algorithm that knows nothing more than the profile of the instance, and has no more than $OPT_\alpha(I)$ balls, has probability less than $a^n$ of filling $(1+\delta)\alpha\lceil \alpha n \rceil$ buckets.*

**Proof of Theorem B.13.** Assume $n$ is sufficiently large that $\Delta = n - \lceil \alpha n \rceil > 0$. Given dA, we will show how to construct an instance $I_n$ with $n$ buckets for which $dA(I_n, OPT_\alpha(I_n)) = 0$. The theorem will follow. We start with $n$ buckets of undetermined size, and apply dA for $n\lceil \lceil \alpha n \rceil / \Delta \rceil$ ball placements, each time informing the algorithm that the placement did not fill the chosen bucket. Now declare each bucket to have size one more than the number of balls it currently contains. Divide the buckets into two sets: $C$, the $\lceil \alpha n \rceil$ buckets containing the fewest balls (ties broken arbitrarily), and $D$, the remaining $\Delta$ buckets. Let $T$ denote the number of balls in the buckets of set $C$. Then $OPT_\alpha(I_n) = T + \lceil \alpha n \rceil$. Now note that the $\Delta$ buckets in $D$ contain at least $\Delta\lceil \lceil \alpha n \rceil / \Delta \rceil \ge \lceil \alpha n \rceil$ balls. Thus, when we stopped dA, it had already placed $T$ balls in the buckets of $C$ and at least $\lceil \alpha n \rceil$ balls in the buckets of $D$, for a total of at least $OPT_\alpha(I_n)$ balls, without filling any bucket. So $dA(I_n, OPT_\alpha(I_n)) = 0$, as required. $\square$

**Proof of Theorem B.14.** We exhibit a sequence of instances $I_n$ that imply the upper bound on $X_{\alpha,\infty}^{rA} \le \alpha$ for any randomized algorithm rA. These instances are quite simple: $I_n$ contains $\lceil \alpha n \rceil$ buckets of size 1, and the remaining buckets all have capacity $n+1$. This means that $OPT_\alpha(I_n) = \lceil \alpha n \rceil$, and so, when restricted to $OPT_\alpha(I_n)$ balls, algorithm rA can only place balls in this many buckets. It can only *fill* a bucket when it chooses one with size 1, since all the others have size exceeding $OPT_\alpha(I_n)$. However, since it has no information about the identity of the buckets, the expected number of buckets of size 1 that it will choose is only $\big(\lceil \alpha n\rceil\big)\big(\lceil \alpha n\rceil/n\big)$. This means that the ratio of this number to the target number is $\lceil \alpha n\rceil/n$, yielding the desired limit as $n \to \infty$. $\square$

**Proof of Theorem B.15.** We use the same examples as in the proof of Theorem B.14. The algorithm can only afford to fill buckets of size 1, and has $\lceil \alpha n\rceil$ balls with which to do so. Given the random relabeling of the buckets, its success is thus simply the probability that, in making $\lceil \alpha n\rceil$ random bucket choices, it will find at least $(1+\delta)\alpha\lceil \alpha n\rceil$ size-1 buckets, where the expected number of size-1 buckets it chooses goes to $\alpha\lceil \alpha n\rceil$ as $n \to \infty$. The claim follows by an application of the Chernoff bound used in the proof of Theorem B.12. $\square$