Electronic Colloquium on Computational Complexity, Revision 2 of Report No. 176 (2019)

On Prover-Efficient Public-Coin Emulation of Interactive Proofs

Gal Arnon* Guy N. Rothblum†

April 27, 2021

Abstract

A central question in the study of interactive proofs is the relationship between private-coin proofs, where the verifier is allowed to hide its randomness from the prover, and public-coin proofs, where the verifier's random coins are sent to the prover. The seminal work of Goldwasser and Sipser [STOC 1986] showed how to transform private-coin proofs into public-coin ones. However, their transformation incurs a super-polynomial blowup in the running time of the honest prover.

In this work, we study transformations from private-coin proofs to public-coin proofs that preserve (up to polynomial factors) the running time of the prover. We re-consider this question in light of the emergence of doubly-efficient interactive proofs, where the honest prover is required to run in polynomial time and the verifier should run in near-linear time. Can every private-coin doubly-efficient interactive proof be transformed into a public-coin doubly-efficient proof? Adapting a result of Vadhan [STOC 2000], we show that, assuming one-way functions exist, there is no general-purpose black-box private-coin to public-coin transformation for doubly-efficient interactive proofs.

Our main result is a loose converse: if (auxiliary-input infinitely-often) one-way functions do *not* exist, then there exists a general-purpose efficiency-preserving transformation. To prove this result, we show a general condition that suffices for transforming a doubly-efficient private coin protocol: every such protocol induces an efficiently computable function, such that if this function is efficiently invertible (in the sense of one-way functions), then the proof can be efficiently transformed.

This result motivates a study of other general conditions that allow for efficiency-preserving private to public coin transformations. We identify two such additional (incomparable) conditions. The first allows for transforming any *constant-round* interactive proof (even if it is not doubly-efficient). The second allows for transforming any private coin interactive proof where (roughly) it is possible to approximate the number of verifier coins consistent with a partial transcript. We demonstrate the applicability of this final result by using it to transform a private-coin protocol of Rothblum, Vadhan and Wigderson [STOC 2013], obtaining a doubly-efficient public-coin protocol for verifying that a given graph is close to bipartite in a setting for which such a protocol was not previously known.

*galarnon42@gmail.com. Weizmann Institute of Science. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819702)

†rothblum@alum.mit.edu. Weizmann Institute of Science. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819702)

ISSN 1433-8092

Contents

1 Introduction

Interactive proofs (IPs), introduced by Goldwasser, Micali and Rackoff [GMR85] in 1985 are an important object in the study of complexity and cryptography. An interactive proof is an interactive protocol between two parties, a "prover" and a "verifier", where the prover is trying to convince the verifier of the membership of a string in a language. If this claim is true, then the verifier should be convinced with high probability. Otherwise, if the claim is false, then no matter what the prover does, the verifier should reject the claim with high probability. Since their inception, a central question in the study of interactive proofs has been the connection between private-coin proofs, where the verifier is allowed to hide its randomness from the prover, and public-coin proofs, where hiding information is not allowed. Public-coin protocols are especially appealing since they are easier to analyze and manipulate [FGM+89, BM89, BGG+88, FS86]. Goldwasser and Sipser [GS86] showed that any private-coin interactive proof can be transformed into a public-coin proof while preserving the number of rounds (up to an additive constant).

One issue with this transformation is that of the honest prover's running time. Vadhan [Vad00] showed that (assuming the existence of one-way functions) there exist protocols that cannot be transformed to be public-coin in a black-box manner while preserving the running time of the prover (up to polynomial factors). While in the classical setting the running time of the prover is considered unbounded, the recent line of works on *doubly-efficient* interactive proofs (deIPs) [GKR08] restricts the honest prover to run in polynomial time. We emphasize that soundness is required to hold against computationally unbounded adversaries. Doubly-efficient interactive proofs apply only to tractable computations, and are therefore of interest when the verifier time can be smaller than the time required to decide the language without the help of a prover. Indeed, the main focus in the literature is on verifiers that run in near-linear time. Goldreich [Gol18] gives a survey on recent work on doubly-efficient interactive proofs.

This Work. In this work we ask whether transformations of proofs from using private coins to using public coins are applicable to the doubly-efficient setting:

Which private-coin doubly-efficient interactive proofs can be transformed into public-coin doubly-efficient proofs and how can this be done?

We tackle the above question from a number of angles. Some of our results also apply to proofs that are not doubly-efficient.

We extend Vadhan's impossibility result to show that the existence of one-way functions implies that there are no transformations from private-coin deIPs to public-coin deIPs that work in a natural "black-box" way (See Appendix A for more information). Note that since deIPs exist only for problems in $\mathcal{BPP}$, one can *always* transform such proofs to using "public-coins" by having the verifier solve the problem on its own. This transformation is not black-box, but it is also not interesting, as the motivation for deIPs is to reduce the verifier's running time to under what is required for it to solve the problem on its own.

Our main result shows that this reliance on one-way functions is essentially tight. Namely, if one-way functions (of a certain type) do not exist, then (essentially) every doubly-efficient proof can be efficiently transformed:

Theorem 1 (Informal Statement of Theorem 5)**.** *Suppose that infinitely-often auxiliary-input one-way functions do not exist. Then every language that has a doubly-efficient private-coin interactive proof with "good enough" soundness has a doubly-efficient public-coin interactive proof with the same number of rounds (up to a constant).*

Remark 1.1. Theorem 1 mentions "good enough" soundness. This is due to the fact that there is a strong degradation in soundness when applying our technique.[1] One could be tempted to amplify the soundness using parallel or sequential repetition, but in the setting of deIPs, the overhead of repeating the protocol in

[1] Specifically, in order for the public-coin protocol that we end up with to have constant soundness error, the soundness error of the original (private-coin) protocol should be $O(\text{poly}(n, r, \ell)^{-r})$ where n is the input length, and r and ℓ are the number of rounds and number of random bits used by the verifier in the original private-coin protocol respectively.

terms of the verifier might be problematic (e.g. it might degrade the verifier's running-time from linear to quadratic).

Viewing this result through the prism of Impagliazzo's worlds [Imp95], it (very) roughly says that in Pessiland (a world where one-way functions do not exist), efficiency-preserving transformation is always possible. We prove Theorem 1 by showing that for every deIP there exists a specific efficiently computable function such that if it is efficiently invertible in the sense of one-way functions[2], then efficiency-preserving transformation is possible (see Section 1.1 for a discussion on the notion of invertibility). We remark that a straightforward implementation of the Goldwasser-Sipser transformation requires exponential running time from the prover, and even an oracle that inverts *any* given function (on random inputs) does not seem sufficient for making their public-coin prover efficient. Indeed, our results require changing the transformation so that the ability to invert becomes sufficient for constructing a public-coin prover.

Using the technique for proving Theorem 1 (and some additional technical work) we show that in Pessiland's "one-way function"-less landscape, standard constant-round proofs (i.e. ones where the honest prover is allowed to run in super-polynomial time) can also be transformed to be public-coin with only a polynomial overhead on the honest prover's running time:

Theorem 2 (Informal Statement of Theorem 6)**.** *Suppose that infinitely-often auxiliary-input one-way functions do not exist. Then every language that has a constant-round private-coin interactive proof has a constant-round public-coin interactive proof where the honest prover's running time is polynomially related to that of the private-coin prover.*

Sufficient conditions for efficient transformation. Both the impossibility result of [Vad00] and our extension to doubly-efficient proofs are proved by demonstrating a specific (arguably contrived) protocol that is hard to transform. It is very natural, then, to ask: considering interactive proofs on a case-by-case basis, for which protocols (or families of protocols) *is* efficiency-preserving transformation possible? In other words we wish to identify sufficient conditions that allow for efficiency-preserving transformation of private-coin proofs to public-coin ones.

In particular, Theorem 1 implies one such condition: Every deIP has an efficiently computable function such that if this function is efficiently invertible in the sense of one-way functions, then efficient transformation for this proof system is possible.

We identify an additional, rather natural, sufficient condition for efficient transformation. We show that if it is possible to efficiently count the number of coins that are consistent with transcripts of the protocol, then it is also possible to efficiently emulate the protocol using public-coins. Unlike in Theorems 1 and 2, this result does not preserve the number of rounds, but it applies to general interactive proofs (even when the protocol has an inefficient honest prover and a polynomial number of rounds).

Theorem 3 (Informal Statement of Theorem 7)**.** *Let $\mathcal{L}$ be a language and suppose that $\mathcal{L}$ has an r-round private-coin interactive proof with communication complexity m, and suppose that for every incomplete transcript it is possible to efficiently approximate the number of verifier random coins that are consistent with the transcript. Then $\mathcal{L}$ has a $2rm$-round public-coin interactive proof with an efficient prover.*

A string of random coins ρ is consistent with an incomplete transcript of an execution of a protocol if for every verifier message α in the transcript, the verifier outputs α when given the transcript prefix leading up to α and using ρ as its random coins. We prove this theorem using a "piecemeal" emulation protocol in which the prover and the verifier together generate a string that is distributed according to the distribution of a random transcript in the private-coin protocol. The soundness error of the resulting protocol is a function of how good the approximation algorithm is. In particular, if one can *exactly* count the number of verifier random coins that are consistent with a transcript efficiently, then soundness is perfectly preserved.

Theorem 3 gives us a condition for efficiently transforming proofs from private-coin to public-coin that is incomparable to the condition implied by Theorems 1 and 2. The condition implied by from Theorems 1 and 2 is efficient distributional inversion for some (efficiently computable) function that depends on the

[2]The requirement that a specific function is *distributionally* invertible [IL89] also suffices.

protocol, whereas Theorem 3 uses efficient approximation of the number of verifier coins consistent with a transcript. We demonstrate a natural protocol for which the efficient counting condition of Theorem 3 is satisfied, whereas we don't know how to efficiently invert the function implied by Theorems 1 and 2.

An application. Rothblum, Vandhan and Wigderson [RVW13] show a private-coin proof of proximity for distinguishing between a graph that is bipartite and graphs that are both far from bipartite and well-mixing. Roughly speaking, an interactive proof of proximity (IPP) is an interactive proof where the verifier has sub-linear query access to the input. The problem of distinguishing between a bipartite graph and a well-mixing graph that is far from bipartite has also been studied extensively in the past in the context of property testing [GR99, GR02]. By applying Theorem 3 to the private-coin protocol of [RVW13] we show a new doubly-efficient public-coin proof system for this problem. We describe this below in more detail.

Theorem 4 (Informal Statement of Theorem 9)**.** *For every $\varepsilon > 0$, there exists a* public-coin *interactive proof of ε-proximity with an* efficient *prover for the problem of distinguishing between bipartite graphs and graphs that are ε-far from bipartite and are well-mixing.*

We remark that no such proof was previously known (except for ε close to 1).

The main property of the RVW bipartiteness protocol that we use, is the fact that it that the (private coin) verifier uses only a logarithmic number of coins (it has logarithmic *randomness complexity*), and this suffices for soundness error $1 - \Omega(\varepsilon)$. Thus, polynomial time suffices for enumerating all possible choices of verifier randomness and for exactly counting how many choices are consistent. The transformation of Theorem 3 is soundness preserving when efficient exact counting is possible, and so gives an efficient protocol with soundness error $1 - \Omega(\varepsilon)$, which can be amplified to obtain constant soundness. See Section 6 for further discussion and details.

We note that a similar argument applies to every proof system where the verifier's randomness complexity is $O(\log n)$. We show that *every* such private-coin IP (resp. IPP) can be transformed to a public-coin IP (resp. IPP) while the honest prover's running time remains polynomial.

We further note that the Goldwasser-Sipser (GS) approach to transforming private-coin proofs into public-coin ones, and the result behind Theorem 1, can also preserve the prover's efficiency when the verifier's randomness complexity is $O(\log n)$. However, the GS approach degrades soundness significantly, and hence usually requires parallel repetition before applying the transformation. Here, since the starting (private coin) protocol has large soundness error, repeating it to reduce the soundness error to the point where the GS approach can be applied requires super-logarithmic randomness complexity, which means that the GS transformation will not preserve the prover's efficiency. In comparison, in this setting Theorem 3 preserves soundness (see above), and so it can be used even when the soundness error of the private-coin protocol is large.

Related Work. A number of works have tackled the question of private versus public coins, including Haitner, Mahmoody and Xiao [HMX10] who showed that if the prover is given an $\mathcal{NP}$ oracle it is possible to transform private-coin protocols into public-coin ones where both the prover and the verifier run in polynomial time. Holenstein and Künzler [HK13] show a public-coin protocol in which the prover helps the verifier sample from a distribution, where in addition to the sampled element the verifier ends up with an approximation of the probability that the element is sampled from the distribution. They then show this can be used for public-coin emulation. Goldreich and Leshkowitz [GL16] improved upon the soundness requirement of Goldwasser and Sipser. In all of the results described the prover is inefficient and the running time of the verifier incurs a polynomial overhead. We additionally note that the celebrated $\mathcal{IP} = \mathcal{PSPACE}$ Theorem [LFKN92, Sha92], implies a non-black-box transformation of private-coin protocols to public-coin ones. The protocol used to show that $\mathcal{PSPACE} \subseteq \mathcal{IP}$ is public-coin, and so one can use this result to transform private-coin protocols into public-coin ones as follows: Given an interactive proof, use the transformation for $\mathcal{IP} \subseteq \mathcal{PSPACE}$ to convert it into a $\mathcal{PSPACE}$ problem. Then use the reduction and protocol showing that $\mathcal{PSPACE} \subseteq \mathcal{IP}$ to construct a public-coin proof. While this transformation is not black-box, it blows up the complexity of the honest prover and the number of rounds of the protocol.

Organization. This paper is organized into 6 sections. Sections 1 and 2 introduce the background and main ideas of the paper and present relevant preliminaries as well as some useful lemmas. Section 3 lays out the protocols and algorithms we use in our prover-efficient round-preserving emulation from distributional inversion. In Section 4 we take the tools developed in the previous section and show how they apply to public-coin emulation of interactive proofs. In Section 5 we use a "piecemeal" emulation protocol to show that prover-efficient public-coin emulation of interactive proofs can also be done given appropriate approximate counting. In Section 6 we show, as an application of our piecemeal emulation protocol, a public-coin interactive proof of proximity for distinguishing between a bipartite graph and a well-mixing expander that is far from bipartite. In Appendix A we prove that if one-way functions exist there are no black-box prover-efficient transformations for doubly-efficient interactive proofs.

1.1 Overview of the Round-Efficient Emulation

Goldwasser and Sipser [GS86] showed not only that it is possible to transform private-coin proofs into public-coin ones, but also that this can be done without significantly increasing the number of rounds in the protocol. We show that, given a distributional inverter for certain functions, the prover can be made efficient. A distributional inverter for a function f is an efficient randomized algorithm that upon receiving an input y drawn from the distribution $f(U_n)$ returns a random element from the set $f^{-1}(y)$.

Remark 1.2. Proving Theorems 1 and 2 would be significantly simpler if we were to consider a stronger form of inversion, where the input y to the inverter can be any element in the support of f. That is, y need not be given as a sample from the distribution $f(U_n)$. We consider the weaker and more complicated variant, since it allows us to establish Theorem 1, and through it the tight relationship between one-way functions and the (non-)existence of private-coin to public-coin transformations that preserve the prover's running time.

We give three toy cases for protocols of increasing complexity, and then discuss the general case. For each case we show how Goldwasser and Sipser's original protocol can be applied in order to transform it to public-coin, and then discuss how we use distributional inversion in order to make the prover efficient. Eventually, this requires changes to the Goldwasser-Sipser transformation. In all three toy cases consider a language $\mathcal{L}$ and a one round private-coin protocol denoted $\langle P, V\rangle$ with perfect completeness and soundness error s. Since it has one round, the protocol is of the following form: On input x the verifier begins with choosing a random $\rho \leftarrow \{0,1\}^\ell$, then sends $\alpha = V(x,\rho)$ where $\alpha \in \{0,1\}^m$. After receiving β from the prover, the verifier accepts if $V(x,\alpha,\beta;\rho) = 1$. Henceforth throughout this overview we omit the shared input x from notation of the verifier and prover functions.

1.1.1 Case 1: Equally Likely Messages With *Known* Number of Messages

The Protocol: In addition to the protocol being one-round and having perfect completeness, we assume the following properties:

- Equally Likely Messages: If x is in the language $\mathcal{L}$, then every pair of messages $\alpha_1, \alpha_2 \in \{0,1\}^m$ that have non-zero probability of being sent by the original verifier V are equally likely: $\Pr_{\rho\leftarrow U_\ell}[V(\rho) = \alpha_1] = \Pr_{\rho\leftarrow U_\ell}[V(\rho) = \alpha_2]$.
- Known Number of Messages For Completeness: If $x \in \mathcal{L}$ then there is a known efficiently computable function $N : \{0,1\}^* \to \mathbb{N}$ such that the total number of messages sent by the verifier with non-zero probability is $N(x)$ (i.e. $N(x) = \left|\left\{\alpha | \exists \rho \in \{0,1\}^\ell \text{ s.t. } V(x;\rho) = \alpha\right\}\right|$).
- Few Messages For Soundness: If $x \notin \mathcal{L}$ then there are significantly fewer than $N(x)$ verifier messages.

As a running example for this case one is encouraged to think of the classical private-coin protocol for the graph non-isomorphism problem [GMW86]. In this language the input is comprised of two n-vertex graphs G_0 and G_1 which are claimed to be non-isomorphic. The protocol is as follows: the verifier chooses a random bit b, and random permutation π. It sends $\tilde{G} = \pi(G_b)$ to the prover who must return some b'. The verifier accepts if $b' = b$. One can easily verify that this protocol has completeness 1 and soundness

error $\frac{1}{2}$. Moreover, assuming for simplicity that the graphs have no automorphisms, every verifier message is equally likely and if the graphs are non-isomorphic the number of possible verifier messages is $N = 2n!$. If the graphs are isomorphic then there are only $n!$ different messages.

The Goldwasser-Sipser Transformation: The transformation of $\langle P, V \rangle$ into a public-coin protocol hinges on the observation that, in this toy case, in order distinguish whether x is in the language, the prover need only show that the number of possible verifier messages is at least N (since by assumption if $x \notin \mathcal{L}$ there are significantly fewer such messages). Thus a (public-coin) "set lower-bound" protocol is used, showing that the set of all valid verifier messages (ones that are sent by V with non-zero probability over the choice of its random coins) is large . Letting $\mathcal{H}_{m,k}$ be a family of pairwise independent hash functions from $\{0,1\}^m$ to $\{0,1\}^k$, U_k be the uniform distribution over k bits and $k = k(N)$ be a value to be discussed later, the final protocol is as follows:

1. The parties execute a "set lower-bound" protocol proving that the number of verifier messages is at least N:
   (a) The verifier chooses a random $h \leftarrow \mathcal{H}_{m,k}$ and $y \leftarrow U_k$ [3]
   (b) The prover returns $\alpha \in \{0,1\}^m$.
   (c) The verifier tests that $h(\alpha) = y$ and otherwise rejects.
2. The prover sends $\rho \in \{0,1\}^\ell$.[4]
3. The verifier accepts if $V(\rho) = \alpha$.

In the case of completeness, where there are N verifier messages, if k is "small enough" (i.e. the hash function is very compressing) it is likely that there exists some valid message α that hashes to y. Conversely, in the case of soundness, where there are significantly fewer than N legal verifier messages, if k is "large enough" then it is unlikely that there will exist a valid message that hashes to y. Thus, completeness and soundness of the protocol are governed by setting k to a reasonable value, which depends on the gap between N and and the magnitude of the prover's lie in the case of a false claim. Note that in this protocol the prover did not even need to send its message β - it was sufficient to use the fact that there is a large gap in the number of verifier messages between the cases of completeness and soundness. The sub-protocol executed in Step 1 is known as the "set lower-bound" protocol, and can be generalized to show a lower-bound on the size of any set S for which the verifier can efficiently test membership. Specifically, the protocol begins with a claim that $N \leq |S|$ and ends with both parties holding an element x for which the verifier needs to verify that it belongs to S. If $N \leq |S|$, then $x \in S$ with high probability, and if $|S| \ll N$, $x \notin S$ with high probability regardless of the prover strategy. A protocol inspired by the set lower-bound protocol is presented and analysed in Section 3.2 under the name "prover-efficient sampling protocol". Going back to the example of graph non-isomorphism, upon receiving h, y from the verifier, the prover would send some graph $\tilde{G}$, a bit b and a permutation π. The verifier would then accept if $h(\tilde{G}) = y$ and $\tilde{G} = \pi(G_b)$.

Prover Efficiency: The prover strategy in the above protocol is inefficient. It receives some h, y and is required to find a legal message α that hashes to y and some choice of randomness ρ that leads to α. However, the prover can be made efficient by giving it oracle access to an inverter for the function $f(h, \rho) = h, h(V(\rho))$. An inverter for a function $f : \{0,1\}^a \to \{0,1\}^b$ is a randomized algorithm that on input y drawn from the distribution $f(U_a)$ returns some element x in the set of preimages of y under f (that is, $x \in f^{-1}(y)$). We stress that the input to the inverter must come from the correct distribution, which in our case is $f(\mathcal{H}_{m,k}, U_\ell) \equiv (\mathcal{H}_{m,k}, \mathcal{H}_{m,k}(V(U_\ell))$. The hash function h is clearly chosen by the verifier from the correct distribution. The image y is drawn by the verifier from the uniform distribution which, if the

[3] In the classic transformation it suffices to set $y = 0^k$ and is described here thus as it will be required later by our transformation.

[4] The final prover message can be merged with the previous one to save a round and is described here as a separate round for clarity.

hash function is compressing enough, will be statistically close to $h(V(U_\ell))$ by the Leftover Hash Lemma. Preimages of (h, y) with respect to f are of the form (h, ρ) where $h(V(\rho)) = y$. The prover can send ρ and use ρ to calculate α. In all of this the prover only needs to make a single oracle call, and to calculate $\alpha = V(\rho)$, and is therefore efficient.

1.1.2 Case 2: Equally Likely Messages With *Unknown* Number of Messages

The Protocol: We make the same assumptions on the protocol as in Case 1, except that the verifier in the transformation does not know N, the number of verifier messages.

The Goldwasser-Sipser Transformation: The protocol is based on two observations made in the case of a cheating prover:

- Since the soundness error is s and the verifier uses ℓ random coins, the number of verifier messages α for which there exist β and ρ such that $V(\alpha, \beta; \rho)$ accepts is at most $s \cdot 2^\ell$.
- For every fixed α and β, the number of coins ρ such that $V(\rho) = \alpha$ and $V(\alpha, \beta; \rho)$ accepts is also bounded from above by $s \cdot 2^\ell$.

If either of the above were not true, it would mean the soundness error is greater than s. Now notice that since all messages are equally likely, if there are N valid verifier messages, then each message has $\frac{2^\ell}{N}$ different coins that are consistent with it. Importantly, if N is small, $\frac{2^\ell}{N}$ is large. This gives rise to the following protocol:

1. Prover sends N, a claim on the number of verifier messages.
2. Prover and verifier execute the set lower-bound protocol to show that the number of legal verifier messages is at least N. The parties end up with some α claimed to be a verifier message.
3. Prover sends some β.
4. The parties execute the set lower-bound protocol to show that the number of coins that are consistent with α and lead the verifier to accept (α, β) is at least $\frac{2^\ell}{N}$. The parties end up with a ρ which is supposed to in the set of random coins that lead the verifier to α.
5. Verifier accepts if $V(\rho) = \alpha$ and $V(\alpha, \beta; \rho) = 1$.

The protocol is complete, since if the prover is honest it is likely to succeed in both the set lower-bound protocols, meaning it samples both a valid message α and valid coins ρ such that $V(\rho) = \alpha$ and $V(\alpha, \beta; \rho) = 1$. We now turn towards soundness. Let S be the set of verifier messages for which the prover has an accepting strategy (messages α for which there exist β and ρ such that $V(\alpha, \beta; \rho)$ accepts). For verifier message α and prover message β, let $T_{\alpha,\beta}$ be the number of random coins ρ such that $V(\alpha, \beta; \rho)$ accepts. Recall from the argument above, that $|S| \leq s \cdot 2^\ell$ and for any α, β, $|T_{\alpha,\beta}| \leq s \cdot 2^\ell$. Now note that if the verifier ends up with a verifier message $\alpha \notin S$ it has a message for which no fixing of β and ρ will make the verifier accept. Thus the prover must try to sample in S. Similarly, after fixing α and β if the verifier has $\rho \notin T_{\alpha,\beta}$ it will reject, and so the cheating prover must try to cause the verifier to end up with an element in $T_{\alpha,\beta}$. To see why the protocol is sound consider the prover's choice of N. If $|S| \leq s \cdot 2^\ell \ll N$, then due to the set lower-bound protocol the prover is unlikely to make the verifier sample from S and so the verifier will reject with high probability. If N is small, then $\frac{2^\ell}{N}$ is large. Fixing α and β, if $|T_{\alpha,\beta}| \leq s \cdot 2^\ell \ll \frac{2^\ell}{N}$, then the verifier is unlikely to end up with such coins, meaning it rejects. Note in the above analysis we have that $s \ll \min\{N, \frac{2^\ell}{N}\}$. Thus if s is small enough to begin with, the prover will be forced to lie and be caught with high probability.

Prover Efficiency: Inspecting the above protocol there are three things that the honest prover needs to do: Count N, the number of verifier messages, execute the prover's side of the set lower-bound protocol to show that there are at least N verifier messages, and execute the prover's side of the set lower-bound protocol to prover that the number of coins that are consistent with α is at least $\frac{2^\ell}{N}$. We explain how to compute each of these efficiently in reverse order:

1. Set Lower-Bound Protocol for Number of Consistent Coins: In this set lower-bound protocol the parties have already computed a verifier message α. The prover receives some hash function h and an image y from the verifier, and must return some ρ such that $V(\rho) = \alpha$ and $h(\rho) = y$. Naively it seems that this can be solved simply if the prover has access to an inverter for the function $f(h, \rho) = h, V(\rho), h(\rho)$ since preimages of (h, α, y) are exactly of the form h', ρ such that $h', V(\rho), h'(\rho) = h, \alpha, y$. The problem with this idea is that to use this inverter it must be that the message α *be drawn from the distribution* $V(U_\ell)$. Thus to use this idea it is imperative that α come from the correct distribution.

2. Set Lower-Bound Protocol for Number of Messages: In order to complete this stage, the prover must find some valid verifier message α that hashes to y, and this (as we did in Case(1))can be done using an inverter for the function $f(h, \rho) = h, h(V(\rho))$. Unfortunately as mentioned in point (1), we need it α to be chosen from the real message distribution $V(U_\ell)$. Unfortunately using an inverter as described, the message α might be drawn from a distribution which is far from the real one.[5] This issue is fixed if we move from using a regular inversion oracle to a *distributional* inversion oracle. Roughly, a distributional inverter for a function $f : \{0,1\}^m \to \{0,1\}^t$ is a randomized algorithm A such if y is drawn from $f(U_m)$, the distribution $A(y)$ is statistically close to a random pre-image of y under f.

3. Computing N: We show in Section 3.5 that given an inversion oracle for the function $f(h, \rho) = h, h(V(\rho))$ it is possible to efficiently approximate N, the number of verifier messages. The way this is done is inspired by the techniques of [Sto83] for approximate counting using an $\mathcal{NP}$ oracle. The prover chooses $h \leftarrow \mathcal{H}_{m,k}$ and $y \leftarrow \{0,1\}^k$ for increasingly larger values of k, and calls the inversion oracle on input (h, y). When k is small relative to the number of verifier messages, there will likely exist a message α that hashes to y, and thus the inverter will return a set of coins ρ such that $V(\rho) = \alpha$. Once k is set to a relatively large value this is unlikely and the inverter will fail. Thus, given the smallest size of k for which the inverter fails the prover can estimate the size of the set.

1.1.3 Case 3: A Two-Cluster Protocol

The Protocol: As in Case 2, we do not assume that the parties know initially the number of verifier messages. Moreover we replace the assumption that all messages are of equal likelihood with the following one:

- Two Clusters: The verifier messages that have non-zero probability can be partitioned into two "clusters" C_0 and C_1, where every message in C_b has equal likelihood p_b. Furthermore, each message in C_1 is significantly more likely than messages in C_0: $p_0 \ll p_1$. Both parties know the values p_0 and p_1 but not $|C_0|$ and $|C_1|$.

The Goldwasser-Sipser Transformation: Rather than giving a claim about the number of possible messages, the prover will claim only that the heaviest of the flat clusters is large (the weight of cluster C_b is $p_b \cdot |C_b|$).

1. Prover calculates $|C_0|$ and $|C_1|$ and chooses a bit b such that $p_b \cdot |C_b| \geq p_{1-b} \cdot |C_{1-b}|$. It sends b and $N = |C_b|$.

2. Verifier tests that $p_b \cdot N \geq \frac{1}{2}$ and otherwise rejects.

[5] Indeed, suppose that the inverter always returns the lexicographically smallest ρ such that $h(V(\rho)) = y$. This will cause a sampling bias towards those messages that have coins that are lexicographically smaller.

3. Prover and verifier execute the set lower-bound protocol to show that the size of cluster C_b is at least N. The parties end up with some α claimed to be in cluster b.

4. Prover sends some β.

5. The parties execute the set lower-bound protocol to show that the number of coins that are consistent with α and lead the verifier to accept (α, β) is at least $p_b \cdot 2^\ell$. The parties end up with a ρ which is supposed to be a set of random accepting coins that lead the verifier to α.

6. Verifier accepts $V(\rho) = \alpha$ and $V(\alpha, \beta; \rho) = 1$.

Completeness can be verified by noting that since there are only two clusters, the heaviest cluster must hold at least half of the weight of the distribution and so the verifier's test that $p_b \cdot N \geq \frac{1}{2}$ will pass. Due to the fact that every message α in C_b has likelihood p_b, there are exactly $p_b \cdot 2^\ell$ different coins that would lead the verifier to output α and to accept. For soundness first fix b. Once b is fixed, the smallest the prover can set N to be is $\frac{1}{2p_b}$ because otherwise the verifier will reject in Step 2. If p_b is very small, this value is large. As in the analysis of Case 2, since the total number of verifier messages for which the prover as an accepting strategy is small, if the claim N is large, it will likely not manage to make the verifier accept. If p_b is large, then the value of N can be set to be small, but in this case the value $p_b \cdot 2^\ell$ in Step 5 is large. Noting that for any α and β the number of coins ρ for which $V(\alpha, \beta; \rho)$ accepts is at most $s \cdot 2^\ell$, which we think of as very small, the prover is unlikely to be able to cause the verifier to end up with coins that will make it accept.

Prover Efficiency: In the classic transformation as described above the prover must do three things: Calculate the size of each cluster, and take part in both executions of the set lower-bound protocol.

1. Counting $|C_0|$ and $|C_1|$: The approximate counting technique as used in Case 2 to approximate N can be used to approximate the number of coins which would lead to a message. Notice that for $\alpha \in C_b$ there are exactly $p_b \cdot 2^\ell$ coins that lead to α. Thus, by randomly choosing many messages and counting how many are in each cluster, the prover can build a "histogram" of the weights of each cluster - a list of each cluster and its respective weight. This is formally addressed in Section 3.3 where it is shown that using both a sampling and a membership oracle one can build such a histogram. An issue with this approach is that the approximation procedure returns an *approximate* value for the number of coins that lead to a message. That is, for a message in C_b it may claim that the number of coins that lead to the message is anywhere in the range $(1 \pm \varepsilon) \cdot p_b \cdot 2^\ell$ for some (relatively small) ε. Since p_0 and p_1 are far from each other, the ranges $(1 \pm \varepsilon) \cdot p_0 \cdot 2^\ell$ and $(1 \pm \varepsilon) \cdot p_1 \cdot 2^\ell$ do not intersect. Thus it is still easy to recognize to which cluster a message belong.

2. Set Lower-Bound Protocol for Number of Messages in C_b: In this part of the protocol the prover receives a hash function h and image y and needs to return a message α such that $\alpha \in C_b$ and $h(\alpha) = y$. The prover cannot simply use an inverter for a function that samples inside of C_b since there may not be an efficient function for sampling in C_b. We instead show that given a distributional inverter for $f(h, \rho) = h, h(V(\rho))$ it is possible to find preimages that are in C_b. This is true because the cluster has significant weight with respect to the distribution of messages, and so for a randomly chosen hash function and random image y the weight of elements that belong to the cluster and are preimages to y is unlikely to be very small relative to all other preimages of y.

3. Set Lower-Bound Protocol for Number of Consistent Coins: The prover's strategy can be made efficient in this protocol in exactly the same way as in the same set lower-bound in the previous case: by inverting $f(h, \rho) = h, V(\rho), h(\rho)$. Doing this has the same issue dealt with in Case 2 - the distribution from which the message α is drawn must be close to uniform. In the classical transformation the value of α will be far from random because the protocol always works with the heaviest cluster. Suppose, for example, that $p_0 \cdot |C_0| = p_1 \cdot |C_1| + \varepsilon$ for a very small ε. Then C_0 will always be chosen, even though in the the real message distribution the likelihood of being in each cluster is almost identical. In order to fix this skew in distribution we make the choice of b "smoother". Rather than setting b as the index of the

heaviest cluster, we let b be random and sampled from the Bernoulli distribution where 0 is drawn with probability $\frac{p_0 \cdot |C_0|}{p_0 \cdot |C_0| + p_1 \cdot |C_1|}$. This presents a minor issue: now it could be that $p_b \cdot N < \frac{1}{2}$. To fix this, we limit the choice of clusters only to ones that have some noticeable probability of appearing, and so the verifier can make sure that the claimed probability is not smaller than this threshold. Similar smoothing techniques were used in [HK13] and [GL16] in different contexts.

1.1.4 Towards the General Case

In the general case, the verifier message distribution cannot be split into a small number of flat clusters and the protocol may have multiple rounds. To keep this overview simple we only consider doubly-efficient proofs. Making the transformation work for constant-round proofs with an inefficient prover requires some slight additional technical work.

General Message Distribution. General Message Distribution: The issue for working with a general distribution for the verifier messages is solved in the classical transformation by defining clusters of messages as follows: cluster i is the set of all the messages with weight in the range 2^{-i} and 2^{-i+1}. In our case, we have to work harder. Firstly, due to the way we use distributional inverters, we will need that for every cluster, the distribution of messages when restricted only to messages in the cluster be statistically close to uniform. This can be solved by splitting the distribution into more clusters - cluster i will now be all messages in the range $(1 + 1/\operatorname{poly}(n))^{-i}$ and $(1 + 1/\operatorname{poly}(n))^{-i+1}$. Note that the probabilities of messages in neighbouring clusters are similar. Therefore, the observation made in analysis of Case 3 that we can distinguish to which cluster a message belongs even though the approximation procedure does not return exact values for the number of coins that lead to a message, is false. To solve this issue, we work with "approximate clusters" - cluster i consists of all the verifier messages for which *the approximation procedure claims* the weight is between $(1 + 1/\operatorname{poly}(n))^{-i}$ and $(1 + 1/\operatorname{poly}(n))^{-i+1}$.

Imperfect Completeness. In order to accommodate protocols with completeness c, recall that the clusters are defined as sets of accepting coins with some weight. In the classical transformation in Case 3 the final prover's claim is that there are $p_b \cdot 2^\ell$ coins which would lead the verifier to accept. Since now p_b is the probability that these coins are sampled *conditioned* on sampling accepting coins, the end claim is changed to $p_c \cdot c \cdot 2^\ell$. In our case, we will not be able to use inverters to find accepting coins, since we only know how to invert efficient functions, and we do not have an efficient function that returns a random accepting coin. We therefore redefine the clusters to refer to gerenal coins, not just accepting ones. This means that in our protocol, the verifier accepts with almost the same probability as in the original private-coin protocol.

Multiple Rounds. The issue of multiple rounds is solved in the original transformation by iteratively emulating each round of the protocol. In the following we ignore the issue of distributions over messages which are not uniform. This is treated as explained under "General Message Distribution". In the Goldwasser-Sipser protocol round i starts with the prefix of a transcript $\gamma_{i-1} = (\alpha_1, \beta_1, \ldots, \alpha_{i-1}, \beta_{i-1})$ and N_{i-1}, a claimed lower bound on the number of coins that are consistent with γ_{i-1}. The prover gives a claim N_i that there are N_i coins consistent with each message possible verifier message conditioned on the transcript prefix γ_{i-1}. The parties next run the set lower-bound protocol. That is, the verifier sends a randomly sampled hash function h and random y. The honest prover sends back α_i such that $h(\alpha_i) = y$ and that α_i is consistent with γ_{i-1} (i.e. there exist ρ such that $\alpha_1 = V(\alpha_1, \beta_1, \ldots, \alpha_{i-1}, \beta_{i-1}; \rho)$). The prover then sends β_i. This process is run iteratively until the parties have a full transcript γ along with a claimed lower-bound on the number of coins consistent with this full transcript, at which point the parties execute a final set lower-bound protocol to sample a set of coins ρ.

To follow Goldwasser and Sipser's formula with an efficient prover, we would like an efficient method such that given a random hash function h, and y find this method outputs a consistent α_i. In the one-round case as explained previously, we noted that $f(h, \rho) = h, h(V(\rho))$ was an efficiently computable function in order to sample $\alpha_1 = V(\rho)$. How can we use the same idea but correlate the output to a transcript? To

more easily illustrate, in the following we consider a 2-round protocol so that our goal is to sample α_2 after the transcript (α_1, β_1) has already been set.

We show that if the proof in question is doubly-efficient, it suffices to invert the function f that on inputs h and ρ: Computes $\alpha_1 = V(\rho)$, $\beta_1 = P(\alpha_1)$, $\alpha_2 = V(\alpha_1, \beta_1; \rho)$ and outputs $(\alpha_1, \beta_1, h(\alpha_2))$. Firstly note that since the prover is efficient the function f can be computed in polynomial time. Next, notice that the distribution $f(\mathcal{H}, U_\ell)$ is identical to that of taking α_1, β_1 from a random execution of the protocol and additionally outputting a random hash of the next verifier message. Consider an inverter for f. Given a random pair α_1, β_1 and a random h, y it returns randomness ρ such that $\alpha_1 = V(\rho)$. Given ρ it is easy to compute $\alpha_2 = V(\alpha_1, \beta_1; \rho)$. Since ρ is consistent with α_1, β_1, we have that α_2 is also consistent with α_1, β_1. Moreover, α_2 will hash to y. This is exactly what we needed.

1.2 Overview of the Piecemeal Emulation Protocol

In this section we overview the techniques used in Section 5 to prove Theorem 7. We prove the theorem by constructing a protocol which we call the "piecemeal" emulation protocol, which is inspired by ideas described in [Sud07] that are accredited to Joe Kilian.

The protocol hinges on a sampling protocol where the goal of the honest prover is to help the verifier generate a transcript that is distributed similarly to a random transcript of the protocol the parties are trying to emulate. Let $\mathcal{L}$ be a language and $\langle P, V \rangle$ be an r-round interactive proof for $\mathcal{L}$ with ℓ bits of randomness and message length m. Since there are r rounds, there are $2r$ messages sent in the protocol. Each message is of length m, and so the length of a complete transcript of the protocol is $2rm$ bits. We assume without loss of generality that the protocol ends with the verifier sending its entire randomness to the prover. To reiterate, our goal is to generate a random transcript of an execution of the proof. We do this bit-by-bit in an iterative manner as follows: Round i begins with a partial transcript prefix γ_{i-1} and a claimed lower bound N_{i-1} on the number of random coins which are consistent with this partial transcript, where $\gamma_0 = \emptyset$ is the empty transcript with the claim that all $N_0 = 2^\ell$ coins are consistent with the empty transcript. By consistent with a partial transcript we mean that had the private-coin verifier received these coins at the beginning of the protocol execution, then this partial transcript would have been generated, with respect to the prover messages. The prover sends two values N_i^0 and N_i^1 where N_i^0 is the number of coins that are consistent with extending the transcript with the bit 0, meaning coins consistent with the transcript $(\gamma_{i-1}, 0)$, and similarly N_i^1 is the number of coins consistent with $(\gamma_{i-1}, 1)$. If the prover can exactly count each of these values, then it should be that $N_{i-1} = N_i^0 + N_i^1$. The verifier tests that indeed $N_{i-1} = N_i^0 + N_i^1$ and chooses a bit b with probability $\frac{N_i^b}{N_{i-1}}$. Both parties set the new transcript to be $\gamma_i = (\gamma_{i-1}, b)$ and the new claim on the number of consistent coins to be $N_i = N_i^b$. This continues on until $i = 2rm$ when a full transcript has been generated, where since we assumed that the verifier ends by outputting its randomness, there can only be one random coin that is consistent with the transcript. Therefore, after the last iteration the verifier tests that the final $N_{2rm} = 1$, and that all verifier messages in the transcript are what V would have sent in an actual execution using randomness from the end of the transcript. Finally, if all these tests pass the verifier and accepts if V accepts given the transcript γ_{2rm}.

For completeness, it can be shown that the protocol described above generates a transcript with the exact same distribution as the original one, since in every stage the next bit of the transcript is chosen with probability equal to the probability that it would appear in a real random transcript conditioned on the part of the transcript that has already been fixed. We now would like to to show that the protocol is sound, i.e. that for $x \notin \mathcal{L}$ a malicious prover cannot cause the verifier to accept in the new protocol with probability greater than in the original protocol. To show this we look the ratio between the number of claimed consistent coins, N and the number of consistent coins that would make the verifier accept in a given round. For a given partial transcript γ we denote by $Acc(\gamma)$ the set of coins ρ such that there exists a legal full transcript of the real execution γ' which begins with γ and in which the verifier accepts.

We begin our inspection of soundness with the final round and work backwards from there. Let $i = 2mr$, and let N_{i-1} and γ_{i-1} be the claim and the partial transcript at the beginning of the iteration. Since the transcript ends with the verifier sending its entire randomness, the number of accepting coins consistent with

a transcript with only one bit missing can be 0, 1 or 2. It can be shown that in every case, the probability that the verifier ends up accepting is at most $\frac{|Acc(\gamma_{i-1})|}{N_{i-1}}$. For conciseness we focus in this overview on what happens if $|Acc(\gamma_{i-1})| = 1$. In this case only one of the two options for the final bit will make the verifier accept. Suppose this bit is 0, then the probability that the verifier accepts reduces to the probability that it chooses $b = 0$, which is $\frac{N_i^0}{N_{i-1}}$. Now, since in the end of the protocol the verifier tests that $N_i = N_{2rm} = 1$ in order for the prover to cause the verifier to accept bit 0 it must set $N_i^0 = 1$. Therefore, the probability that the verifier ends up accepting the transcript is at most $\frac{1}{N_{i-1}} = \frac{|Acc(\gamma_{i-1})|}{N_{i-1}}$.

We now look at other rounds of the protocol. Let γ_{i-1} and N_{i-1} be the inputs to iteration i. Suppose, as our induction hypothesis, that upon entering round $i+1$ with γ_i and N_i the probability that the verifier ends up accepting is $\frac{|Acc(\gamma_i)|}{N_i}$. Let N_i^0 and N_i^1 be the values sent by the prover. By the induction hypothesis, if the verifier chooses bit b, which happens with probability $\frac{N_i^b}{N_{i-1}}$, then it will end up accepting with probability $\frac{|Acc(\gamma_{i-1},b)|}{N_i^b}$. Therefore the probability that the verifier ends up accepting is:

$$\frac{N_i^0}{N_{i-1}} \cdot \frac{|Acc(\gamma_{i-1},0)|}{N_i^0} + \frac{N_i^1}{N_{i-1}} \cdot \frac{|Acc(\gamma_{i-1},1)|}{N_i^1} = \frac{|Acc(\gamma_{i-1},0)| + |Acc(\gamma_{i-1},1)|}{N_{i-1}}$$

Noting that $|Acc(\gamma_{i-1},0)| + |Acc(\gamma_{i-1},1)| = |Acc(\gamma_{i-1})|$ we have that the verifier eventually accepts with probability $\frac{|Acc(\gamma_{i-1})|}{N_{i-1}}$. This inductive argument extends all the way up to γ_0 and N_0 in which case $\frac{|Acc(\gamma_0)|}{N_0}$ is equal to the soundness error of the original protocol.

The actual protocol differs slightly from the one described above. In the real setting, the honest prover cannot exactly calculate N_i^0 and N_i^1, but rather only ε-approximate them. This will mean that the transcript that is sampled is only close to uniform. A further implication of this change is that since the honest prover can err, the verifier now must relax its test that $N_i^0 + N_i^1 = N_{i-1}$. This relaxation turns out to be to test that $\frac{N_{i-1}}{N_i^0+N_i^1} \leq 1 + 3\varepsilon$. This in turn gives the cheating prover some additional leeway, specifically in round i the probability that the verifier ends up accepting changes from $\frac{|Acc(\gamma_{i-1})|}{N_{i-1}}$ to $(1+3\varepsilon)^{2rm-i} \cdot \frac{|Acc(\gamma_{i-1})|}{N_{i-1}}$ (recall that $2rm$ is the number of bits sent in the protocol). If ε is small enough this leeway is insignificant.

2 Preliminaries

2.1 Concentration Bounds

Lemma 2.1 (Markov's Inequality)**.** *Let X be a non-negative random variable and a be a positive real number. Then:*

$$\Pr[X \geq a] \leq \frac{\mathbb{E}[X]}{a}$$

Lemma 2.2 (Chernoff Bound)**.** *Let $p \in [0,1]$ and $X_1, \ldots, X_n$ be independent 0-1 random variables such that $\Pr[X_i = 1] = p$ for each i. Then for every ε it holds that:*

$$\Pr\left[\left|\frac{1}{n}\sum_{i=1}^{n} X_i - p\right| > \varepsilon\right] < 2 \cdot e^{-2\cdot\varepsilon^2\cdot n}$$

2.2 Protocols and Interactive Proofs

Definition 2.3 (Interactive Protocol)**.** *An Interactive Protocol is a pair $\langle P, V\rangle$ of Interactive Turing Machines that are both run on a common input x, whose length we denote by $n = |x|$. The first machine, is called the prover, and the second machine is called the verifier. For inputs of length n the parties altogether send $m = m(n)$ messages, the verifier runs in time t_P, and the verifier runs in time t_V and uses $\ell = \ell(n)$ bits of randomness. The number of messages in a protocol is sometimes counted in rounds, rather than messages, denoted $r = \frac{m}{2}$, where each round consists of a verifier message and a prover message.*

Remark 2.4. We assume without loss of generality that all protocols begin with a verifier message.

Definition 2.5 (Public-Coin Interactive Protocol)**.** *An Interactive Protocol is public-coin if it can be split into two phases. In the first phase (the communication phase) each message sent from the verifier to the prover is a uniformly distributed random binary string. In the second phase (the verification phase) there is no further communication between the verifier and the prover, and the verifier computes some function of the transcript generated in the first phase.*

In this paper, the prover and verifier will be denoted P and V respectively when talking about general interactive protocols. When wishing to stress that the protocol is public-coin, we denote the prover by M and the verifier by A.

The notion of an interactive proof for a language $\mathcal{L}$ is due to Goldwasser, Micali and Rackoff [GMR85].

Definition 2.6 (Interactive Proof [GMR85])**.** *An interactive protocol is an Interactive Proof (IP) for $\mathcal{L}$ if:*

- *Completeness: For every $x \in \mathcal{L}$, if V interacts with P on common input x, then V accepts with probability $c = c(|x|)$.*
- *Soundness: For every $x \notin \mathcal{L}$, and every computationally unbounded cheating prover strategy P^*, the verifier V accepts when interacting with P^* with probability at most $s = s(|x|)$.*

Definition 2.7 (Arthur-Merlin Protocol [Bab85])**.** *An interactive proof for a language $\mathcal{L}$ is an Arthur-Merlin Protocol (AM) if it is public-coin.*

Definition 2.8 (Doubly-Efficient Interactive Proof [GKR08])**.** *A doubly-efficient interactive proof (deIP) for a language $\mathcal{L}$ is an interactive proof where for joint input $x \in \{0,1\}^n$ both the verifier and the prover run in time polynomial in n. Doubly-Efficient IPs can be private-coin or public-coin.*

Definition 2.9 (Set of Consistent Coins)**.** *Let $\langle P, V\rangle$ be an interactive proof where the verifier uses $\ell = \ell(n)$ coins. For $x \in \{0,1\}^n$ and partial transcript γ denote the set of coins consistent with x and γ by $Coins_x(\gamma)$, that is if $\gamma = \alpha_1, \beta_1, \ldots, \alpha_i, \beta_i$ then*

$$Coins_x(\gamma) = \left\{\rho \in \{0,1\}^\ell \mid V(x;\rho) = \alpha_1 \wedge \forall j \in \{2,\ldots,i\} :\ V(x, \alpha_1, \beta_1, \ldots, \alpha_{j-1}, \beta_{j-1}; \rho) = \alpha_j\right\}$$

where $Coins_x(\emptyset) = \{0,1\}^\ell$. We sometimes abuse notation and let $Coins_x(\gamma)$ also denote the uniform distribution over the set of coins consistent with x and γ.

Definition 2.10 (Set of Accepting Coins)**.** *Let $\langle P, V\rangle$ be an interactive proof. For $x \in \{0,1\}^n$ and partial transcript γ, the set of accepting coins consistent with x and γ, denoted by $Acc_x(\gamma)$, are the coins that are consistent with x and γ, which lead to an accepting transcript. Formally:*

$$Acc_x(\gamma) = \{\rho \in Coins_x(\gamma) \mid \exists \gamma' :\ \rho \in Coins_x(\gamma, \gamma') \text{ and } V(x, (\gamma, \gamma'); \rho) = 1\}$$

Definition 2.11 (Message Distribution)**.** *Let $\langle P, V\rangle$ be an interactive proof, $x \in \{0,1\}^n$ be a global input and γ be a partial transcript ending in a prover message. Then the message distribution of $\langle P, V\rangle$ with respect to x and γ, denoted $Msg_x(\gamma)$ is defined by the following random process:*

- *Sample ρ uniformly from $Coins_x(\gamma)$*
- *Output the next verifier message consistent with ρ: $V_x(\gamma; \rho)$.*

If $\gamma = (\alpha_1, \beta_1, \ldots, \alpha_{i-1}, \beta_{i-1}, \alpha_i)$ ends in a verifier message, then $Msg_x(\gamma)$ is the singleton α_i.

2.3 Statistical Distance

Definition 2.12 (Statistical Distance)**.** *Let P and Q be two distributions over countable set U. The statistical distance between P and Q is defined as follows:*

$$\Delta(P,Q) = \frac{1}{2}\sum_{u\in U}|P(u) - Q(u)|$$

*For $\delta \in [0,1]$, if $\Delta(P,Q) \leq \delta$, we say that P and Q are δ-*close*, and if P and Q are not δ-close, then they are δ-*far*.*

We now give a number of useful claims regarding statistical distance

Claim 2.13 (Basic Properties of Statistical Distance)**.** *Let P, Q and W be distributions over U, A be a (randomized) Turing Machine and E be some event. Then:*

- Triangle Inequality: $\Delta(P,Q) \leq \Delta(P,W) + \Delta(W,Q)$
- Post-Processing: $\Delta(A(P), A(Q)) \leq \Delta(P,Q)$
- Multiple Samples: Let $(P_1,\ldots,P_t)$ and $(Q_1,\ldots,Q_t)$ be t independent samples from P and Q respectively. Then $\Delta((P_1,\ldots,P_t),(Q_1,\ldots,Q_t)) \leq t\cdot\Delta(P,Q)$
- Conditioning on Events: If E and $\neg E$ both have non-zero probability by P and Q, then

$$\Delta(P,Q) \leq \Pr[E]\,\Delta\left(P_{|_E}, Q_{|_E}\right) + \Pr[\neg E]\,\Delta\left(P_{|_{\neg E}}, Q_{|_{\neg E}}\right)$$

Claim 2.14. *Let X be a distribution over universe U and A and B be randomized Turing Machines. Then*

$$\Delta(A(X),B(X)) = \sum_{u\in U}\Pr[X=u]\Delta(A(u),B(u))$$

Proof. Suppose the output of the machines A and B is over universe W. Then:

$$\begin{aligned}
\Delta(A(X),B(X)) &= \frac{1}{2}\sum_{w\in W}|\Pr[A(X)=w] - \Pr[B(X)=w]| \\
&= \frac{1}{2}\sum_{w\in W}\left|\sum_{u\in U}\Pr[X=u]\left(\Pr[A(u)=w] - \Pr[B(u)=w]\right)\right| \\
&= \sum_{u\in U}\Pr[X=u]\left(\frac{1}{2}\sum_{w\in W}|\Pr[A(u)=w] - \Pr[B(u)=w]|\right) \\
&= \sum_{u\in U}\Pr[X=u]\Delta(A(u),B(u))
\end{aligned}$$

□

Claim 2.15. *Let $x_1,\ldots,x_m$ be positive real numbers such that there exists $x_i \neq 0$, $0 \leq \varepsilon \leq \frac{1}{2}$, and $w_1,\ldots,w_m$ be such that*

$$\forall i \in [m]\ (1-\varepsilon)\,x_i \leq w_i \leq (1+\varepsilon)\,x_i$$

Define the distributions X and W as follows:

$$\Pr[X=i] = \frac{x_i}{\sum_{j\in[m]}x_j} \qquad\qquad \Pr[W=i] = \frac{w_i}{\sum_{j\in[m]}w_j}$$

then $\Delta(X,W) \leq 2\varepsilon$.

Proof. Let us inspect the statistical distance between X and W:

$$\Delta(X,W) = \frac{1}{2}\sum_{i\in[m]}\left|\frac{x_i}{\sum_{j\in[m]}x_j} - \frac{w_i}{\sum_{j\in[m]}w_j}\right|$$

Looking at a specific $i \in [m]$, and recalling that $(1-\varepsilon)\,x_i \leq w_i \leq (1+\varepsilon)\,x_i$:

$$\frac{x_i}{\sum_{j\in[m]}x_j} - \frac{(1+\varepsilon)\,x_i}{\sum_{j\in[m]}(1-\varepsilon)\,x_j} \leq \frac{x_i}{\sum_{j\in[m]}x_j} - \frac{w_i}{\sum_{j\in[m]}w_j} \leq \frac{x_i}{\sum_{j\in[m]}x_j} - \frac{(1-\varepsilon)\,x_i}{\sum_{j\in[m]}(1+\varepsilon)\,x_j}$$

which gives us:

$$\frac{(1-\varepsilon)\,x_i - (1+\varepsilon)\,x_i}{(1-\varepsilon)\sum_{j\in[m]}x_j} \leq \frac{x_i}{\sum_{j\in[m]}x_j} - \frac{w_i}{\sum_{j\in[m]}w_j} \leq \frac{(1+\varepsilon)\,x_i - (1-\varepsilon)\,x_i}{(1+\varepsilon)\sum_{j\in[m]}x_j}$$

which in turn implies:

$$\left(\frac{2\varepsilon}{1-\varepsilon}\right)\cdot\frac{-x_i}{\sum_{j\in[m]}x_j} \leq \frac{x_i}{\sum_{j\in[m]}x_j} - \frac{w_i}{\sum_{j\in[m]}w_j} \leq \left(\frac{2\varepsilon}{1+\varepsilon}\right)\frac{x_i}{\sum_{j\in[m]}x_j}$$

and so since $1-\varepsilon > 0$, for each $i \in [m]$:

$$\left|\frac{x_i}{\sum_{j\in[m]}x_j} - \frac{w_i}{\sum_{j\in[m]}w_j}\right| \leq \left(\frac{2\varepsilon}{1-\varepsilon}\right)\cdot\frac{x_i}{\sum_{j\in[m]}x_j}$$

Which finally shows that:

$$\begin{aligned}\Delta(X,W) &\leq \frac{1}{2}\sum_{i\in[m]}\left[\left(\frac{2\varepsilon}{1-\varepsilon}\right)\cdot\frac{x_i}{\sum_{j\in[m]}x_j}\right]\\ &= \frac{\varepsilon}{1-\varepsilon}\\ &\leq 2\varepsilon\end{aligned}$$

□

Claim 2.16. *Fix $m \in \mathbb{N}$. Let X be a discrete probability distribution over universe $[m]$ and $0 \leq \varepsilon \leq \frac{1}{2m}$. Define $x_i = \Pr[X=i]$ and let $w_1 \ldots w_m$ be such that:*

$$\max\{0, x_i - \varepsilon\} \leq w_i \leq x_i + \varepsilon$$

and for at least one index i, $w_i \neq 0$. Define the distribution W as follows:

$$\Pr[W=i] = \frac{w_i}{\sum_{j\in[m]}w_j}$$

then $\Delta(X,W) \leq 2\varepsilon m$.

Proof. We again inspect the statistical distance

$$\Delta(X,W) = \frac{1}{2}\sum_{i\in[m]}\left|x_i - \frac{w_i}{\sum_{j\in[m]}w_j}\right|$$

Noting that

$$x_i - \frac{x_i+\varepsilon}{\sum_{j\in[m]}[x_j-\varepsilon]} \leq x_i - \frac{w_i}{\sum_{j\in[m]}w_j} \leq x_i - \frac{x_i-\varepsilon}{\sum_{j\in[m]}[x_j+\varepsilon]}$$

and that $\sum_{j\in[m]} x_j = 1$ the result is that

$$x_i - \frac{x_i+\varepsilon}{1-\varepsilon m} \leq x_i - \frac{w_i}{\sum_{j\in[m]} w_j} \leq x_i - \frac{x_i-\varepsilon}{1+\varepsilon m}$$

which in turn implies that

$$\frac{(1-\varepsilon m)\, x_i - x_i - \varepsilon}{1-\varepsilon m} \leq x_i - \frac{w_i}{\sum_{j\in[m]} w_j} \leq \frac{(1+\varepsilon m)\, x_i - x_i + \varepsilon}{1+\varepsilon m}$$

and therefore

$$\frac{-\left(\varepsilon m x_i + \varepsilon\right)}{1-\varepsilon m} \leq x_i - \frac{w_i}{\sum_{j\in[m]} w_j} \leq \frac{\varepsilon m x_i + \varepsilon}{1+\varepsilon m}$$

and so, since $1-\varepsilon m > 0$ for every $i \in [m]$

$$\left| x_i - \frac{w_i}{\sum_{j\in[m]} w_j} \right| \leq \frac{\varepsilon m x_i + \varepsilon}{1-\varepsilon m}$$

and so

$$\begin{aligned}
\Delta\left(X, W\right) &\leq \frac{1}{2} \sum_{i\in[m]} \left[\frac{\varepsilon m x_i + \varepsilon}{1-\varepsilon m} \right] \\
&= \frac{\varepsilon m \sum_{i\in[m]} x_i + \varepsilon m}{2\left(1-\varepsilon m\right)} \\
&= \frac{\varepsilon m}{1-\varepsilon m} \\
&\leq 2\varepsilon m
\end{aligned}$$

where the final inequality is since $\varepsilon m < \frac{1}{2}$ □

Claim 2.17. *Let A and B be randomized Turing Machines receiving input from universe U_I and with output from universe U_O, X and Y be distributions over $\{0,1\}^\ell$ such that $\Delta\left(X, Y\right) \leq \delta_1$ Q be a distribution over $\{0,1\}^n$ such that $\Delta\left(A\left(Q\right), B\left(Q\right)\right) \leq \delta_1$ and Q' be a distribution such that for every $u \in U_I$, $\Pr[Q' = u] \leq \alpha \Pr[Q = u]$. Let D be an oracle-aided algorithm such that given oracle K and input from distribution X, makes a single oracle call to K which is distributed according to Q'. Then:*

$$\Delta\left(D^A\left(X\right), D^B\left(Y\right)\right) \leq \delta_1 + \alpha\delta_2$$

Proof. Let us split D given access to oracle K into two algorithms, D_1 and D_2 such that D_1 outputs a state s, and a query answer $K\left(q\right)$, and D_2 receives s and $K\left(q\right)$, and outputs whatever D outputs. That is, D_1 is D up to (and including) the oracle query, and D_2 is D after receiving the oracle's reply. By assumption D_1 given an input distributed according to X outputs query distributed as Q. Therefore

$$\Delta\left(D_1^A\left(X\right), D_1^B\left(X\right)\right) \leq \Delta\left(A\left(Q'\right), B\left(Q'\right)\right)$$

We now show that this expression is upper-bounded by $\alpha\delta_2$. By Claim 2.14, we have that for every distribution W over universe $\{0,1\}^n$, $\Delta\left(A(W), B(W)\right)$ is equivalent to $\sum_{w\in\{0,1\}^n} \Pr[W = w]\Delta\left(A(w), B(w)\right)$, and so:

$$\begin{aligned}
\Delta\left(A(Q'), B(Q')\right) &= \sum_{q\in U_I} \Pr[Q' = q]\Delta\left(A(q), B(q)\right) \\
&\leq \sum_{q\in U_I} \alpha \Pr[Q = q]\Delta\left(A(q), B(q)\right) \\
&= \alpha\Delta\left(A(Q), B(Q)\right) \\
&\leq \alpha\delta_2
\end{aligned}$$

By the post-processing property in Claim 2.13,

$$\Delta\left(D_1^B(X), D_1^B(Y)\right) \leq \Delta(X,Y) \leq \delta_1$$

Which due to the triangle inequality on statistical distance implies that

$$\Delta\left(D_1^A(X), D_1^B(Y)\right) \leq \Delta\left(D_1^A(X), D_1^B(X)\right) + \Delta\left(D_1^B(X), D_1^B(Y)\right) \leq \delta_1 + \alpha\delta_2$$

We now can use post-processing to claim that applying A_2 does not affect the statistical distance of the output:

$$\begin{aligned}\Delta\left(D^A(X), D^B(Y)\right) &= \Delta\left(D_2\left(D_1^A(X)\right), D_2\left(D_1^B(Y)\right)\right)\\ &\leq \Delta\left(D_1^A(X), D_1^B(Y)\right)\\ &\leq \delta_1 + \alpha\delta_2\end{aligned}$$

□

This gives us the following useful corollary, showing that the statistical distance does not pile up, even when samples are taken adaptively (but non-adversarially):

Corollary 2.18 (Adaptive Sampling)**.** *Let A and B be randomized Turing Machines receiving input from universe U_I and with output from universe U_O, X and Y be distributions over $\{0,1\}^\ell$ such that $\Delta(X,Y) \leq \delta_1$ Q be a distribution over $\{0,1\}^n$ such that $\Delta(A(Q), B(Q)) \leq \delta_1$ and Q' be a distribution such that for every $u \in U_I$, $\Pr[Q' = u] \leq \alpha \Pr[Q = u]$. Let D be an oracle-aided algorithm such that given oracle K and input from distribution X, makes at most q calls to K which are all distributed according to Q' (but may be correlated). Then:*

$$\Delta\left(D^A(X), D^B(Y)\right) \leq \delta_1 + q\alpha\delta_2$$

Proof. We split D into q parts, $D_0, \ldots, D_q$, where D_i represents D from after query $i-1$ up until before query number i. That is for any oracle K and distribution W, $D^K(W) = D_q^K(...(D_2^K(D_1^K(W))))$. Let $H_i^K(W) = D_q^K(...(D_2^K(D_1^K(W))))$. Then $W \equiv H_0^K(W)$ and $H_q^K(K) = D^K(W)$. Since in $H_0^A(X)$ and $H_0^B(Y)$, no queries have been made, $\Delta\left(H_0^A(X), H_0^B(Y)\right) \leq \Delta(X,Y) \leq \delta_1$. Fix $0 < i \leq q$ and suppose that $\Delta\left(H_{i-1}^A(X), H_{i-1}^B(Y)\right) \leq \delta_1 + (i-1)\alpha\delta_2$. Then for every oracle K and distribution W, $H_i^K(W) = D_i^K(H_i^K(W))$. Thus, by Claim 2.17,

$$\begin{aligned}\Delta\left(H_i^A(X), H_i^B(X)\right) &\leq \Delta\left(D_i^A(H_{i-1}^A(X)), D_i^B(H_{i-1}^B(X))\right)\\ &\leq \Delta\left(H_i^A(X), H_i^B(X)\right) + \alpha\delta_2\\ &\leq \delta_1 + i\alpha\delta_2\end{aligned}$$

Inductively proving the claim. □

2.4 Hash Functions and Entropy

Intuitively, a good hash function should spread the elements in its domain as uniformly as possible over the image. Throughout this paper we will use $\mathcal{H}_{n,m}$ to denote families of hash functions from $\{0,1\}^n$ to $\{0,1\}^m$. We can quantify this intuition by the following lemmas:

Lemma 2.19 ("Almost-Uniform Cover" of hash functions, [Gol08, Implied by Proof of Lemma D.6])**.** *Let $m \leq n$ and k be integers, $\mathcal{H}_{n,m}$ be a family of $2k$-wise independent hash functions and $S \subseteq \{0,1\}^n$. Then for every $\lambda > 0$ and $y \in \{0,1\}^m$:*

$$\Pr_{h \leftarrow \mathcal{H}_{n,m}}\left[\left|S \cap h^{-1}(y)\right| \notin (1-\lambda, 1+\lambda)\frac{|S|}{2^m}\right] \leq \left(\frac{2k \cdot 2^m}{\lambda^2 |S|}\right)^k$$

This allows us to show the following useful corollary, which says that even if the image chosen for the hash is not chosen arbitrarily, but rather chosen as the hash value of some $x \in S$, there is still strong concentration around a uniform cover over the values y:

Corollary 2.20. *Let $m \leq n$ and k be integers, $\mathcal{H}_{n,m}$ be a family of $2k+1$-wise independent hash functions and $S \subseteq \{0,1\}^n$. Then for every $\lambda \in (0,1)$ and $x \in S$:*

$$\Pr_{h \leftarrow \mathcal{H}_{n,m}}\left[\left|S \cap h^{-1}\left(h\left(x\right)\right)\right| \notin 1 + \left(1-\lambda, 1+\lambda\right)\frac{|S|-1}{2^m}\right] \leq \left(\frac{2k \cdot 2^m}{\lambda^2\left(|S|-1\right)}\right)^k$$

Proof. Due to $2k+1$-wise independence, fixing any y and conditioning on $h\left(x\right) = y$, the marginal distribution over the hash functions remains $2k$-wise independent. Using Lemma 2.19 for $S' = S \backslash \{x\}$ and $y = h\left(x\right)$:

$$\Pr_{h \leftarrow \mathcal{H}_{n,m}}\left[\left|\{\rho \in S' | h\left(\rho\right) = y\}\right| \notin \left(1-\lambda, 1+\lambda\right)\frac{|S'|}{2^m}\right] \leq \left(\frac{2k \cdot 2^m}{\lambda^2 |S'|}\right)^k$$

Plugging in the fact that if $x \in S$, then $|S'| = |S| - 1$ and $\left|S \cap h^{-1}\left(y\right)\right| = \left|S' \cap h^{-1}\left(y\right)\right| + 1$ we get the corollary. □

Lemma 2.21 ([Vad12, Construction 3.32])**.** *For $n \in \mathbb{N}$, $1 \leq k \leq n$ and $1 \leq m \leq n$ there exists a family $\mathcal{H}_{n,m}$ of k-wise independent hash functions. Moreover, a function h can be sampled from $\mathcal{H}_{n,m}$ and evaluated on any $x \in \{0,1\}^n$ in time $O(kn)$.*

Definition 2.22 (Min-Entropy)**.** *Let D be a distribution over $\{0,1\}^\ell$, then the min-entropy of D, denoted $H_\infty\left(D\right)$, is equal to $-\log\left(\max_{d \in \{0,1\}^\ell} \Pr\left[D = d\right]\right)$. Note that $H_\infty\left(D\right) \leq \log(Supp(D))$.*

Lemma 2.23 (Leftover Hash Lemma [ILL89])**.** *Let X be a random variable with universe $\{0,1\}^\ell$ and min-entropy at least k, and let $\mathcal{H}_{\ell,t}$ be a family of 2-universal hash functions[6] for $t \leq k - 2\log\left(1/\varepsilon\right)$, then*

$$\Delta\Big(\left(\mathcal{H}, \mathcal{H}\left(X\right)\right), \left(\mathcal{H}, U_t\right)\Big) \leq \varepsilon$$

2.5 One-Way Functions and Distributional Inversion

Definition 2.24 (One-Way Function)**.** *An efficiently computable function $f : \{0,1\}^n \to \{0,1\}^{m(n)}$ is a (strongly) one-way function (OWF) if for every PPTM A*

$$\Pr_{x \leftarrow U_n}\left[f\left(A\left(1^n, f\left(x\right)\right)\right) = f\left(x\right)\right] = negl\left(n\right)$$

Definition 2.25 (Distributionally One-Way Function [IL89])**.** *An efficiently computable function $f : \{0,1\}^n \to \{0,1\}^{m(n)}$ is distributionally one-way if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ such that for any PPTM A, the distribution defined by $\left(x, f\left(x\right)\right)$ and the distribution defined by $\left(A\left(1^n, f\left(x\right)\right), f\left(x\right)\right)$ have statistical distance at least $\frac{1}{p(n)}$ when $x \leftarrow U_n$.*

The following lemma shows the existence of distributionally one-way functions is equivalent to (regular) one-way functions.

Lemma 2.26 (Impagliazzo and Luby [IL89])**.** *One-way functions exist if and only if distributionally one-way functions exist.*

In this paper we will work with a stronger notion - auxiliary-input one-way functions.

[6] Any pairwise independent hash function is also 2-universal

Definition 2.27 (Auxiliary-Input One-Way Functions [OW93])**.** *An efficiently computable function* $f : \{0,1\}^n \times \{0,1\}^{\ell(n)} \to \{0,1\}^{m(n)}$ *is auxiliary-input one-way (AOWF) if for any PPTM* A*, there exists an infinite set* $\mathcal{Z}_A$ *such that for any* $z \in \mathcal{Z}_A$*:*

$$\Pr_{x \leftarrow U_{\ell(|z|)}} \left[f\left(z, A\left(z, 1^{\ell(|z|)}, f\left(z,x\right)\right)\right) = f\left(z,x\right)\right] = negl\left(|z|\right)$$

And its distributional variant:

Definition 2.28 (Auxiliary-Input Distributionally One-Way Function)**.** *An efficiently computable function* $f : \{0,1\}^n \times \{0,1\}^{\ell(n)} \to \{0,1\}^{m(n)}$ *is auxiliary-input distributionally one-way (ADOWF) if there exists polynomial* $p : \mathbb{N} \to \mathbb{N}$ *such that for any PPTM* A*, there exists an infinite set* $\mathcal{Z}_A$ *where for any* $z \in \mathcal{Z}_A$ *the distribution defined by* $(z, x, f\left(z,x\right))$ *and the distribution defined by* $\left(z, A\left(z, 1^{\ell(n)}, f\left(z,x\right)\right), f\left(z,x\right)\right)$ *have statistical distance at least* $\frac{1}{p(n)}$ *when* $x \leftarrow U_{\ell(n)}$*.*

Claim 2.29. *Auxiliary-input distributionally one-way functions exist if and only if auxiliary-input one-way functions exist.*

The proof of Claim 2.29 is essentially identical to the proof of Lemma 2.26.

Definition 2.30 (δ Auxiliary-Input Distributional Inverter)**.** *Let* $f : \{0,1\}^n \times \{0,1\}^{\ell(n)} \to \{0,1\}^{m(n)}$ *be an efficiently computable function and let* $\delta : \mathbb{N} \to [0,1]$ *be a function. An Auxiliary-Input Distributional Inverter (*δ*-ADI) for* f *is a randomized algorithm* ADI_f *such that there exists a constant* N_0 *such that for every* $n > N_0$ *and* $z \in \{0,1\}^n$ *the distributions* $\left(z, ADI_f\left(z, 1^{\ell(n)}, f\left(z,x\right)\right), f\left(z,x\right)\right)$ *and* $\left(z, x, f\left(z,x\right)\right)$ *where* $x \leftarrow U_{\ell(n)}$ *have statistical distance at most* $\delta\left(n\right)$ *.*

2.6 Pseudo-Random Functions and Permutations

Definition 2.31 (Pseudo-Random Function Family)**.** *A family of functions* $\Psi = \{\Psi_n\}_{n \in \mathbb{N}}$ *where* Ψ_n *are functions of the form* $f_k : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ *indexed by a key* $k \in \{0,1\}^{\tau(n)}$ *is pseudo-random if:*

- For all $x \in \{0,1\}^n$ and $k \in \{0,1\}^{\tau(n)}$, $f_k\left(x\right)$ can be efficiently evaluated given x and k.
- For any oracle-aided PPTM D:

$$\left| \Pr_{k \leftarrow \{0,1\}^{\tau(n)}} \left[D^{f_k}\left(1^n\right) = 1\right] - \Pr_{f \leftarrow R_n} \left[D^{f}\left(1^n\right) = 1\right] \right| = negl\left(n\right)$$

  where R_n is the set of random functions from $\{0,1\}^n$ to $\{0,1\}^{\ell(n)}$.

Lemma 2.32 (Goldreich, Goldwasser and Micali [GGM86])**.** *If one-way functions exist, then there are psudo-random functions.*

Definition 2.33 (Strong Pseudo-Random Permutation Family)**.** *A family of permutations* $\Psi = \{\Psi_n\}_{n \in \mathbb{N}}$ *where* Ψ_n *are permutations of the form* $f_k : \{0,1\}^n \to \{0,1\}^n$ *indexed by a key* $k \in \{0,1\}^{\tau(n)}$ *is strongly pseudo-random if:*

- For all $x \in \{0,1\}^n$ and $k \in \{0,1\}^{\tau(n)}$, $f_k\left(x\right)$ and f_k^{-1} can be efficiently evaluated given x and k.
- For any oracle-aided PPTM D:

$$\left| \Pr_{k \leftarrow \{0,1\}^{\tau(n)}} \left[D^{f_k, f_k^{-1}}\left(1^n\right) = 1\right] - \Pr_{f \leftarrow \Pi_n} \left[D^{f, f^{-1}}\left(1^n\right) = 1\right] \right| = negl\left(n\right)$$

  where Π_n is the set of random permutations over $\{0,1\}^n$.

Lemma 2.34 (Luby and Rackoff [LR88])**.** *If one-way functions exist, then there are strong psudo-random permutations.*

2.7 Partitions and Histograms

A common theme in this paper is that we wish to take a large set, split it into a number of partitions, and then restrict ourselves to one partition based on the probability that a random sample from the set lands inside this partition.

Definition 2.35 (Partitioning)**.** *A partitioning of a set $S \subseteq \{0,1\}^n$ into I subsets, denoted by $\mathbb{S}$, is a set of I non-empty subsets of S, $S_1, \ldots, S_I \subseteq S$ such that $\cup_{i=1}^{I} S_i = S$ and for each $i \neq j$, S_i and S_j are disjoint. A partitioning of a distribution D, denoted $\mathbb{D}$, is defined by a partitioning of the set $Supp(D)$ into I parts, where distribution D_i is the distribution of D conditioned on D landing in partition i of $Supp(D)$.*

We now define the notion of a histogram of a partitioning, which will allow us to choose a partition to work on relative to its weight:

Definition 2.36 ((η,σ)-Histogram)**.** *Let D be a distribution over $S \subseteq \{0,1\}^\ell$, and $\mathbb{S} = \{S_i\}_{i=1}^{I}$ be a partitioning of S into I subsets. Then for $0 \leq \eta, \sigma \leq 1$ an (η,σ)-histogram with regards to $\mathbb{S}$ and D is a set of values $(p_1, \ldots, p_I)$ such that for all $i \in [I]$:*

1. *$|p_i - \Pr_{\rho \leftarrow D}[\rho \in S_i]| \leq \sigma + \eta$. Moreover, if $p_i \neq 0$ then $|p_i - \Pr_{\rho \leftarrow D}[\rho \in S_i]| \leq \eta$*
2. *The value p_i is either equal to 0 or greater than σ*

If $\eta = \sigma$ then we say that it is a η-Histogram.

A useful fact implied by the above definition is that for a (η,σ)-Histogram, if $p_i = 0$ then $\Pr_{\rho \leftarrow D}[\rho \in S_i] \leq \sigma + \eta$ and alternatively, if $p_i \neq 0$ then $\Pr_{\rho \leftarrow D}[\rho \in S_i] \geq \sigma - \eta$. Additionally, note that for any D and partitioning $\mathbb{S} = \{S_i\}_{i \in [I]}$, a 0-Histogram of $\mathbb{S}$ is one such that for any $i \in [I]$, $p_i = \Pr_{\rho \leftarrow D}[\rho \in S_i]$.

2.8 Coupon Collector

In a number of proofs in this paper we are given an oracle for uniformly sampling from a set $T \subseteq \{0,1\}^\ell$ and are required to either sample all elements of T (if it is small) or recognize that T is large. We do this by taking many samples, and inspecting the number of distinct elements which we have sampled.

Claim 2.37 (Coupon Collector from a Set)**.** *Let $T \subseteq \{0,1\}^\ell$ be a set. For $q \geq 3$, let S be a set of distinct elements formed by q^3 uniform samples in T. Then*

$$\Pr[|S| < \min\{q, |T|\}] \leq \frac{1}{q}$$

Proof. This is essentially a "coupon collector" problem - for some set, how many random samples must we take until we have seen each unique item. Let $W \subseteq T$ be a set of $k = \min\{|T|, q\}$ elements. It is well known that the expected number of uniformly chosen samples from W required until all distinct elements in W are seen is less than $k \log k + k + 1$.[7] Then letting ξ be the random variable indicating how many samples are needed until every unique item is sampled, by Markov's inequality

$$\Pr[|S| < \min\{q, |T|\}] = \Pr\left[\xi \geq q^3\right] \leq \frac{k \log k + k + 1}{q^3}$$

Since $q > 3$, $\frac{k \log k + k + 1}{q^3} \leq \frac{1}{q}$ as required. □

[7] More precisely, the expectation is $k \cdot H_k$ where H_k is the k-th harmonic number

3 Algorithms and Protocols Used

3.1 Organization and Oracle Naming Convention

Organization: This section is split into four major subsections. We explain the role each one, while referring to Section 1.1 for the bigger picture.

1. Section 3.2 contains our variant of the set-lower bound/sampling protocol shown in Section 1.1.1. Our protocol enables an efficient verifier to sample from a distribution D that has small min-entropy. The prover is efficient if it has access to an oracle that, given a random hash function h and a random image y, can sample random elements from D conditioned on $h(D) = y$. Recall that in Section 1.1.3 we had the problem that since there does not necessarily exist an efficient function that samples in a cluster, the prover will have to complete the sampling protocol while having access to an inversion oracle for the entire verifier, and hope that sampled preimages land inside the required cluster. In Section 3.2.4 we analyze this in a slightly different setting. There is a distribution D and set S in the support of D, where the verifier wishes to sample from the distribution D conditioned on landing in S. We show that if S has sufficient weight under D and the ratio between its heaviest and lightest elements is small (i.e. $\frac{\max_{\rho \in S} \Pr[D=\rho]}{\min_{\rho \in S} \Pr[D=\rho]}$ is small), then the prover can be made efficient if it can sample random elements from D conditioned on $h(D) = y$.

2. Section 3.3 describes how to, given a partition of the support into subsets, build an approximate histogram of a distribution. The algorithm is efficient given a sampling oracle for the distribution and a membership oracle for the subsets. For our prover-efficient emulation these subsets will be the approximate clusters described in Point 1 of Section 1.1.4.

3. Section 3.4 describes a generic "sampling via subsets" protocol. Given a distribution D and a partitioning of its support into subsets, the goal of the honest prover is to help the verifier sample an element from the distribution, along with its subset index. For private-coin to public-coin emulation the distribution will be the verifier message distribution and the subsets will be the approximate clusters. Thus, sampling an element with its subset index means sampling a verifier message along with an approximation of the number of coins that are consistent with the message. This protocol is first described simply in the case where the prover has inversion oracles for each cluster separately. That is, the prover is able to, for every subset S, given a hash function h and image y, sample from D conditioned on landing in S and $h(D) = y$. This protocol uses as a sub-protocol the sampling protocol of Section 3.2, which requires the verify know a lower-bound on the min-entropy of the distribution it is sampling from. To accommodate this the verifier receives a function that, if the prover is honest, for every subset gives a lower-bound on the min-entropy of D conditioned on landing in the subset. As noted in Section 1.1.3 in general we cannot hope to invert functions to allow us to sample conditioned in landing in a cluster. Section 3.4.4 gives an analysis showing that we can still execute a subsets-via-sampling protocol with efficient prover if: (1) For every subset, D conditioned on landing in the subset is statistically close to a distribution with small min entropy. (2) In the case of an honest prover, the verifier can lower-bound this min-entropy. (3) For each subset the ratio of its heaviest elements to its lightest elements is small. These properties all hold for our prover-efficient public-coin emulation.

4. Section 3.5 contains the algorithm for approximate counting from inversion alluded to in Section 1.1.2.

Oracle Naming Convention: In this section, in order to allow for some abstraction, our protocols and algorithms access certain functionalities through oracles. We aggregate here a full explanation for each of the types of oracles used as a reference to the reader. Oracles will be denoted by the Sans Serif font.

Samp_D^δ: Parameterized by a distribution D over $\{0,1\}^\ell$. Receives no input and returns a value such that $\Delta\left(\mathsf{Samp}_D^\delta, D\right) \leq \delta$.

Prover-Efficient Sampling Protocol

Joint Input: $N \in [2^m]$

Parameters: $m \in \mathbb{N}$, $0 < \varepsilon < 1$

Prover Oracle Access: $\mathsf{HashInv}_D^\delta$

1. Set $k \triangleq \max\left\{0, \left\lfloor \log_2\left(\frac{\varepsilon^3 N}{8}\right)\right\rfloor\right\}$, and let $\mathcal{H}_{m,k}$ be a family of pairwise independent hash functions.
2. Verifier:
   (a) Choose and send $h \leftarrow \mathcal{H}_{m,k}$ and $y \leftarrow \{0,1\}^k$.
3. Prover:
   (a) Receive h, y.
   (b) Sample $\rho \leftarrow \mathsf{HashInv}_D^\delta(k, h, y)$ and send it to the verifier.
4. Verifier:
   (a) Receive ρ. Reject if $h(\rho) \neq y$.
5. Both parties output ρ.

Figure 1: Prover-Efficient Public-Coin Sampling Protocol

$\mathsf{HashInv}_D^\delta$: Parameterized by a distribution D over $\{0,1\}^\ell$. Receives as input an index $k \in [\ell]$, a function $h \in \mathcal{H}_{\ell,k}$ and an image $y \in \{0,1\}^k$. The output is such that $\Delta\left(\mathsf{HashInv}_D^\delta(k,h,y), D_{|h(D)=y}\right) \leq \delta$ when $h \leftarrow \mathcal{H}_{\ell,k}$ and $y \leftarrow h(D)$. In order to simplify notation, if $k = 0$, i.e. the hash function has no output, then we define $\mathsf{HashInv}_D^\delta(0,\cdot,\cdot)$ to be identical to Samp_D^δ.

$\mathsf{Subset}_{D,\mathbb{D}}^\delta$: Parametrized by a distribution D over $\{0,1\}^\ell$ and a partitioning of D, $\mathbb{D} = \{D_i\}_{i=1}^I$. Receives as input a value $\rho \leftarrow D$. With probability $1-\delta$ returns $i \in [I]$ such that $\rho \in Supp(D_i)$.

$\mathsf{InSet}_{D,S}^\delta$: Parametrized by a distribution D over $\{0,1\}^\ell$ and set $S \subseteq \{0,1\}^\ell$. Receives as input a value $\rho \leftarrow D$. With probability $1-\delta$ returns whether $\rho \in S$.

3.2 A Prover-Efficient Sampling Protocol

We construct a constant-round public-coin sampling protocol with an efficient prover, where the prover has access to an oracle. In this protocol, which can be viewed as a generalization of the "set lower-bound" protocol used in [GS86], the honest prover is trying to help the verifier to to sample from a distribution D of entropy at least $\log N$, while a malicious prover tries to cause the verifier to sample from a set S where $|S| \ll N$. The protocol is roughly as follows: The verifier chooses a random pairwise independent hash function h, and a random image y. The prover then samples an element from the distribution D conditioned on $h(D) = y$ and sends it back to the verifier. If the entropy of D is high, this should not be a problem, but for any set S that is is much smaller than the claimed N if the range of the hash function is of an appropriate size then (w.h.p) no such element exists. The most computationally difficult task in the protocol is for the honest prover to sample an element from D conditioned on $h(D) = y$. We therefore give it access to an oracle that given y and h does this sampling.

Lemma 3.1 (Properties of the Sampling Protocol, Figure 1)**.** *Let $m \in \mathbb{N}$, $0 < \varepsilon < 1$ and let $\mathcal{H}$ be a family of pairwise independent hash functions. Then the protocol described in Figure 1 is a public-coin interactive protocol with the following properties:*

Verifier Efficiency: Running time $O(m)$, Randomness $O(m)$

Prover Efficiency: Running time $O(m)$*, Oracle calls* 1

Number Of Messages: 2

Completeness: Let D *be a distribution over* $\{0,1\}^m$ *and* N *be such that* $0 \leq \log N \leq H_\infty(D)$*. If the honest prover is given oracle access to* $\mathsf{HashInv}_D^\delta$*, then the output of the protocol has statistical distance at most* $\frac{\varepsilon}{2} + \delta$ *from taking a sample from* D*.*

Soundness: For any set $S \subseteq \{0,1\}^m$ *and* $N \in [2^m]$ *and any unbounded malicious prover, the probability that the verifier ends the execution with an element from* S *is at most* $\frac{8}{\varepsilon^3} \cdot \frac{|S|}{N}$*.*

3.2.1 Efficiency

We begin by analyzing the verifier's efficiency. The verifier must first generate a pairwise independent hash function. Generating this hash function from $\{0,1\}^m$ to $\{0,1\}^k$ takes $O(m)$ time and randomness. It then generates y which takes time and randomness $O(k) = O(m)$, and sends h and y to the prover. It receives one element, ρ, and tests that $h(\rho) = y$, which takes time $O(m)$. Therefore the verifier runs in time and uses randomness $O(m)$. We now analyze the running time of the prover. The prover receives h, y each of which is of length $O(m)$ and then makes a singe query to its oracle. Therefore the total running time of the prover is $O(m)$.

3.2.2 Completeness

Suppose D is a distribution over $\{0,1\}^m$ and N is such that $\log N \leq H_\infty(D)$. We give four hybrids:

- H_1: The output of the protocol as described in Figure 1.
- H_2: The protocol is as in H_1 except the verifier chooses $y \leftarrow h(D)$ rather than uniformly.
- H_3: The protocol is as in H_2, except the prover is given access to $\mathsf{HashInv}_D^0$ rather than $\mathsf{HashInv}_D^\delta$.
- H_4: Output a random sample from D.

We show in Claim 3.2 that $\Delta(H_1, H_2) \leq \frac{\varepsilon}{2}$, in Claim 3.3 that $\Delta(H_2, H_3) \leq \delta$ and in Claim 3.4 that H_3 and H_4 are identical. Thus by the triangle inequality H_1 and H_4 are of statistical distance at most $\frac{\varepsilon}{2} + \delta$.

Claim 3.2. *The statistical distance between* H_1 *and* H_2 *is at most* $\frac{\varepsilon}{2}$*.*

Proof. We show that the statistical distance between $(\mathcal{H}, U_k)$ and $(\mathcal{H}, \mathcal{H}(D))$ is at most $\frac{\varepsilon}{2}$. Since this is the only difference between hybrids 1 and 2, this will imply that the hybrids are close. The distribution D has min-entropy at least $\log N$ and the hash function compresses to k bits where $k = \lfloor \log N - 3\log(2/\varepsilon) \rfloor$, and so $k \leq H_\infty(D) - 3\log(2/\varepsilon)$. By the Leftover Hash Lemma (Lemma 2.23), the distributions $(\mathcal{H}, \mathcal{H}(D))$ and $(\mathcal{H}, U_k)$ have statistical distance at most $\frac{\varepsilon}{2}$. □

Claim 3.3. *The statistical distance between* H_2 *and* H_3 *is at most* δ*.*

Proof. In both oracles the call to the oracle is done from the distribution expected, namely $(\mathcal{H}, \mathcal{H}(D))$. Then by adaptive sampling (Corollary 2.18) the statistical distance of the honest prover using $\mathsf{HashInv}_D^\delta$ is at most δ-far from the distribution when the prover has access to the perfect oracle $\mathsf{HashInv}_D^0$. □

Claim 3.4. *The distributions* H_3 *and* H_4 *are identical.*

Proof. The values h and y are drawn from the distribution expected by $\mathsf{HashInv}^0_D$, and so $\mathsf{HashInv}^0_D(k,h,y)$ returns a random element in D conditioned on $h(D)=y$. Fix $h\in\mathcal{H}$, then for every $\rho\in Supp(D)$:

$$\begin{aligned}\Pr[H_3=\rho|h \text{ chosen}] &= \Pr_{d\leftarrow D}[d=\rho|h(d)=h(\rho)]\Pr_{y\leftarrow h(D)}[h(\rho)=y]\\ &= \Pr_{d\leftarrow D}[d=\rho|h(d)=h(\rho)]\Pr_{d\leftarrow D}[h(\rho)=h(d)]\\ &= \Pr_{d\leftarrow D}[d=\rho]\\ &= \Pr[H_4=\rho]\end{aligned}$$

Since this is true for every h, we can remove the conditioning to see that for every $\rho\in Supp(D)$, $\Pr[H_3=\rho]=\Pr[H_4=\rho]$. □

3.2.3 Soundness

We now inspect the probability that a dishonest prover will be able to cause the verifier to sample an element from some $|S|<N$. Letting ρ be the output of the protocol, we wish to show that

$$\Pr[\rho\in S]\leq\frac{8}{\varepsilon^3}\cdot\frac{|S|}{N}$$

If $N\leq\frac{8}{\varepsilon^3}$ then the ratio $\frac{|S|}{N}$ is so small that multiplying it with $\frac{8}{\varepsilon^3}$ yields a value greater than 1, making the claim trivial. In more detail, $1\leq|S|$, and $N\leq\frac{8}{\varepsilon^3}$ which means that $\frac{\varepsilon^3}{8}\leq\frac{|S|}{N}$, and so $1\leq\frac{8}{\varepsilon^3}\cdot\frac{|S|}{N}$. We now look at the case that $N>\frac{8}{\varepsilon^3}$, in which $k>0$. The prover succeeds in causing the verifier to end up with a value in S if given h and y it can find an element in $S\cap h^{-1}(y)$, since sending this element will make the verifier accept. Therefore, fixing $h\in\mathcal{H}_{m,k}$:

$$\begin{aligned}\Pr[\rho\in S|h \text{ chosen}] &\leq \sum_{y\in\{0,1\}^k}\Pr[\text{y chosen}]\cdot I\left[S\cap h^{-1}(y)\neq\emptyset\right]\\ &\leq \frac{1}{2^k}\sum_{y\in\{0,1\}^k}\left|S\cap h^{-1}(y)\right|\\ &= \frac{|S|}{2^k}\end{aligned}\tag{3.1}$$

Where $I\left[S\cap h^{-1}(y)\neq\emptyset\right]$ is equal to one if and only if $S\cap h^{-1}(y)\neq\emptyset$ and is otherwise equal to zero. Equation (3.1) is true because $|S|=\sum_{y\in\{0,1\}^k}\left|S\cap h^{-1}(y)\right|$. Note that the bound on the probability that the prover wins (i.e. $\rho\in S$) conditioned on h does not depend on h, and so we can remove the conditioning to get

$$\Pr[\rho\in S]\leq\frac{|S|}{2^k}\leq\frac{8}{\varepsilon^3}\cdot\frac{|S|}{N}$$

Where the final inequality is reached by substituting: $2^k=2^{\left\lfloor\log_2\left(\frac{\varepsilon^3 N}{8}\right)\right\rfloor}\geq\frac{N\varepsilon^3}{8}$.

3.2.4 A Variation on the Sampling Protocol

Looking ahead, the protocol as described in Figure 1 will not be sufficient for our uses. We introduce two changes which will satisfy our requirements.

The first is that finding a lower-bound for the min-entropy of a distribution may be a very difficult task. In some cases it is however possible to lower-bound the min-entropy of a distribution which is close to D. Consider the following: Suppose D is λ-close to a flat distribution Z for which we the size of the support is known. Since Z is flat its min-entropy can be easily found. Our observation, is that while a lower-bound on the min-entropy of Z is not necessarily a bound on the min-entropy of D, the fact that they are statistically close implies that sampling with this claim does not skew the output probability much.

The second variation we consider is that we require the protocol to work even when the prover is given $\mathsf{HashInv}$ for a distribution for which is larger than the one we wish to sample from. More precisely we will be interested in using the protocol for the distribution of D conditioned on the event $D \in S$ for some set $S \subseteq Supp(D)$, which we denote by D_S, but where the prover has access to $\mathsf{HashInv}_D$ rather than the expected $\mathsf{HashInv}_{D_S}$. In Figure 2, we show how to emulate $\mathsf{HashInv}_{D_S}$ using $\mathsf{HashInv}_D$. The algorithm, on inputs $(h, y) \leftarrow (\mathcal{H}, \mathcal{H}(D_S))$, takes multiple samples using its oracle $\mathsf{HashInv}_D$, and returns a sample which lands in S if one exists. Note that this procedure requires the ability to recognize whether an element belongs to S.

Putting both variants together yields the following corollary to Lemma 3.1:

Corollary 3.5 (Variation of the Sampling Protocol)**.** *There exists a public-coin interactive protocol which is a variation on Lemma 3.1 where the honest prover is given additional inputs $0 < \kappa, \mu \leq 1$ such that the prover efficiency and completeness properties are changed to the following:*

Prover Effieicency: Running time $O\left(\frac{\mu m^3}{\varepsilon^2 \kappa}\right)$, Oracle calls $O\left(\frac{\mu m^3}{\varepsilon^2 \kappa}\right)$

Completeness: Let D be a distribution over $\{0,1\}^m$, $S \subseteq Supp(D)$ be a set, Z be a distribution over S. Then letting D_S represent D conditioned on landing in S, if:

- *The honest prover is given access to $\mathsf{HashInv}_D^\delta$ and $\mathsf{InSet}_{D,S}^\delta$.*
- *The distributions D_S and Z are statistically close: $\Delta(D_S, Z) \leq \lambda$.*
- *The min-entropy of Z is lower-bounded by $\log N$: $0 < \log N \leq H_\infty(Z)$.*
- *The probability of landing in S is bounded by κ: $\kappa \leq \Pr[D \in S]$.*
- *The ratio of the maximum and minimum probabilities of the elements in S with regards to D are bounded by μ: $\mu \geq \frac{\max_{\rho \in S} \Pr[D=\rho]}{\min_{\rho \in S} \Pr[D=\rho]}$.*

Then the output distribution of the protocol and D are of statistical distance at most:

$$\lambda + \frac{3}{2}\varepsilon + \frac{80m^3\mu}{\kappa^2\varepsilon^2} \cdot \delta$$

The remaining properties are identical to those specified in Lemma 3.1.

Proof. The honest prover is identical to the one described in Lemma 3.1 given N, except the single call to $\mathsf{HashInv}_{D_S}$ will be replaced by the simulator described in Figure 2. The simulator is given access to $\mathsf{HashInv}_D^\delta$ and $\mathsf{InSet}_{D,S}^\delta$ and parameters κ and μ. In order to prove the corollary we introduce four hybrids:

- H_1': The output of the protocol as described above.
- H_2': Identical to H_1', except the verifier chooses $y \leftarrow h(D)$ (exactly as H_2 in Section 3.2.2).
- H_3': Identical to H_2', except the prover has access to perfect oracles $\mathsf{HashInv}_D^0$ and $\mathsf{InSet}_{D,S}^0$ rather than $\mathsf{HashInv}_D^\delta$ and $\mathsf{InSet}_{D,S}^\delta$.
- H_4': Take a random sample from D_S.

By Claims 3.6, 3.7 and 3.9, $\Delta(H_1', H_2') \leq \lambda + \frac{\varepsilon}{2}$, $\Delta(H_2', H_3') \leq \frac{80m^3\mu}{\kappa^2\varepsilon^2} \cdot \delta$ and $\Delta(H_3', H_4') \leq \varepsilon$. By the triangle inequality for statistical distance, we have that $\Delta(H_1', H_4') \leq \lambda + \frac{3}{2}\varepsilon + \frac{80m^3\mu}{\kappa^2\varepsilon^2} \cdot \delta$.

Turning to analysis of the rest of the properties of the algorithm, by Claim 3.7, the prover runs in time $O\left(\frac{\mu m^3}{\varepsilon^2 \kappa}\right)$ and makes $O\left(\frac{\mu m^3}{\varepsilon^2 \kappa}\right)$ queries to its oracles, and each oracle call has multiplicative cost $\frac{1}{\kappa}$. □

We first show that even though the parties do not necessarily has a lower-bound to the entropy of D_S, the protocol output does not change by much:

HashInv Simulator: $HSim$

Parameters: $0 < \varepsilon \leq \frac{1}{2}$, $0 < \kappa \leq 1$, $0 < \mu \leq 1$.

Input: $k \in [m]$, $h \in \mathcal{H}_{m,k}$, $y \in \{0,1\}^k$

Oracle Access: $\mathsf{HashInv}_D^\delta$, $\mathsf{InSet}_{D,S}^\delta$

Algorithm:

1. For $i = 1$ to $\frac{40m^3}{\varepsilon^2} \cdot \frac{\mu}{\kappa}$:
   (a) Let $\rho \leftarrow \mathsf{HashInv}_D^\delta(k, h, y)$ be a fresh sample.
   (b) If $\mathsf{InSet}_{D,S}^\delta(\rho) = True$, output ρ and quit.
2. If no element has been found, output $FAIL$.

Figure 2: HashInv Simulator, $HSim$

Claim 3.6. *The statistical distance between H_1' and H_2' is at most $\lambda + \frac{\varepsilon}{2}$.*

Proof. The proof is nearly identical to that of Claim 3.2. By the Leftover Hash Lemma, since $k = \lfloor \log N - 3\log(2/\varepsilon) \rfloor \leq H_\infty(Z) + 3\log(2/\varepsilon)$, we have that $(\mathcal{H}, U_k)$ and $(\mathcal{H}, \mathcal{H}(Z))$ have statistical distance $\frac{\varepsilon}{2}$. Since $\Delta(D, Z) \leq \lambda$, we have that $\Delta((\mathcal{H}, \mathcal{H}(Z)), (\mathcal{H}, \mathcal{H}(D))) \leq \lambda$, and so $(\mathcal{H}, U_k)$ and $(\mathcal{H}, \mathcal{H}(D))$ have statistical distance $\lambda + \frac{\varepsilon}{2}$. □

Now that we have that the choice h, y of the verifier are close to a useful distribution, we show that if the hash inversion simulator uses imperfect oracles, it is still close to its ideal output:

Claim 3.7. *The statistical distance between H_2' and H_3' is at most $\frac{80m^3\mu}{\kappa^2\varepsilon^2} \cdot \delta$.*

Proof. Consider the following process, A: Receive h and y. Take a sample $\rho \leftarrow \mathsf{HashInv}_D^\delta(k, h, y)$ and $i' \leftarrow \mathsf{InSet}_{D,S}^\delta(\rho)$. Output ρ and i'. The process A represents one iteration of the $HSim$ algorithm. Now, consider A' which is identical to A, but with access to $\mathsf{HashInv}_D^0$ and $\mathsf{InSet}_{D,S}^0$. Then by adaptive sampling, we have that $\Delta(A(\mathcal{H}, \mathcal{H}(D)), A'(\mathcal{H}, \mathcal{H}(D))) \leq 2\delta$ (one δ coming from the difference in ρ and the second from the difference in i'). The actual input to $HSim$, and therefore also to A, comes from $\mathcal{H}, \mathcal{H}(D_S)$. Since $\Pr[D \in S] \geq \kappa$, we have that for every $\rho \in S$,

$$\Pr[D = \rho | D \in S] = \frac{\Pr[D \in S | D = \rho] \Pr[D = \rho]}{\Pr[D \in S]} \leq \frac{1}{\kappa} \Pr[D = \rho]$$

Thus, by adaptive sampling Corollary 2.18, the statistical distance between $A(\mathcal{H}, \mathcal{H}(D_S))$ and $A'(\mathcal{H}, \mathcal{H}(D_S))$ is at most $\frac{2\delta}{\kappa}$.

An entire execution of $HSim$ consists of $\frac{40m^3}{\varepsilon^2} \cdot \frac{\mu}{\kappa}$ such iterations, and so the statistical distance between $HSim$ when given $\mathsf{HashInv}_D^\delta$ and $\mathsf{InSet}_{D,S}^\delta$ and the same algorithm when given $\mathsf{HashInv}_D^0$ and $\mathsf{InSet}_{D,S}^0$ is at most $\frac{80m^3\delta}{\varepsilon^2} \cdot \frac{\mu}{\kappa^2}$. □

We would like to show that H_3' and H_4' are close, but in order to do so, we first give a helpful claim, which says that with high probability over the choice of the hash function h, for every y, the weight of the elements in D_S that map by h to y is not very small. This will mean that after taking many samples of $D_{|h(D)=y}$, it is likely that the prover will sample at least one element in S.

Claim 3.8. *Let $0 < \varepsilon \leq \frac{1}{2}$, D be a distribution over $\{0,1\}^m$, $S \subseteq Supp(D)$ be a set such that $\Pr[D \in S] \geq \kappa$ and $\mu \geq \frac{\max_{\rho \in S} \Pr[D=\rho]}{\min_{\rho \in S} \Pr[D=\rho]}$, and $\mathcal{H}_{m,k}$ be a set of pairwise independent hash functions for $k \leq \log|S| + \log_2\left(\frac{\varepsilon^3}{8}\right)$.*

Then for every $y \in \{0,1\}^k$, with probability $1 - \frac{\varepsilon}{2}$ over the choice of $h \leftarrow \mathcal{H}_{m,k}$:

$$\Pr_{d \leftarrow D}[d \in S | h(d) = y] \geq \frac{\kappa}{\mu} \cdot \frac{\varepsilon}{40m^2}$$

Proof. We begin by showing that for any fixed $y \in \{0,1\}^k$, with high probability over the choice of h, the number of elements in S which are mapped by h to y is not too small. Note that since $H_\infty(D_S) \leq \log_2(|S|)$, $\frac{2^k}{|S|} \leq \frac{\varepsilon^3}{8}$. We utilize the uniform-cover property of pairwise independent hash functions (Lemma 2.19) to get:

$$\Pr_{h \leftarrow \mathcal{H}_{m,k}}\left[\left|S \cap h^{-1}(y)\right| < (1-\varepsilon)\frac{|S|}{2^k}\right] \leq \frac{2 \cdot 2^k}{\varepsilon^2 |S|} \leq \frac{\varepsilon}{4}$$

Thus with probability $1 - \frac{\varepsilon}{4}$ there are at least $(1-\varepsilon)\frac{|S|}{2^k}$ elements mapped to y.

We now show that it is unlikely that the number of elements mapped to y from S is very small when compared with the rest of the support of D. We do this by splitting the support of D into partitions, each of which are close to uniform, and showing that it is unlikely that any of these partitions will have "too many" elements mapped to y. Let $\lambda = 1/m$. We partition the support of D into $I = \log_{1+\lambda}(2^m) \approx m^2$ subsets $\mathbb{D} = \{D_i\}_{i=1}^I$. Subset D_i is the set of elements $\rho \in Supp(D)$ such that $(1+\lambda)^{-(i+1)} \leq \Pr_{d \leftarrow D}[d = \rho] \leq (1+\lambda)^{-i}$. Let $p_H = \max_{\rho \in S} \Pr[D \in \rho]$ and $p_L = \min_{\rho \in S} \Pr[D \in \rho]$. Notice that:

$$\frac{|S|}{\sum_{i=1}^I (1+\lambda)^{-i} |Supp(D_i)|} \geq \frac{\kappa}{(1+\lambda)p_H} \tag{3.2}$$

Since $|S| \geq \frac{\Pr[D \in S]}{p_H}$ and $|Supp(D_i)| \geq (1+\lambda)^{i+1} \Pr[D \in D_i]$. The I subsets can be split in to two categories:

- I_{small}: Subsets D_i such that $\frac{|Supp(D_i)|}{2^k} < \frac{\varepsilon}{4I}$
- I_{big}: Subsets D_i such that $\frac{|Supp(D_i)|}{2^k} \geq \frac{\varepsilon}{4I}$

Subsets which are in I_{small} are so small that with high probability no element will be mapped by h to y, and subsets in I_{big} will with high probability have no more than $\left(1 + \frac{6I}{\varepsilon}\right)\frac{|Supp(D_i)|}{2^k}$ such elements. Details follow:

- I_{small}: Fix $i \in I_{small}$. Since the hash function family is 1-wise independent, for every $\rho \in Supp(D_i)$, $\Pr_h[h(\rho) = y] = 2^{-k}$. Taking the union bound over all elements in the support of D_i, we get that with probability at least $1 - 2^{-k}|Supp(D_i)| \geq 1 - \frac{\varepsilon}{4I}$ no element in D_i is mapped by h to y.
- I_{big}: Fix $i \in I_{big}$. By Lemma 2.19, setting $\lambda = \frac{6I}{\varepsilon} > \sqrt{32} \cdot \frac{I}{\varepsilon}$ we have

$$\begin{aligned}
\Pr_{h \leftarrow \mathcal{H}_{m,k}}\left[\left|Supp(D_i) \cap h^{-1}(y)\right| > \left(1 + \frac{6I}{\varepsilon}\right)\frac{|Supp(D_i)|}{2^k}\right] &\leq \frac{2 \cdot 2^k}{\lambda^2 |Supp(D_i)|} \\
&\leq \frac{8I}{\varepsilon\lambda^2} \\
&\leq \frac{8I}{\varepsilon} \cdot \frac{\varepsilon^2}{32I^2} \\
&\leq \frac{\varepsilon}{4I}
\end{aligned}$$

Taking the union bound over all I subsets, we have that with probability $1 - \frac{\varepsilon}{4}$ no subset in I_{small} has any element mapped to y and no subsets in I_{big} has more than $\left(1 + \frac{6I}{\varepsilon}\right)\frac{|Supp(D_i)|}{2^k}$ elements. Thus, with probability at least $1 - \frac{\varepsilon}{2}$:

$$\left|S \cap h^{-1}(y)\right| \geq (1-\varepsilon)\frac{|S|}{2^k} \tag{3.3}$$

and

$$\forall i \in I: \ \left|Supp(D_i) \cap h^{-1}(y)\right| \leq \left(1 + \frac{6I}{\varepsilon}\right) \frac{|Supp(D_i)|}{2^k} \tag{3.4}$$

Conditioned on this event occurring we have the following:

$$\begin{aligned}
\Pr\left[D \in S | h(D) = y\right] &= \frac{\sum_{\rho \in S \cap h^{-1}(y)} Pr_{d \leftarrow D}[d = \rho]}{\sum_{\rho \in Supp(D) \cap h^{-1}(y)} Pr_{d \leftarrow D}[d = \rho]} \\
&\geq \frac{p_L \left|S \cap h^{-1}(y)\right|}{\sum_{i \in I} (1+\lambda)^{i+1} \left|Supp(D_i) \cap h^{-1}(y)\right|} && (3.5) \\
&\geq \frac{(1-\varepsilon) p_L |S|}{(1 + \frac{6I}{\varepsilon}) \sum_{i \in I_{big}} (1+\lambda)^{i+1} |Supp(D_i)|} && (3.6) \\
&\geq \frac{\varepsilon(1-\varepsilon)}{\varepsilon + 6I} \cdot \frac{p_L \cdot \kappa}{p_H (1+\lambda)} && (3.7)
\end{aligned}$$

Where Equation (3.5) is because for $\rho \in S$, p_L is a lower bound on $\Pr_{d \leftarrow D}[d = \rho]$ and for $\rho \in Supp(D_i)$, $\Pr_{d \leftarrow D}[d = \rho] \leq (1+\lambda)^{-i+1}$. Equation (3.6) is true by applying Equation (3.3) and Equation (3.4), and finally Equation (3.7) is correct due to Equation (3.2). The claim is completed by noting that since $0 < \varepsilon \leq 1/2$:

$$\begin{aligned}
\frac{p_L \kappa}{p_H} \cdot \frac{\varepsilon(1-\varepsilon)}{(1+\lambda)(\varepsilon + 6I)} &\geq \frac{p_L \kappa}{p_H} \cdot \frac{\varepsilon}{2(1+\lambda)(\varepsilon + 6I)} \\
&\geq \frac{p_L \kappa}{p_H} \cdot \frac{\varepsilon}{40I} \\
&\geq \frac{p_L \kappa}{p_H} \cdot \frac{\varepsilon}{40m^2} \\
&\geq \frac{\kappa}{\mu} \cdot \frac{\varepsilon}{40m^2}
\end{aligned}$$

□

Now we can give the final claim to prove that H'_3 is close to H'_4:

Claim 3.9. *The statistical distance between H'_3 and H'_4 is at most ε.*

Proof. The inputs (h, y) to $HSim$ are drawn from the joint distribution $(\mathcal{H}, \mathcal{H}(D_S))$. By Claim 3.8, since:

$$\begin{aligned}
k &\leq \log N + \log_2\left(\frac{\varepsilon^3}{8}\right) \\
&\leq H_\infty(Z) + \log_2\left(\frac{\varepsilon^3}{8}\right) \\
&\leq \log|Supp(Z)| + \log_2\left(\frac{\varepsilon^3}{8}\right) \\
&= \log|S| + \log_2\left(\frac{\varepsilon^3}{8}\right)
\end{aligned}$$

With probability $1 - \frac{\varepsilon}{2}$ over the choice of h, no matter which y was given:

$$\Pr_{d \leftarrow D}[d \in S | h(d) = y] \geq \frac{\kappa}{\mu} \cdot \frac{\varepsilon}{40m^2}$$

Histogram Generator $Hist$

Parameters: Partition size $I \in [2^m]$, Error bound η, Cut-off σ.

Oracle Access: A sampling oracle Samp_D^δ and access to $\mathsf{Subset}_{D,\mathbb{D}}^\delta$

1. Set $J = \frac{m}{\eta^3}$, for each $i \in [I]$, $s_i = 0$
2. For $j = 1$ to J:
   (a) Sample $\rho \leftarrow \mathsf{Samp}_D^\delta$. Let $i' = \mathsf{Subset}_{D,\mathbb{D}}^\delta(\rho)$.
      i. Increase sum $s_{i'}$: $s_{i'} \leftarrow s_{i'} + 1$
3. For each $i \in [I]$:
   (a) Let $\tilde{p}_i = \frac{s_i}{J}$.
   (b) If $\tilde{p}_i < \sigma$: $p_i \leftarrow 0$. Otherwise set $p_i = \tilde{p}_i$
4. Return $(p_1, \ldots, p_I)$

Figure 3: (η, σ)-Histogram Generation Algorithm

Conditioned on this event occurring, taking $q = \frac{m}{\varepsilon} \cdot \left(\frac{40m^2}{\varepsilon} \cdot \frac{\mu}{\kappa}\right)$ samples from $D_{|h(D)=y}$, the probability that none of the elements sampled land inside of S is at most

$$\left(1 - \frac{m}{\varepsilon q}\right)^q \sim e^{-\frac{m}{\varepsilon}} < \frac{\varepsilon}{2}$$

Thus, with probability at least $1 - \varepsilon$ the algorithm samples an element in S. Since each sample is from the distribution D conditioned on $h(D) = y$, conditioned on landing in S, the element found is drawn from D conditioned on $D \in S$ and $h(D) = y$, which is identical to D_S conditioned on $h(D_S) = y$. Thus, all-in-all, when given inputs (h, y) drawn from $(\mathcal{H}, \mathcal{H}(D_S))$, the distribution output by $HSim$ is ε-close to sampling from D_S conditioned on $h(D_S) = y$. By an identical proof to that of Claim 3.4, this process ends with a sample randomly chosen from D_S □

3.3 Approximate Histograms

We are given a distribution D and a partitioning of D into I different subsets, $\mathbb{D}$. Our goal is to return a histogram showing the weight of each subset in $\mathbb{D}$ with regards to D. The tools at our disposal are oracle access to random samples from D and access to $\mathsf{Subset}_{D,\mathbb{D}}$ which when given an element $\rho \leftarrow D$ returns the partition index from $\mathbb{D}$ to which ρ belongs. To compute such a histogram we will take many samples from D, and test to which partition they belong, counting how many belonged to each subset. If we take enough samples, and a partition has enough weight with respect to D, then the ratio of samples taken to the number of samples in D_i is a good approximation of the weight of of subset D_i by D.

Lemma 3.10 ((η, σ)-Histogram Generator)**.** *Let D be a distribution over $\{0,1\}^m$ and $\mathbb{D} = \{D_i\}_{i=1}^I$ be a partitioning of D into I subsets where $I \in [2^m]$. Then given parameters $0 < \eta, \sigma \leq 1$ and oracle access to Samp_D^δ and $\mathsf{Subset}_{D,\mathbb{D}}^\delta$, the output of $Hist$, as described in Figure 3, is an (η, σ)-Histogram (see Definition 2.36) with regards to D and $\mathbb{D}$ except with probability $2^{-O(\frac{m}{\eta})} + \frac{2m\delta}{\eta^3}$. Moreover, $Hist$ runs in time $O\left(\frac{mI}{\eta^3}\right)$, and makes $O\left(\frac{m}{\eta^3}\right)$ oracle calls.*

Proof. We begin by looking at the running time and number of oracle calls. The algorithm makes J different iterations. In each iteration it makes one query to each of its oracles and changes a simple counter. The algorithm then must generate the output, which is mI bits. Therefore the running time is $O(mI + J) =$

$O\left(\frac{mI}{\eta^3}\right)$ and the number of oracle calls is $O(J) = O\left(\frac{m}{\eta^3}\right)$. In Proposition 3.11 we show that when given oracle access to ideal oracles, the probability that the algorithm fails to output a (η, σ)-histogram is $2^{-O(\frac{m}{\eta})}$.

Consider if the algorithm is given access to Samp^0_D and $\mathsf{Subset}^\delta_{D,\mathbb{D}}$. Since all inputs to $\mathsf{Subset}^\delta_{D,\mathbb{D}}$ are taken directly from the perfect sampling oracle, it will, for every one of the $\frac{m}{\eta^3}$ queries it receives, fail with probability at most δ. Using the union bound, with probability $\frac{m\delta}{\eta^3}$ it will not fail on any of the queries. Putting this together with the probability of failure in Proposition 3.11, in this experiment the algorithm fails with probability at most $2^{-O(\frac{m}{\eta})} + \frac{m\delta}{\eta^3}$.

Next, consider moving from access to Samp^0_D and $\mathsf{Subset}^\delta_{D,\mathbb{D}}$ to access to Samp^δ_D and $\mathsf{Subset}^\delta_{D,\mathbb{D}}$, as in the lemma statement. By adaptive sampling Corollary 2.18, the statistical distance between the outputs given access to Samp^0_D relative to Samp^δ_D is at most δ multiplied by the number of queries. Since the event that a correct histogram is output happens with probability $1 - (2^{-O(\frac{m}{\eta})} + \frac{m\delta}{\eta^3})$, in this final experiment it must also happen with probability at least $1 - (2^{-O(\frac{m}{\eta})} + \frac{2m\delta}{\eta^3})$. □

Proposition 3.11. *Let D be a distribution and $\mathbb{D} = \{D_i\}_{i=1}^I$ be a partitioning of D into I subsets where $I \in [2^m]$. Then given parameters $0 < \eta, \sigma \leq 1$ and oracle access to Samp^0_D and $\mathsf{Subset}^0_{D,\mathbb{D}}$, the output of $Hist$ is an (η, σ)-Histogram with regards to D and $\mathbb{D}$ except with probability $2^{-O(\frac{m}{\eta})}$.*

Proof. First, we note that $\mathsf{Subset}^0_{D,\mathbb{D}}$ is only guaranteed to work if it receives inputs from the correct distribution, namely inputs drawn from D. Since all queries to $\mathsf{Subset}^0_{D,\mathbb{D}}$ are samples taken directly from Samp^0_D which represents a random sample from D, the oracle does indeed receive inputs from the correct distribution. The oracle $\mathsf{Subset}^0_{D,\mathbb{D}}$ is ideal, and so never fails to classify elements in their respective subsets. Let $(p'_1, \ldots, p'_I)$ be a 0-histogram of $\mathbb{D}$ with respect to D (i.e. for every $i \in [I]$, $p'_i = \Pr_{\rho \leftarrow D}[\rho \in Supp(D_i)]$) and let $(\tilde{p}_1, \ldots, \tilde{p}_I)$ be the values defined in Stage 3.a of a random execution of $Hist$ as described in Figure 3. The algorithm takes J samples and for each sample tests to which subset it belongs. Since each sample is in subset i with probability p'_i, by the Chernoff bound (Lemma 2.2) by Stage 2.c:

$$\Pr\left[|\tilde{p}_i - p'_i| \geq \eta\right] \leq 2e^{-2J\eta^2}$$

We have now bounded the probability of receiving significant additive error on the probability approximation of a single subset. We can use the union bound in order to extend this to all subsets together, namely

$$\begin{aligned}\Pr\left[\exists i \in [I] : |\tilde{p}_i - p'_i| \geq \eta\right] &\leq 2 \cdot I \cdot e^{-2J\eta^2} \\ &= 2 \cdot I \cdot e^{-2\eta^2 \cdot \frac{m}{\eta^3}} \\ &\leq 2 \cdot 2^m \cdot e^{-2\frac{m}{\eta}} \\ &= 2^{-O(\frac{m}{\eta})}\end{aligned}$$

Conditioned on the event that all values of $\tilde{p}_i$ are in this range, we show that indeed the values p_i constitute an (η, σ)-Histogram of $\mathbb{D}$ (Definition 2.36). That is, that:

1. $|p_i - \Pr_{\rho \leftarrow D}[\rho \in Supp(D_i)]| \leq \sigma + \eta$. Moreover, if $p_i \neq 0$ then $|p_i - \Pr_{\rho \leftarrow D}[\rho \in Supp(D_i)]| \leq \eta$
2. $p_i \notin (0, \sigma)$

We begin by proving Item 1, conditioned on the event that for each i, $\tilde{p}_i$ is within additive error from p'_i as analyzed above. By the triangle inequality:

$$|p'_i - p_i| \leq |p'_i - \tilde{p}_i| + |\tilde{p}_i - p_i| \leq \eta + |\tilde{p}_i - p_i|$$

In Stage 3.b, of the algorithm, each value p_i is set to either 0 or left as $\tilde{p}_i$, depending on if $\tilde{p}_i$ is smaller than σ or not. This means that:

1. If $\tilde{p}_i \leq \sigma$, then $p_i = 0$ and so $\tilde{p}_i - p_i \leq \sigma$ which gives $|p'_i - p_i| \leq \eta + \sigma$
2. Otherwise $\tilde{p}_i > \sigma$ and then $p_i = \tilde{p}_i \neq 0$, and $|p'_i - p_i| \leq \eta$

Proving Item 1. Item 2 follows by definition - each value of p_i is either set to 0 or is greater than σ. □

3.4 Sampling Via Subsets

Let D be a distribution over $\{0,1\}^m$ and $\mathbb{D} = \{D_i\}_{i=1}^I$ be a partitioning of D into I parts. Our goal is for a prover to help a verifier sample an element from D and know also to which partition the element belongs. Suppose that the prover is honest, and that it is possible given i and $p \leq \Pr[D \in Supp(D_i)]$ it is possible for the verifier to lower-bound the min-entropy of D_i using oracle access to a function f (specifically $\log f(i,p) \leq H_\infty(D_i)$).

The protocol proceeds as follows - the prover creates a histogram $(p_1, \ldots, p_I)$ with respect to $\mathbb{D}$ and D, and chooses a partition based on histogram values- partition i is chosen with probability $\frac{p_i}{\sum_j p_j}$. The prover then sends to the verifier the partition index i and p, a claimed probability of landing in partition D_i. The verifier then uses these values, and access to f in order to find a lower-bound on the min-entropy of D_i. Now, the prover can help the verifier sample from D_i using the prover-efficient sampling protocol described in Section 3.2. Since the histogram is a representation of the probability of landing in each subset, each value of i is chosen with roughly the probability that D lands in subset D_i. The sampling protocol samples close to the distribution D_i, and so altogether the output distribution is close to that of a random sample from D along with its subset.

We now turn towards soundness. Suppose a malicious prover has for each i a set $S_i \subseteq \{0,1\}^m$ of elements for which it "wins" if the output of the protocol is (i,ρ) where $\rho \in S_i$. We show that it is unlikely that the prover will win. In the protocol, the prover sends some subset index i^* and probability p, the verifier uses f to calculate some claimed lower bound $N = f(i^*,p)$ and the two parties execute the sampling protocol. By the soundness property of the sampling protocol, if $|S_{i^*}|$ is much smaller than N, then the prover will likely fail. Note that the claim N set by the oracle f which the verifier has access to, and so in order to have a meaningful soundness property, it should be that the oracle is such that in the case of soundness it does not return very small values.

Lemma 3.12 (Properties of the "Sampling Via Subsets" protocol, Figure 4)**.** *Let $m \in \mathbb{N}$. Given parameters $I \in [2^m]$ and $0 < \varepsilon < 1$ the Sampling via Subsets protocol in Figure 4 is a public-coin interactive protocol with the following properties:*

Verifier Efficiency: Running time $O(m)$, Randomness $O(m)$. Oracle calls: one to f.

Prover Efficiency: Running time $O\left(\frac{mI^4}{\varepsilon^3}\right)$, Oracle calls: $O\left(\frac{mI^3}{\varepsilon^3}\right)$ to $\mathsf{Samp}_S^{\delta_1}$ and $\mathsf{Subset}_{D,\mathbb{D}}^{\delta_1}$ and one to $\mathsf{HashInv}_{D_i}^{\delta_2}$ for some i.

Number Of Messages: 3 Starting with a prover message.

Completeness: Let D be a distribution over $\{0,1\}^m$ and let $\mathbb{D} = \{D_i\}_{i=1}^I$ be a partitioning of $Supp(D)$ into I parts, and let $\widetilde{D} \equiv \left(D, \mathsf{Subset}_{D,\mathbb{D}}^0(D)\right)$ be the joint distribution of taking a sample from D along with the subset to which it belongs. Suppose the prover is given oracle access to $\mathsf{Samp}_D^{\delta_1}$, $\mathsf{Subset}_{D,\mathbb{D}}^{\delta_1}$ and for every $i \in [I]$, $\mathsf{HashInv}_{D_i}^{\delta_2}$ and both parties are given access to f such that for every i and p such that $0 \leq p \leq \Pr[D \in Supp(D_i)]$, $f(i,p) \leq H_\infty(D_i)$. Then letting Q be the output distribution of the protocol:

$$\Delta\left(Q, \widetilde{D}\right) \leq 2\varepsilon + O\left(\left(\frac{I}{\varepsilon}\right)^3 m\delta_1 + \delta_2\right)$$

Soundness: Let $f : [I] \times (0,1] \to \{0,1\}^m$ be a function given as oracle access to both parties and let $S_1, \ldots, S_I \subseteq \{0,1\}^m$ be sets. Then for every $j \in [I]$, and every cheating prover P^* the probability that the verifier outputs (ρ, i) such that $i = j$ and $\rho \in S_j$ when interacting with P^* is at most $\frac{8}{\varepsilon^3} \cdot \frac{|S_j|}{\min_{p' > \frac{\varepsilon}{6I}} f(j,p')}$.

Sampling Via Subsets

Parameters: Partition count $I \in [2^m]$, Sampling error $0 < \varepsilon < 1$. Set $\eta \triangleq \frac{\varepsilon}{6I}$.

Prover Oracle Access: $\mathsf{Samp}_D^{\delta_1}$, $\mathsf{Subset}_{D,\mathbb{D}}^{\delta_1}$, and for every $i \in [I]$, $\mathsf{HashInv}_{D_i}^{\delta_2}$.

Joint Oracle Access: $f : [I] \times (0,1] \to \{0,1\}^m$

Histogram Stage:

1. Prover:
   (a) Generate $(p_1, \ldots, p_I)$, a histogram using the histogram generation algorithm described in Lemma 3.10 with parameters $\eta_{hist} = \eta$ and $\sigma = 2\eta$ and oracle access to $\mathsf{Samp}_S^{\delta_1}$ and $\mathsf{Subset}_{D,\mathbb{D}}^{\delta_1}$. Choose a subset i by the distribution defined as follows

$$\Pr\left[\text{Choose subset } i\right] = \frac{p_i}{\sum_{i' \in [I]} p_{i'}}$$

   (b) Send i and p_i to the verifier.
2. Verifier:
   (a) Receive i and p_i. If $p_i < 2\eta$, reject.

Sampling Stage:

1. Set $N = f(i, p - \eta)$.
2. The parties take part in the prover-efficient sampling protocol (Lemma 3.1) to sample from D_i, claiming that D_i has entropy at least N, with parameter ε, where the prover is given oracle access to $\mathsf{HashInv}_{D_i}^{\delta_2}$.
3. The sampling protocol ends with both parties outputting some $\rho \in \{0,1\}^m$
4. Output ρ and i.

Figure 4: Sampling Via Subsets Protocol

3.4.1 Efficiency

Verifier Efficiency: In the histogram stage the verifier receives an index and a probability which has precision of m bits. It does a simple test on it. In the sampling stage the verifier simply runs the verifier of the sampling protocol, which by Lemma 3.1 takes time and randomness $O(m)$. Thus the verifier runs in total time $O(m)$ and randomness $O(m)$.

Prover Efficiency: The honest prover begins by calculating a histogram of the subsets, which by Lemma 3.10 takes $O\left(\frac{mI}{\eta^3}\right) = O\left(\frac{mI^4}{\varepsilon^3}\right)$ time and makes $O\left(\frac{mI^3}{\varepsilon^3}\right)$ oracle calls to Samp and Subset. It then samples i, which takes $O(m \log I)$ time, and runs the prover of the sampling protocol, which by Lemma 3.1 runs in time $O(m)$, making one oracle call to $\mathsf{HashInv}$. All the honest prover running time is $O\left(\frac{mI^4}{\varepsilon^3}\right)$ and it makes $O\left(\frac{mI^3}{\varepsilon^3}\right)$ calls to Samp and Subset and one to $\mathsf{HashInv}$.

Message Number: The histogram stage has the prover sending the verifier a single message. The sampling stage begins with a verifier message and has 2 messages in total. Therefore there are 3 messages sent.

3.4.2 Completeness

We define hybrids each of which represents the distribution of the output string ρ, and the claimed subset i in different experiments:

1. H_1: The output distribution of the protocol as described in Figure 4.
2. H_2: The histogram stage is as in H_1, but if the prover generated a correct histogram, then the sampling stage is replaced by the verifier taking a sample from the distribution $\rho \leftarrow D_i$, rather than running the sampling protocol (where i is the index chosen in the histogram stage).
3. H_3: The histogram stage is replaced by the prover generating a 0-histogram of D w.r.t. $\mathbb{D}$ and the verifier does not test that the value it receives is at least than 2η. The sampling stage is as in H_2.
4. H_4: The ideal distribution: $\left(D, \mathsf{Subset}^0_{D,\mathbb{D}}(D)\right)$, the joint distribution of a uniform sample from D, and the subset to which it belongs.

We first show that H_1 is close to H_2. The only difference between the two hybrids is that the sampling stage is replaced with the verifier taking a uniform sample of the required partition. The fact that the hybrids are close is due to the completeness property of the sampling protocol:

Proposition 3.13. *The statistical distance between H_1 and H_2 is at most $\frac{\varepsilon}{2} + \delta_2$.*

Proof. If $p < 2\eta$, then the verifier rejects immediately in both hybrids. If $p \geq 2\eta$, and the histogram was generated incorrectly, then both hybrids are identical. This leaves only the case that $p \geq 2\eta$ and the histogram was generated correctly. In this case, since p is the value output by a $(\eta, 2\eta)$-histogram, $p - \eta \leq Pr_{d \leftarrow D}[d \in Supp(D_i)]$. By assumption: $\log N = \log f(i, p - \eta) \leq H_\infty(D_i)$. Therefore, by the correctness of the sampling protocol (Lemma 3.1), the output of the sampling protocol has statistical distance $\frac{\varepsilon}{2} + \delta_2$ from sampling from D_i. □

Next, we have shown that the sampling is done correctly, we show that working with an $(\eta, 2\eta)$-histogram does not skew the choice of subset by much, since it is quite representative of the distribution.

Proposition 3.14. *The statistical distance between hybrids H_2 and H_3 is at most $\frac{3}{2}\varepsilon + \frac{2m\delta_1}{\eta^3}$.*

Proof. The difference between the hybrids is that the histogram in H_2 is sampled from $Hist$, whereas in H_3 it is sampled from a 0-histogram. This affects the choice of subset index i. In H_2 the prover tries to generate a $(\eta, 2\eta)$-histogram with respect to $\mathbb{D}$ which by Lemma 3.10 succeeds with probability $1 - (2^{-O(\frac{m}{\eta})} + \frac{2m\delta_1}{\eta^3})$.

The distribution of the subset index i chosen relative a $(\eta, 2\eta)$-histogram is close to the choice relative to a 0-histogram. Details follow: Let $(p_1, \ldots, p_I)$ be a $(\eta, 2\eta)$-histogram and $(p'_1, \ldots p'_I)$ be a and 0-histogram, both with respect to D and partitioning $\mathbb{D}$. Let W be the distribution where index i is chosen with probability $\frac{p_i}{\sum_{j\in[I]} p_j}$, and let W' be the distribution where index i is chosen with probability $\frac{p'_i}{\sum_{j\in[I]} p'_j}$. Note that $\Pr[W' = i] = \Pr_{\rho\leftarrow D}[\rho \in D_i]$. We wish to use Claim 2.16 with its ε parameter set to $3\eta = \frac{\varepsilon}{2I} < \frac{1}{2I}$ to show that W is $6I\eta$-close to W'. Since $6I\eta = \varepsilon$ this will mean they are ε-close.

In order to use Claim 2.16 in this way we must show that the $(\eta, 2\eta)$-histogram is not all zeros, and that for all $i \in [I]$, $\max\{0, p'_i - 3\eta\} \leq p_i \leq p'_i + 3\eta$. The fact that for all $i \in [I]$, $\max\{0, p'_i - 3\eta\} \leq p_i \leq p'_i + 3\eta$ follows from the definition of a $(\eta, 2\eta)$-histogram. The histogram will not be set to all zeros, due to the setting of $\eta = \frac{\varepsilon}{6I}$. For any $(\eta, 2\eta)$-histogram of $\mathbb{D}$ and D, and any index j, if $p_j = 0$ this implies that $\Pr[W' = i] \leq 3\eta < \frac{1}{2I}$. Therefore, in any $(\eta, 2\eta)$-histogram any subset which has weight at least $\frac{1}{I}$ will not have its histogram value set to zero. By an averaging argument, there must be some subset i such that $\frac{1}{I} \leq \Pr_{\rho\leftarrow D}[\rho \in D_i]$, and so there must be some subset with non-zero value in the histogram. Therefore Claim 2.16 shows that the choice of the subset index in a 0-histogram is at distance at most $6I\eta = \varepsilon$ from the choice of index using a $(\eta, 2\eta)$-histogram.

We have yet to show that the verifier will not reject the value set by the prover in H_2. Conditioned on the histogram being generated correctly, implies that $p_i \geq 2\eta$, and so the verifier's test will always pass. In H_3 the verifier never rejects, and so conditioned on the histogram being generated correctly the verifier's action will be identical.

To recap, the prover in H_2 generates a $(\eta, 2\eta)$-histogram with probability $1-(2^{-O(\frac{m}{\eta})}+\frac{2m\delta_1}{\eta^3})$. Conditioned on that event, choosing a subset based on this histogram is ε-close to the choice of subset in H_3. Putting this all together, the hybrids are of statistical distance at most $\varepsilon + 2^{-O(\frac{m}{\eta})} + \frac{2m\delta_1}{\eta^3}$. Since $\eta = \frac{\varepsilon}{6I}$ we have that $2^{-O(\frac{m}{\eta})} \ll \frac{\varepsilon}{2}$ and so this expression is bounded by $\frac{3}{2}\varepsilon + \frac{2m\delta_1}{\eta^3}$. □

All that we have left to show is that H_3, which uses an ideal histogram and sampling, is identical to sampling from D and giving its subset:

Proposition 3.15. *The distributions represented by hybrids H_3 and H_4 are identical.*

Proof. In H_3 the prover generates a 0-Histogram. The prover will therefore at the histogram stage, choose cluster i with probability $\Pr_{\rho\leftarrow D}[\rho \in D_i]$. In the sampling stage, the verifier samples uniformly from D_i. Let $\rho \in \{0,1\}^m$ and j be such that $\rho \in D_j$ then recalling that ρ cannot belong to any other subset:

$$\begin{aligned}\Pr[D = \rho] &= \Pr[D \in D_j] \cdot \Pr[D = \rho | D \in D_j] \\ &= p_j \cdot \Pr[D = \rho | D \in D_j] \\ &= \Pr[\text{prover chooses subset } j] \cdot \Pr[\text{verifier samples } \rho \text{ from } S_j] \\ &= \Pr[\text{verifier returns } \rho]\end{aligned}$$

Therefore ρ is distributed according to sampling from D. The protocol always outputs the correct subset and so the output distribution in H_3 is identical to H_4. □

We can now put all of the previous claims together to get:

Claim 3.16. *The statistical distance between the distributions H_1 and H_4 is bounded from above by $2\varepsilon + O\left(\left(\frac{I}{\varepsilon}\right)^3 m\delta_1 + \delta_2\right)$.*

Proof. Using Proposition 3.13, Proposition 3.14 and Proposition 3.15 and by the triangle inequality for

statistical distance:

$$\begin{aligned}\Delta(H_1, H_4) &\leq \Delta(H_1, H_2) + \Delta(H_2, H_3) + \Delta(H_3, H_4) \\ &\leq \frac{\varepsilon}{2} + \delta_2 + \frac{3}{2}\varepsilon + \frac{2m\delta_1}{\eta^3} \\ &\leq 2\varepsilon + \frac{2m\delta_1}{\eta^3} + \delta_2 \\ &= 2\varepsilon + O\left(\left(\frac{I}{\varepsilon}\right)^3 m\delta_1 + \delta_2\right)\end{aligned}$$

□

3.4.3 Soundness

Let $f : [I] \times (0,1] \to \{0,1\}^m$ be a function and $S_1, ..., S_I \subseteq \{0,1\}^m$ be sets such that, for all j and $p' > \eta$, $|S_j| < f(j, p')$. Let $i \in [I]$ be any subset which the cheating prover chooses, and let p be the value the prover sends to the verifier. If the value p is smaller than 2η then the verifier rejects immediately. Therefore in order for the prover to cause the verifier to sample in S_i the value p must be greater or equal to 2η. If this is indeed the case, the size claimed by the prover in the sampling stage is $N = f(j, p - \eta)$. By Lemma 3.1 the probability that the prover manages to cause the verifier to end up with an element in S_i is at most

$$\frac{8}{\varepsilon^3} \cdot \frac{|S_i|}{N} = \frac{8}{\varepsilon^3} \cdot \frac{|S_i|}{f(i, p-\eta)} \leq \frac{8}{\varepsilon^3} \cdot \frac{|S_i|}{\min_{p' > \eta} f(i, p')} = \frac{8}{\varepsilon^3} \cdot \frac{|S_i|}{\min_{p' > \frac{\varepsilon}{6I}} f(i, p')}$$

3.4.4 A Variation on the Protocol

The protocol as described in Figure 4 has access to $\mathsf{HashInv}$ oracles for each subset of D. Looking ahead, we will not be able to supply the honest prover with these oracles, but rather one global $\mathsf{HashInv}$ oracle for D in its entirety. Additionally the protocol requires that the parties know how to reasonably lower-bound the min-entropy of every (sufficiently large) partition, a requirement which we wish to relax.

Fortunately, using Corollary 3.5, we can deal with both of these variations, yielding the following corollary to Lemma 3.12:

Corollary 3.17 ("Sampling Via Subsets" protocol with limited oracle access)**.** *There exists a public-coin interactive protocol for which is identical to Lemma 3.12, except:*

- *The honest prover's oracles are replaced to* $\mathsf{HashInv}^{\delta}_{D}$ *and* $\mathsf{Subset}^{\delta}_{D,\mathbb{D}}$*. The prover additionally receives input* μ *such that for every* $i \in [I]$*:*

$$\mu \geq \frac{\max_{\rho \in Supp(D_i)} Pr[D = \rho]}{\min_{\rho \in Supp(D_i)} Pr[D = \rho]}$$

- *For every* $i \in [I]$ *there exists a distribution* Z_i *over* $Supp(D_i)$ *such that* $\Delta(D_i, Z_i) \leq \lambda$*.*
- *The function* f *is changed such that for every* $i \in [I]$ *and* $0 < p \leq \Pr[D \in Supp(D_i)]$*,* $\log f(i,p) \leq H_\infty(Z_i)$*.*

In the new protocol the completeness parameter is changed to $\lambda + 3\varepsilon + O\left(\left(\frac{mI\mu}{\varepsilon}\right)^4 \cdot \delta\right)$*. The honest prover runs in time* $O\left(\frac{\mu m^3 I^4}{\varepsilon^3}\right)$ *and makes* $O\left(\frac{\mu m^3 I^4}{\varepsilon^3}\right)$ *calls to its oracles. The remaining properties are identical to those specified in Lemma 3.12.*

Proof. In the histogram stage, the $\mathsf{Samp}_D^{\delta_1}$ used to construct the histogram is replaced with $\mathsf{HashInv}_D^{\delta}(0, \perp, \perp)$, and $\mathsf{Subset}_{D,\mathbb{D}}^{\delta_1}$ is replaced with $\mathsf{Subset}_{D,\mathbb{D}}^{\delta}$. In the sampling stage, the sampling protocol used is replaced from the one described in Lemma 3.12 to that of Corollary 3.5, where $\mathsf{InSet}_{D,S}^{\delta}$ is implemented using $\mathsf{Subset}_{D,\mathbb{D}}(i, \cdot)$ where i is the index chosen in the histogram stage. In order to hep the verifier sample from D_i, this new prover expects to receive inputs κ, μ along with oracle access to $\mathsf{HashInv}_D$ and $\mathsf{InSet}_{D_i,Supp(D_i)}$. Recall that by the end of the histogram stage both parties have a subset index $i \in [I]$ and a value p_i. The value κ is set to $\eta = \frac{\varepsilon}{6I}$ and μ will be given as is.

Let H_1' represent the output of this protocol, and H_2' be H_2 as in the proof of completeness (Section 3.4.2) except emulating H_1' rather than H_1 if the histogram is incorrect. We wish to show that H_1' is close to H_2'. Mimicking the proof of Proposition 3.13, we note that if $p_i < 2\eta$, then the verifier rejects immediately in both hybrids and that if $p_i \geq 2\eta$, but the histogram was generated incorrectly, then both hybrids are identical. If the histogram was generated correctly, then $\kappa = \eta \leq p_i - \eta \leq \Pr[D \in Supp(D_i)]$ and by assumption $\mu \geq \frac{\max_{\rho \in Supp(D_i)} Pr[D=\rho]}{\min_{\rho \in Supp(D_i)} Pr[D=\rho]}$. Therefore, by Corollary 3.5, the output of the sampling protocol has statistical distance $\lambda + \frac{3}{2}\varepsilon + \frac{80m^3\mu}{\kappa^2\varepsilon^2} \cdot \delta$ from randomly sampling from D_i. Therefore the statistical distance between H_1' and H_2' is at most $\lambda + \frac{3}{2}\varepsilon + \frac{80m^3\mu}{\kappa^2\varepsilon^2} \cdot \delta$

The proof that H_2' and H_3 (as defined in Section 3.4.2) are of statistical distance at most $\frac{3}{2}\varepsilon + \frac{432mI^3\delta}{\varepsilon^3}$ is identical to the proof of Proposition 3.14, which shows that H_2 and H_3 are close. Since H_3 is identical to $H_4 \equiv W$, the ideal distributions, using the triangle inequality for statistical distance we have that

$$\begin{aligned}
\Delta(H_1', W) &\leq \Delta(H_1', H_2') + \Delta(H_2', H_3) + \Delta(H_3, H_4) \\
&\leq \lambda + \frac{3}{2}\varepsilon + \frac{80m^3\mu}{\kappa^2\varepsilon^2} \cdot \delta + \frac{3}{2}\varepsilon + \frac{432mI^3\delta}{\varepsilon^3} \\
&= \lambda + 3\varepsilon + O\left(\left(\frac{mI\mu}{\varepsilon}\right)^4 \cdot \delta\right)
\end{aligned}$$

Turning to analysis of the efficiency of the prover, by Corollary 3.5, the sampling protocol requires $O\left(\frac{\mu m^3}{\varepsilon^2\kappa}\right)$steps and oracle calls. The histogram stage has not changed - generating the histogram takes $O\left(\frac{mI^4}{\varepsilon^3}\right)$ time and the prover makes $O\left(\frac{mI^3}{\varepsilon^3}\right)$. Thus together the prover runs in total time:

$$O\left(\frac{mI^4}{\varepsilon^3} + \frac{\mu m^3 I}{\varepsilon^3}\right) = O\left(\frac{\mu m^3 I^4}{\varepsilon^3}\right)$$

and makes the same number of calls to its oracles.

None of the changes made alter the number of messages and the soundness and verifier efficiency properties, and so they remain identical to those described in Lemma 3.12. □

3.5 Set Cardinality Approximation

In this section we show that inversion implies approximation of the number of elements in a set. Given a set $S \subseteq \{0,1\}^\ell$ and oracle access to $\mathsf{HashInv}_{U_S}$, we would like to construct an algorithm that approximates the size of S. This algorithm is similar to approximate counting using an $\mathcal{NP}$ oracle (see [Gol08, Theorem 6.27]) but in our case instead of the $\mathcal{NP}$ oracle, the algorithm has access to a "hash inversion" oracle, $\mathsf{HashInv}_{U_S}$.

Suppose we choose a random hash function h from a family of pairwise independent hash functions $\mathcal{H}$ mapping $\{0,1\}^\ell$ to $\{0,1\}^k$, and a random image $y \in \{0,1\}^k$. Due to the "almost-uniform cover" property of hash functions (Lemma 2.19), the number of elements in S that h maps to y is proportional to the ratio $\frac{|S|}{2^k}$, (except with probability that is itself proportional to $\frac{2^k}{|S|}$). This can be used to lower-bound the size of S - choose a random pair h, y and if $S \cap h^{-1}(y)$ is large, then it is likely that $|S| > 2^k$. The algorithm works in iterations on the size k, beginning by performing this test for $k = 0$, and if the test passes, it tries the next largest k. We can approximate the size of $|S|$ based on the value of k in which the algorithm fails. The

actual protocol makes a minor change to this recipe - the protocol will not choose a random element y, but will rather choose a random $x \in S$ and set $y = h(x)$. This is done because the prover has oracle access only to the function $\mathsf{HashInv}_{U_S}$, which expects to receive inputs (h, y) drawn from the distribution $(\mathcal{H}, \mathcal{H}(U_S))$. After setting y as described above, in order to use the same ideas as before, we can use 3-wise independence instead of pairwise. The hash function remains pairwise independent even when conditioned on $y = h(x)$.

Lemma 3.18. *((Amplified) Cardinality Approximation) Let $S \subseteq \{0,1\}^\ell$ be some set and $\mathcal{H}$ be a family of pairwise independent hash functions. Then for any parameters $0 < \nu \leq 1$, $0 < \varepsilon < \frac{1}{2}$ there exists randomized algorithm CardApprox such that:*

$$\Pr\left[CardApprox_{\nu,\varepsilon}^{\mathsf{HashInv}^{\delta}_{U_S}}\left(1^\ell\right) \notin (1 \pm \nu)|S|\right] \leq \varepsilon + O\left(\frac{\ell^4}{\nu^6}\log\frac{1}{\varepsilon}\cdot\delta\right)$$

Moreover, $CardApprox_\nu$ runs in time $O\left(\frac{\ell^4}{\nu^6}\log\frac{1}{\varepsilon}\right)$ and makes $O\left(\frac{\ell^4}{\nu^6}\log\frac{1}{\varepsilon}\right)$ queries to its oracle.

Proof. In Figure 5 we describe a slightly simpler algorithm which ignores the ε parameter. This algorithm, given access to an ideal inversion oracle, $\mathsf{HashInv}^0_{U_S}$, is analyzed in Proposition 3.19, and it is shown that it succeeds except with probability $\frac{1}{\ell}$. Keeping an ideal inversion oracle, we can reduce the error to ε by running it $\Theta\left(\log\frac{1}{\varepsilon}\right)$ times and returning the median of the values generated. This means that the running time and number of oracle calls are both $O\left(\frac{\ell^4}{\nu^6}\log\frac{1}{\varepsilon}\right)$. Using adaptive sampling (Corollary 2.18), moving from ideal $\mathsf{HashInv}^0_{U_S}$ oracle to $\mathsf{HashInv}^{\delta}_{U_S}$, the statistical distance at most $O\left(\frac{\ell^4}{\nu^6}\log\frac{1}{\varepsilon}\cdot\delta\right)$, and so the event that the value output is not in the correct range happens with probability at most:

$$\varepsilon + O\left(\frac{\ell^4}{\nu^6}\log\frac{1}{\varepsilon}\cdot\delta\right)$$

□

Proposition 3.19. *Let $S \subseteq \{0,1\}^\ell$ be some set and $\mathcal{H}$ be a family of 3-wise independent hash functions. Then for any parameter $0 < \nu \leq 1$ the algorithm $CardApprox_\nu$ as described in Figure 5 satisfies:*

$$\Pr\left[CardApprox_{\nu}^{\mathsf{HashInv}^{0}_{U_S}}\left(1^\ell\right) \in (1 \pm \nu)|S|\right] = 1 - \Theta\left(\frac{1}{\ell}\right)$$

Moreover, the running time and the number of queries to the inversion oracle of $CardApprox_\nu$ are both $O\left(\frac{\ell^4}{\nu^6}\right)$.

Proof. We have two cases to handle. If $|S| < q$ then we hope to sample every element in S in Stage 1, returning $|S|$, and if $|S| \geq q$ then we need the algorithm to stop at the right iteration. In the former case, where S is small, since $\mathsf{HashInv}_{U_S}(0, \perp, \perp)$ samples ideally from U_S, we can invoke the coupon collector theorem, Claim 2.37, so that with probability $1 - \frac{1}{q} = 1 - O(2^{-t})$ the algorithm will return all of the elements in S, and the algorithm will end in Stage 1.b returning $|S|$. We now turn to the latter case, in which S has at least q elements. Let $i = \lfloor \log_2 |S| \rfloor$. We begin by showing that the algorithm is unlikely to stop before iteration $i - t$.

Claim 3.20. *Let $i = \lfloor \log_2 |S| \rfloor$. The probability that $CardApprox_\nu$ terminates before iteration $i - t$ is $\Theta(\ell \cdot 2^{-t})$.*

Proof. Using Claim 2.37, the algorithm passes Stage 1 with probability $1 - \Theta(2^{-t})$. We now look at the probability that the algorithm stops at some iteration $k \in [i - t - 1]$. Since $i > k$ and $i \geq 1$,

$$\begin{aligned}\frac{|S| - 1}{2^k} &\geq \frac{2^i - 1}{2^{i-t-1}} \\ &= 2^{t+1} - 2^{t+1-i} \\ &\geq 2^{t+1} - 1\end{aligned}$$

Cardinality Approximator $CardApprox$

Input: Length parameter 1^{ℓ}

Oracle Access: $\mathsf{HashInv}^{\delta}_{U_S}$

Parameters: Approximation parameter ν.

1. Let $t \triangleq \log(\ell) + \max\{\log(\ell), -2\log(\nu)\}$ and $q \triangleq 2^t$
2. Invoke the oracle $\mathsf{HashInv}^{\delta}_{U_S}(0, \perp, \perp)$ q^3 times to receive samples $(\rho_1, \ldots, \rho_{q^3})$
   (a) Let T be the set of unique elements in $\{\rho_i\}_{i=1}^{q^3}$.
   (b) If $|T| < q^3$, then output $|T|$
3. Sample $x \leftarrow \mathsf{HashInv}^{\delta}_{U_S}(0, \perp, \perp)$.
4. For $k = 1$ to $\ell - t$:
   (a) Choose $h \leftarrow \mathcal{H}_{m,k}$ and $y = h(x)$.
   (b) Invoke $\mathsf{HashInv}^{\delta}_{U_S}(k, h, y)$ for q^3 times to receive samples $\left(\rho'_1, \ldots, \rho'_{q^3}\right)$
   (c) Let T' be the set of unique elements in $\{\rho'_i\}_{i=1}^{q^3}$
   (d) If $|T'| \geq q$ go to the next iteration.
   (e) Otherwise, invoke $\mathsf{HashInv}^{\delta}_{U_S}(k, h, y)$ for $m = (1 + (1+\nu) \cdot 2^t)^3$ times to receive $(\rho''_1, \ldots, \rho''_m)$
   (f) Return $1 + 2^t \cdot (|T''| - 1)$ where T'' is the set of unique elements in $\{\rho''_i\}_{i=1}^{m}$
5. Return 2^{ℓ}

Figure 5: Cardinality Approximator

Using Corollary 2.20 with parameters $\lambda = \frac{1}{2}$ and pairwise independence, for any $x \in S$

$$\begin{aligned}\Pr_{h \leftarrow \mathcal{H}_{\ell,k}}\left[|\{\rho \in S | h(\rho) = h(x)\}| \notin 1 + \left(\frac{1}{2}, \frac{3}{2}\right) \cdot \frac{|S|-1}{2^k}\right] &\leq 8 \cdot \frac{2^k}{|S|-1} \\ &\leq 8 \cdot 2^{-t}\end{aligned}$$

and so

$$\begin{aligned}\Pr_{h \leftarrow \mathcal{H}_{\ell,k}}\left[|\{\rho \in S | h(\rho) = h(x)\}| < 2^t\right] &\leq \Pr_{h \leftarrow \mathcal{H}_{\ell,k}}\left[|\{\rho \in S | h(\rho) = y\}| < 1 + \frac{1}{2} \cdot \frac{|S|-1}{2^k}\right] \\ &\leq \Pr_{h \leftarrow \mathcal{H}_{\ell,k}}\left[|\{\rho \in S | h(\rho) = y\}| \notin 1 + \left(\frac{1}{2}, \frac{3}{2}\right) \cdot \frac{|S|-1}{2^k}\right] \\ &\leq 8 \cdot 2^{-t}\end{aligned}$$

Therefore, with probability at least $8 \cdot 2^{-t}$, there are at least q elements in S which map to $y = h(x)$. The algorithm takes q^3 samples, attempting to sample at least q samples in the set $\{\rho \in S | h(\rho) = y\}$ by using HashInv. Note that the input to HashInv is distributed according to $(\mathcal{H}, \mathcal{H}(U_S))$ and so the sampling is done correctly. By Claim 2.37, given that the number of preimages of y in S is at least q after taking q^3 samples, the probability that the algorithm does not see q distinct elements is $\Theta\left(\frac{1}{q}\right) = \Theta(2^{-t})$. Putting together the probability that $|\{\rho \in S | h(\rho) = y\}| < q$ and the probability that the algorithm does not see 2^t distinct elements given that $|\{\rho \in S | h(\rho) = y\}| \geq q$, the probability that the algorithm stops at iteration k is $\Theta(2^{-t})$. Hence by using the union bound, the probability that the algorithm terminates in any of the $i-t-1$ stages is $\Theta((i-t)2^{-t})$. Since $i \leq \ell$, and including the addition of the probability of terminating before step 3 is upper bounded by $O(\ell \cdot 2^{-t})$. □

Assuming the algorithm has passed iteration $i-t-1$, we show that with high probability it will not reach iteration $i-t+2$.

Claim 3.21. *Let $i = \lfloor \log_2 |S| \rfloor$. If the algorithm reaches iteration $k = i-t+1$ then it will terminate with probability $1 - \Theta(2^{-t})$.*

Proof. Noting that

$$2^{t-2} \leq \frac{|S|-1}{2^k} = \frac{2^i - 1}{2^{i-t+1}} \leq 2^{t-1}$$

Since $t > 2$, we have that $1 + \frac{3}{2} \cdot 2^{t-1} \leq 2^t$. We again utilize Corollary 2.20 with $\lambda = \frac{1}{2}$, to get

$$\begin{aligned}\Pr_{h \leftarrow \mathcal{H}_{\ell,k}}\left[|\{\rho \in S | h(\rho) = h(x)\}| \notin 1 + \left(\frac{1}{2}, \frac{3}{2}\right) \cdot \frac{|S|-1}{2^k}\right] &\leq 8 \cdot \frac{2^k}{|S|-1} \\ &\leq 8 \cdot 2^{-t}\end{aligned}$$

and

$$\begin{aligned}\Pr_{h \leftarrow \mathcal{H}_{m,k}}\left[|\{\rho \in S | h(\rho) = y\}| \geq 2^t\right] &\leq \Pr_{h \leftarrow \mathcal{H}_{m,k}}\left[|\{\rho \in S | h(\rho) = y\}| > 1 + \frac{3}{2} \cdot \frac{|S|-1}{2^k}\right] \\ &\leq \Pr_{h \leftarrow \mathcal{H}_{\ell,k}}\left[|\{\rho \in S | h(\rho) = h(x)\}| \notin 1 + (0,2) \cdot \frac{|S|-1}{2^k}\right] \\ &\leq 8 \cdot 2^{-t}\end{aligned}$$

Given that $|\{\rho \in S | h(\rho) = y\}| < 2^t$, the algorithm will stop at this iteration since there are less than q distinct elements to sample. Therefore the probability that the algorithm will terminate in iteration $i-t+1$ is $1 - \Theta(2^{-t})$. □

Now that we have bounded the probability that the algorithm terminates either in stage $i-t$ or $i-t-1$ we need to show that the it approximates size of S to within the required range.

Claim 3.22. *Let $i = \lfloor \log_2 |S| \rfloor$. Given that $CardApprox_\nu$ stopped at iteration $i-t$ or $i-t+1$, the probability that it returns a value in the range $(1 \pm \nu)|S|$ is $1 - O\left(\nu^{-2} \cdot 2^{-t}\right)$.*

Proof. Since $k \in \{i-t, i-t+1\}$, $\frac{2^i-1}{2^{i-t+1}} \leq \frac{|S|-1}{2^k} \leq \frac{2^i-1}{2^{i-t}}$ and so $2^{t-2} \leq \frac{|S|-1}{2^k} \leq 2^t$. Once again utilizing Corollary 2.20 with $\lambda = \nu$:

$$\begin{aligned} \Pr_{h \leftarrow \mathcal{H}_{m,k}} \left[|\{r \in S | h(r) = y\}| \notin 1 + (1 \pm \nu) \cdot \frac{|S|-1}{2^k} \right] &< \frac{2^k}{\nu^2 |S|} \\ &\leq 4 \cdot \nu^{-2} \cdot 2^{-t} \end{aligned}$$

If $|\{r \in S | h(r) = y\}|$ is indeed in the range required, then $|\{r \in S | h(r) = y\}| < 1 + (1+\nu) \cdot 2^t$. Therefore, by the coupon collector problem, Claim 2.37, using $m = 1 + (1+\nu) \cdot 2^t$, the probability that the algorithm will not sample all of the elements of $S \cap h^{-1}(y)$ in Stage 3.*e* is $O\left(\frac{1}{m}\right) = \Theta\left(\nu^{-1} 2^{-t}\right)$. Putting this together with the probability that $|\{r \in S | h(r) = y\}|$ is not in the correct range, we have that conditioned on the algorithm stopping on iteration $i-t$ or $i-t+1$: With probability $1 - \Theta\left(\nu^{-2} \cdot 2^{-t}\right)$, $|T''| \in 1 + (1 \pm \nu) \cdot \frac{|S|-1}{2^k}$ and so:

$$(1-\nu)|S| \leq 1 + (1-\nu)(|S|-1) \leq 1 + 2^t(|T''|-1) \leq 1 + (1+\nu)(|S|-1) \leq (1+\nu)|S|$$

□

By taking Claims 3.20 to 3.22 together using the union bound, the probability that the algorithm fails to return a value in the correct range is $1 - \Theta\left(2^{-t}\left(\ell + \nu^{-2}\right)\right)$. Noting our choice of $t = \log(\ell) + \max\{\log(\ell), -2\log(\nu)\}$ the failure probability is $\Theta\left(\frac{1}{\ell}\right)$.

All that remains is to analyze the running time of $CardApprox_\nu$.

Claim 3.23. *The running time and the number of queries to the inversion oracle of $CardApprox_\nu$ are both $O\left(\frac{\ell^4}{\nu^6}\right)$.*

Proof. The running time is dominated by taking ℓ iterations. In each iteration the algorithm first uses the inversion oracles to sample 2^{3t} preimages and in the final iteration before it returns a value takes another $\left((1+\nu) \cdot 2^{t+1}\right)^3$ samples of preimages. Note that since $\nu \leq 1$, $\left((1+\nu) \cdot 2^{t+1}\right)^3 = O\left(2^{3t}\right)$. For each set of preimages sampled, the algorithm simply counts how many distinct elements there are, which can be done in polynomial time in the number of elements the algorithm must go over. Therefore both the running time of the algorithm and the number of queries send to the oracle is

$$O\left(\ell \cdot 2^{3t}\right) = O\left(\ell \cdot \left(\frac{\ell^3}{\nu^6}\right)\right) = O\left(\frac{\ell^4}{\nu^6}\right)$$

□

Together, Claims Claim 3.20, Claim 3.21 and Claim 3.22 show that the algorithm will with the required probability return a value in the requested bound, and Claim 3.23 shows that the algorithm is efficient depending on ν, together proving Proposition 3.19. □

4 Round-Preserving Emulation from Inversion

In this section, we show a public-coin emulation for private-coin proofs that preserves the running time of both parties and the number of rounds in the protocol. Our emulation is inspired by the celebrated result of Goldwasser and Sipser [GS86], based on the following idea: Rather than proving that a random interaction leads to the verifier accepting, the prover proves that there are many sets of coins that would lead to the verifier accepting if the verifier were to use them as its random input. We begin by giving some notation for this section:

Notation: For the entirety of this section we consider a language $\mathcal{L}$ that has an r-round interactive proof $\Pi = \langle P, V \rangle$ with ℓ bits of randomness, m-bit messages, completeness c, soundness error s, prover running time t_P and verifier running time t_V. For $x \in \{0,1\}^n$ and $\rho \in \{0,1\}^\ell$ we denote by $\Pi^i_x(\rho)$ the partial transcript of the protocol executed up to the i'th message on input x and with verifier randomness ρ, where for any x and ρ, $\Pi^0_x(\rho) = \emptyset$ is the empty transcript. Additionally, for $1 \leq i \leq r$, we denote by $t_{P,i}$ the running time of the prover in round i and $t_{V,i}$ the running time of the verifier in the same round. Note that $\sum_{i=1}^r t_{P,i} = t_P$ and $\sum_{i=1}^r t_{V,i} = t_V$.
Fix $x \in \mathcal{L}$ and a transcript prefix $\gamma = (\alpha_1, \beta_1, \ldots, \alpha_i, \beta_i)$. Recall Definitions 2.9 to 2.11:

- $Coins_x(\gamma)$: The set of verifier coins that are consistent with γ. We will sometimes overload the term and refer to $Coins_x(\gamma)$ as the uniform distribution over the set of consistent coins. That is, U_ℓ conditioned on: For every $j < i$, $V_x(\alpha_1, \beta_1, \ldots, \alpha_{j-1}, \beta_{j-1}; U_\ell) = \alpha_j$.

- $Acc_x(\gamma)$: The set of verifier coins that are consistent with γ and for which the prover has an accepting strategy. Let $Coins_x(\gamma)$ be as above, then:

$$Acc_x(\gamma) = \{\rho \in Coins_x(\gamma) \mid \exists \gamma' : \ \rho \in Coins_x(\gamma, \gamma') \text{ and } V(x, (\gamma, \gamma'); \rho) = 1\}$$

- $Msg_x(\gamma)$: The distribution of the next verifier message, consistent with γ. Let $Coins_x(\gamma)$ be as above, then $Msg_x(\gamma)$ is the distribution of the next message of V generated by sampling consistent coins: $V_x(\gamma; Coins_x(\gamma))$.

Overview: Round i of the protocol will begin with a partial transcript γ_{i-1} and a claimed lower bound N_{i-1} on $|Coins_x(\gamma_{i-1})|$, number of coins consistent with γ_{i-1}. Note that, unlike in Goldwasser and Sipser's original protocol, the bound is a claim about $|Coins_x(\gamma_{i-1})|$ rather than about the number of *accepting* and consistent coins, $|Acc_x(\gamma_{i-1})|$. Looking ahead, this is done since do not know how to use inversion for the honest prover to approximate the number of accepting coins, but only of consistent coins in general. The by-product of this is that in completeness the transcripts sampled by our emulation are close to random, rather than always accepting as in the original transformation. For soundness, the verifier will also need to check in the last round that the transcript and coins sampled in the final verifier message are accepting (recall that without loss of generality we assume that the last message in the private-coin protocol is the verifier revealing its randomness).

We begin by looking at completeness. The honest prover's goal is to help the verifier sample a next verifier message α_i which is consistent with γ_{i-1}. If the parties begin with a (reasonable) lower-bound on the entropy of the distribution of messages consistent with γ_{i-1} then the parties could run the sampling protocol described in Section 3.2. We would have liked to execute this for each round iteratively- in each round the prover helps the verifier sample a new message, and then the prover returns its answer, thus constructing a transcript. The issue with this approach is that even if for a given round we have a lower bound on the entropy of the distribution of messages consistent with the transcript, it is unclear how to find such a lower-bound for the next round, and convince the verifier of this bound.

We solve this problem similarly to Goldwasser and Sipser, by introducing the concept of clusters. Consider the following experiment: Suppose the honest prover can, given $x \in \mathcal{L}$ and partial transcript γ_{i-1}, for every message α which is consistent with γ_{i-1}, calculate the size of the support of $Coins_x(\gamma_{i-1}, \alpha)$. Then we can define the following partitioning of $Msg_x(\gamma_{i-1})$ into $\ell + 1$ parts: subset j will be all the messages α consistent with γ_{i-1} such that $2^{j-1} \leq |Coins_x(\gamma_{i-1}, \alpha)| < 2^j$. We call these subsets *Clusters*. Now we would like to sample a message along with its cluster index, by using the sampling via subsets protocol, where the subsets are the clusters. One of the requirements to use the sampling via subsets protocol, is that the parties need access to a function f that, given a cluster index and p, a lower-bound on the probability of landing in this cluster, gives a lower-bound on the entropy of $Msg_x(\gamma_{i-1})$ conditioned on landing in cluster i. Suppose the parties have a lower-bound N_{i-1} on $|Coins_x(\gamma_{i-1})|$, and suppose for every j, $Msg_x(\gamma)$ conditioned on landing in cluster j is flat (i.e. it is uniform over its support). By definition, if a message is in cluster j then there are at most 2^{j+1} coins that lead to this message. Then, if the probability of landing in cluster j is p,

there are at least $\frac{pN_{i-1}}{2^{j+1}}$ messages in this cluster. Since the message distribution conditioned on landing in cluster j is uniform over its support, this means that $\log \frac{2^{j+1}}{pN_{i-1}}$ is a lower-bound on the min-entropy of the message distribution conditioned on landing in the cluster.

Now that the parties can successfully use the sampling via subsets protocol in order to sample (α_i, j) a message and the index of the cluster to which it belongs, we can, from the cluster index, find a lower-bound on the number of coins consistent with α_i (namely 2^j), and feed this into the protocol iteratively. Looking ahead, we will not be able to work with clusters as described above, but rather the prover will have a partitioning of the message space representing an "approximate clustering" - where we know a bound on the number of coins consistent with each message in the approximate cluster.

For soundness, a round of the protocol described above begins with a partial transcript γ_{i-1}, and a claim N_{i-1}. Consider the ratio between the prover's claim and the number of accepting coins consistent with the current transcript, $\frac{N_{i-1}}{|Acc_x(\gamma_{i-1})|}$, which we call the "gap". We say that the malicious prover "wins" if it can decrease the gap significantly. That is, if the sampling via subsets protocol ends with values γ_i and N_i, then we want $\frac{N_i}{|Acc_x(\gamma_i)|}$ to not be significantly smaller than $\frac{N_{i-1}}{|Acc_x(\gamma_{i-1})|}$. To see why it is of interest to us to limit the prover's ability to shrink the gap, consider output values γ_i and N_i such that $N_i = Acc_x(\gamma_i)$. Now the malicious prover can, using the sampling via subsets protocol honestly, and make the verifier end up with a next message which is consistent with an accepting transcript. Since it can do this for every round it can generate a full accepting transcript and make the verifier accept. Luckily, we can bound the number of elements that do shrink the gap significantly. Recall that the protocol has soundness s. Thus, in order to cause the verifier to accept, the gap must go all the way down from $\frac{1}{s}$ to 1, and this together means that the verifier will reject with good probability.

Organization: We begin in Section 4.1 by showing that if it is possible to efficiently invert certain functions it is possible to implement the functionality of HashInv (see Section 3.1) for useful distributions. In Section 4.2 we show that HashInv can be used to partition the messages based on the number of coins that are consistent with them. Finally, in Section 4.3 we put all of these tools together with ones developed in Section 3 in order to emulate a protocol.

4.1 Implementing HashInv

In order to use the tools developed in Section 3 we need to give the prover access to an oracle HashInv defined in Section 3.1 as follows:

- $\mathsf{HashInv}^{\delta}_{D}$: Parametrized by a distribution D over $\{0,1\}^{\ell}$ and error $0 \leq \delta \leq 1$. Receives as input $k \in [\ell]$, $h \in \mathcal{H}_{\ell,k}$ and $y \in \{0,1\}^k$. The output is such that $\Delta\left(\mathsf{HashInv}^{\delta}_{D}(k,h,y), D_{h(D)=y}\right) \leq \delta$ when $h \leftarrow \mathcal{H}_{\ell,k}$ and $y \leftarrow h(D)$.

Specifically, we will be interested in using D as the distribution of either verifier coins or verifier messages in the protocol. Let $\langle P, V\rangle$ be a protocol for $\mathcal{L}$ in which the verifier uses ℓ bits of randomness and the verifier messages have length m. Thus, translating this to HashInv we get:

- $\mathsf{HashInv}^{\delta}_{Coins_x(\gamma)}$: Receives inputs $k \in [\ell]$ and $(h,y) \leftarrow (\mathcal{H}_{\ell,k}, \mathcal{H}_{\ell,k}(Coins_x(\gamma)))$ and returns an output ρ that is δ-close to $Coins_x(\gamma)$ conditioned on the hash of the coins equalling y.
- $\mathsf{HashInv}^{\delta}_{Msg_x(\gamma)}$: Receives as input $k \in [m]$, $(h,y) \leftarrow (\mathcal{H}_{m,k}, \mathcal{H}_{m,k}(Msg_x(\gamma)))$. The output is drawn from a distribution that is δ-close to $Msg_x(\gamma)$ conditioned on the message hashing to y.

We will not be able to implement the above functionalities for every transcript γ. Rather, we will only be able to do so for random γ.

Remark 4.1. To show HashInv for coins we only need to show it for transcripts that end in a verifier message, since for any γ ending in a verifier message and prover message β, $Coins_x(\gamma) \equiv Coins_x(\gamma, \beta)$. Similarly, when implementing HashInv for messages we only need to show how it can be done for transcripts that end

in a prover message. This is because when γ ends with a verifier message, $Msg_x(\gamma)$, the distribution of the next message consistent with γ, always returns the last message in γ.

Overview: We begin by showing in Claim 4.2 that for any efficiently sampleable distribution D, efficiently computable dependency function Dep and efficiently computable morphing function g, there exists a function f such that if f is invertible then it is possible to efficiently sample from $g(D)$ conditioned on $Dep(D) = d$. In Sections 4.1.1 and 4.1.2 we show how to use Claim 4.2 to implement $\mathsf{HashInv}$ in order to sample coins and messages that are consistent with a transcript and that hash to some value for both doubly-efficient proofs and for constant-round proofs.

Consider the simpler goal of, given a transcript $\gamma = (\alpha_1, \beta_1, \ldots, \alpha_i, \beta_i)$, sampling from $Coins_x(\gamma)$. We use Claim 4.2 as follows:

- Doubly-Efficient Proofs: For doubly-efficient proofs the task described above will be relatively easy. Since we can efficiently sample transcripts of the protocol, we can set D to be U_ℓ. $Samp$ can therefore be easily implemented. Dep on input ρ will output the transcript consistent with ρ, $\gamma = \Pi^i_x(\rho)$ up to round i. Dep is efficient because the protocol is doubly-efficient. g will be the identity function. Note that $g(D)$ conditioned on $Dep(D) = d$ is exactly the distribution $Coins_x(\gamma)$. Then Claim 4.2 implies that if $f = Dep(Samp(\cdot))$ is invertible then there exists an algorithm that can be used to sample from $Coins_x(\gamma)$.

- Constant-Round Proofs: For general constant-round proofs we have to take a different strategy since the protocol itself is inefficient. The structure of our construction is iterative. Assuming there exists an efficient sampler A^i such that for $\gamma = (\alpha_1, \beta_1, \ldots, \alpha_i)$, $A^i(\gamma) \equiv Coins_x(\gamma)$, we show a function f^{i+1} that uses A^i as a sub-procedure such that if f^{i+1} is invertible, then there is an efficient sampler A^{i+1} such that, given $\gamma' = (\alpha'_1, \beta'_1, \ldots, \alpha'_{i+1})$, samples from $Coins_x(\gamma')$.

  Suppose for specific i it were possible to, given $\gamma = (\alpha_1, \beta_1, \ldots, \alpha_i)$, sample from $Coins_x(\gamma)$ using an algorithm A^i. We show how to implement sampling of coins which will work with transcripts up to round $i+1$. The idea is that if we can sample coins that are consistent with $\gamma = (\alpha_1, \beta_1, \ldots, \alpha_i)$ then for a fixed β_i we can sample a the next verifier message α_{i+1} from the distribution of next verifier messages that are both consistent with γ and β_i. Inverting this sampler will allow us to sample coins consistent with transcripts of the form $(\alpha_1, \beta_1, \ldots, \alpha_{i+1})$. In more detail, let f^i be as follows: Take in as auxiliary inputs $\gamma = (\alpha_1, \beta_1, \ldots, \alpha_i)$ and β_i and regular input randomness r. Sample $\rho = A^i(\gamma; r)$ and output $\alpha_{i+1} = V(\gamma, \beta_i; r)$. Suppose there exists an inverter Inv for f^i. Now, given $\gamma = (\alpha_1, \beta_1, \ldots, \alpha_{i+1})$ we can sample coins consistent with γ as follows: Call Inv with auxiliary inputs $\gamma' = (\alpha_1, \beta_1, \ldots, \alpha_i)$ and β_i and regular input α_{i+1}. This will return randomness r that causes A^i to output coins ρ that are both consistent with γ' and for whom $V(\gamma', \beta_i; \rho) = \alpha_{i+1}$ which is exactly identical to sampling coins from $Coins_x(\gamma)$.

  We now formulate this in the language of Claim 4.2. For $\gamma = (\alpha_1, \beta_1, \ldots, \alpha_i)$ let D_γ be the distribution of choosing a random set of coins ρ from $Coins_x(\gamma)$. Clearly we can implement $Samp$ to sample D_γ by receiving γ as auxiliary input and using $A^i(\gamma)$. We set Dep to output the next verifier message $\alpha = V_x(\gamma; \rho)$. Let $\gamma' = (\alpha_1, \beta_1, \ldots, a_i, \beta_i, \alpha_{i+1})$ be the transcript until round $i+1$ which we want to sample coins that are consistent with and let $\gamma = (\alpha_1, \beta_1, \ldots, \alpha_{i-1}, \beta_{i-1}, \alpha_i, \beta_i)$ be its prefix until round i. We set Dep to receive as auxiliary inputs γ and β_i and output the next verifier message $\alpha = V_x(\gamma, \beta_i; \rho)$. g is, again, the identity function. The distribution $g(D_\gamma)$ conditioned on $Dep_{\gamma,\beta_i}(D_\gamma) = \alpha_{i+1}$ is identical to $Coins_x(\gamma)$ conditioned on $V_x(\gamma; Coins_x(\gamma)) = \alpha_{i+1}$. This is exactly the distribution $Coins_x(\gamma')$ since the set of coins that are consistent with both γ' and α_{i+1} is exactly the set of coins consistent with γ. Together, using Claim 4.2 implies the existence of A^{i+1} that samples from the coins consistent with transcripts of length $i+1$. To show the initial algorithm A^0, transcripts of length 0 have no prover messages, and so we can use the same techniques used for doubly-efficient interactive proofs. This concludes the induction showing how coins can be sampled for every transcript length.

  The induction described above can only be done a constant number of times while preserving the efficiency of the functions. Suppose function A^0 takes time t to compute. Then A^1, which is the

inverter for the function described above that uses A^0 as a sub-procedure, can run in time $O(t^{c_1})$ for some (arbitrary) $c_1 \in \mathbb{N}$. This continues such that A^i runs in time $O(t^{\prod_{j=0}^{i} c_j})$ for some $c_1, \ldots, c_i \in \mathbb{N}$. If $i = \omega(1)$, then this value can be super-polynomial. This restriction is the reason our technique doesn't extend to general protocols with a super-constant number of rounds.

In the general case, in which we wish to sample from $Coins_x(\gamma)$ conditioned on the sample hashing to some value, we add to $Samp$ that it should also output a random hash function h and to $Dep(\rho, h)$ in addition to outputting a transcript of the protocol (or a message α in the case of constant-round proofs), outputs h and $y = h(\rho)$. Implementing HashInv for messages is similar to the ideas we use for coins. For doubly-efficient proofs now $Dep(h, \rho)$, rather than outputting $\gamma = \Pi_x^i(\rho)$ and $y = h(\rho)$ outputs $\gamma = \Pi_x^i(\rho)$ and $y = h(V(\gamma; \rho))$. Additionally, we set $g(\rho)$ to calculate $\gamma = \Pi_x^i(\rho)$ and output $V_x(\gamma; \rho)$ (recall that g is the function used on D in Claim 4.2). For constant-round proofs we use A^i as before, but in this case we will use it to be able to sample messages consistent with an i round transcript (rather than coins consistent with an $i+1$-round transcript as in the previous explanation). Fix $\gamma = (\alpha_1, \beta_1, \ldots, \alpha_i, \beta_i)$. Then $Samp$ samples a random hash function h and uses A to sample $\rho \leftarrow Coins_x(\gamma_{i-1})$. $Dep(h, \rho)$ will output $h(V_x(\gamma; \rho))$.

Claim 4.2. *Let D be a distribution, $Samp$ be an efficient sampler for D with error δ_1 and Dep and g efficiently computable functions. Let $f(r) \triangleq Dep(Samp(r))$. Suppose that f has a distributional inverter Inv with error δ_2. Define $A(d) \triangleq g(Samp(Inv(d)))$. Then when $d \leftarrow Dep(D)$, the distributions $A(d)$ and $g(D)$ conditioned on $Dep(D) = d$ are $(2\delta_1 + \delta_2)$-close.*

Proof. We begin by observing that:

$$\Delta\Big(\big(Samp(U_\tau), Dep(Samp(U_\tau))\big), \big(D, Dep(D)\big)\Big) \leq \delta_1$$

Suppose towards contradiction that there exists a distinguisher B that has success probability greater than δ_1. Then it can be used to distinguish between $Samp(U_\tau)$ and D. Receiving a sample d we give $(d, Dep(d))$ to B and answer as it answers. If $d \leftarrow D$ then $(d, Dep(d)) \leftarrow (D, Dep(D))$ and if $d \leftarrow Samp(U_\tau)$ then $(d, Dep(d)) \leftarrow Samp(U_\tau), Dep(Samp(U_\tau))$ and so the distinguisher manages to distinguish the cases with probability greater than δ_1.

Now consider the inverter Inv. By definition of f and distributional inversion:

$$\Delta\Big(\big(U_\tau, Dep(Samp(U_\tau))\big), \big(Inv(Dep(Samp(U_\tau))), Dep(Samp(U_\tau))\big)\Big) \leq \delta_2$$

Applying $Samp$ and then g to the left expression of each part and substituting $A(\cdot)$ for $g(Samp(Inv(\cdot)))$ we have that:

$$\Delta\Big(\big(g(Samp(U_\tau)), Dep(Samp(U_\tau))\big), \big(A(Dep(Samp(U_\tau))), Dep(Samp(U_\tau))\big)\Big) \leq \delta_2$$

Now, we using the triangle inequality we replace the left hand side with $(g(D), Dep(D))$ by our initial observation, so that

$$\Delta\Big(\big(g(D), Dep(D)\big), \big(A(Dep(Samp(U_\tau))), Dep(Samp(U_\tau))\big)\Big) \leq \delta_1 + \delta_2$$

Finally, we use the triangle inequality to replace $Samp(U_\tau)$ to D on the right-hand side which increases the statistical distance by at most δ_1:

$$\Delta\Big(\big(g(D), Dep(D)\big), \big(A(Dep(D)), Dep(D)\big)\Big) \leq 2\delta_1 + \delta_2$$

Let $d \leftarrow Dep(D)$. To see that $A(d)$ and $g(D)$ conditioned on $Dep(D) = d$ have statistical distance at most $delta = 2\delta_1 + \delta_2$ notice that $(g(D), Dep(D))$ is identical to the process in which d is sampled from $Dep(D)$ and then z is sampled from D conditioned on $Dep(D) = d$ and outputting $(g(z), d)$. □

Remark 4.3. An identical claim to Claim 4.2 if the distribution under question, D_z, is defined by auxiliary input $z \in S$ for some set $S \subseteq \{0,1\}^*$. In this case since $Samp$ receives this auxiliary input, f and A also receive it. The inverter Inv will need to be an *auxiliary-input* distributional inverter. Furthermore, if Inv only manages to invert for certain auxiliary-input lengths $\mathcal{I} \subseteq \mathbb{N}$, then A only samples close to $g(D_z)$ conditioned on $Dep(D_z) = d$ when considering $z \in S \cap (\cup_{i \in \mathcal{I}} \{0,1\}^i)$.

4.1.1 Doubly-Efficient Interactive Proofs

In this section we show how to implement hash inversion oracles for the distributions that we will require in the emulation protocol when the protocol $\Pi = \langle P, V \rangle$ that we are trying to emulate is doubly-efficient.

Proposition 4.4. *Let $\Pi = \langle P, V \rangle$ be a doubly-efficient interactive proof with r-rounds, ℓ bits of verifier randomness and verifier messages of length m. Then there exists an efficiently computable function f with the following property: Suppose there exists an auxiliary-input distributional inverter for f with error δ. Then there exist efficient randomized algorithms A and B such that for every $x \in \mathcal{L}$ and $0 \leq i \leq 2r$:*

- *For every $0 \leq k \leq \ell$, if $\gamma \leftarrow \Pi^i_x(\gamma)$ $h \leftarrow \mathcal{H}_{\ell,k}$ and $y \leftarrow h(Coins_x(\gamma))$ then $A_{x,i,k}(h, y, \gamma)$ and $Coins_x(\gamma)$ conditioned on $h(Coins_x(\gamma)) = y$ are δ-close.*
- *For every $0 \leq k \leq m$, if $\gamma \leftarrow \Pi^i_x(\gamma)$ $h \leftarrow \mathcal{H}_{m,k}$ and $y \leftarrow h(Msg_x(\gamma))$ then $B_{x,i,k}(h, y, \gamma)$ and $Msg_x(\gamma)$ conditioned on $h(Msg_x(\gamma)) = y$ are δ-close.*

If the inverter inverts only for auxiliary inputs of length $\mathcal{I} \subseteq \mathbb{N}$, then implementation of A and B is for every $x \in \mathcal{L} \cap (\cup_{t \in \mathcal{I}} \{0,1\}^t)$.

Proof. Fix x and i. We define distribution $D^i_{x,k} \equiv (\mathcal{H}_{\ell,k}, \Pi^i_x(U_\ell), U_\ell)$ and functions:

- $Samp^i_{x,k}(h, \rho)$: Output h, ρ.
- $Dep^{coins}_{x,k}(h, \rho)$: Output $h, \Pi^i_x(\rho), h(\rho)$.
- $Dep^{msg}_{x,k}(h, \gamma, \rho)$: Let $\Pi^i_x(\rho)$. Output $h, \gamma, h(V_x(\gamma; \rho))$.
- $g^{coins}_{x,k}(h, \rho)$: Output ρ.
- $g^{msg}_{x,k}(h, \rho)$: Let $\Pi^i_x(\rho)$. Output $V_x(\gamma; \rho)$.

Notice that all of the functions defined are efficient. The functions Dep^{coins},Dep^{msg},g^{coins} and g^{msg} are efficiently computable since at most they calculate h and V which are efficient. $Samp^i$ calculates $\Pi^i_x(\rho)$ which is efficient since Π is doubly-efficient. Additionally, observe the following properties:

1. $Samp^i_{x,k}(\mathcal{H}_{\ell,k}, U_\ell) \equiv D^i_{x,k}$.
2. $Dep^{coins}_{x,k}(D)$ is identical to the distribution output by the process that samples $h \leftarrow \mathcal{H}_{\ell,k}$, $\gamma \leftarrow \Pi^i_x(U_\ell)$ and $y \leftarrow h(Coins_x(\gamma))$.
3. $Dep^{msg}_{x,k}(D)$ is identical to the distribution output by the process that samples $h \leftarrow \mathcal{H}_{\ell,k}$, $\gamma \leftarrow \Pi^i_x(U_\ell)$ and $y \leftarrow h(Msg_x(\gamma))$.
4. $g^{coins}_{x,k}(D) \equiv U_\ell$.
5. $g^{msg}_{x,k}(D)$ is identical to the distribution of choosing $\gamma \leftarrow \Pi^i_x(U_\ell)$ and outputting the last verifier message.

In the following, we omit the auxiliary inputs x and k for clarity. We now use Claim 4.2. Using D^i, $Samp^i$, Dep^{coins} and g^{coins}, and noticing that $Samp(\mathcal{H}_{\ell,k}, U_\ell) \equiv D$, we get that if $Samp^i(Dep^{coins}(\cdot))$ is invertible with error δ, then there is an algorithm A^i such that: When $d \leftarrow Dep^{coins}(D)$, the distributions $A^i(d)$ and $g^{coins}(D)$ conditioned on $Dep^{coins}(D) = d$ are δ-close. Unpacking this, we get that if (h, γ, y)

are drawn from the distribution described in Property 2, then $A^i(h,\gamma,y)$ is δ-close to U_ℓ conditioned on $\Pi^i_x(U_\ell)=\gamma$ and $h(U_\ell)=y$. The distribution U_ℓ conditioned on $\Pi^i_x(U_\ell)=\gamma$ and $h(U_\ell)=y$ is identical to $Coins_x(\gamma)$ conditioned on $h(Coins_x(\gamma))$.

Using D^i, $Samp^i$, Dep^{msg} and g^{msg} we get a similar claim. If $Samp^i(Dep^{msg}(\cdot))$ is invertible with error δ then there exists an algorithm $A^{msg,i}$ such that: If (h,γ,y) are drawn from the distribution described in Property 3, then $B^i(h,\gamma,y)$ is δ-close to $Msg_x(\gamma)$ conditioned on $h(Msg_x(\gamma))=y$.

Observe that for every value of i the above algorithms required inversion of the functions $Samp^i_{x,k}(Dep^{coins}_{x,k}(\cdot))$ and $Samp^i_{x,k}(Dep^{msg}_{x,k}(\cdot))$. Define:

$$f_{b,x,i,k}(h,\rho)=\begin{cases}Samp^i_{x,k}(Dep^{coins}_{x,k}(h,\rho)) & b=0\\ Samp^i_{x,k}(Dep^{msg}_{x,k}(h,\rho)) & b=1\end{cases}$$

Access to an auxiliary-input distributional inverter for f can be used to invert $Samp^i_{x,k}(Dep^{coins}_{x,k}(\cdot))$ and $Samp^i_{x,k}(Dep^{msg}_{x,k}(\cdot))$ for every i, and so inverting this function is sufficient to promise that all of the algorithms exist. Finally, the proposition is true by setting $A_{x,i,k}(h,\gamma,y)=A^i_{x,k}(h,\gamma,y)$ and $B_{x,i,k}(h,\gamma,y)=B^i_{x,k}(h,\gamma,y)$. □

4.1.2 Constant-Round Interactive Proofs

Proposition 4.5. *Let $\Pi=\langle P,V\rangle$ be a constant-round interactive proof with r-rounds, ℓ bits of verifier randomness and verifier messages of length m. Then there exist $2r$ efficiently computable functions with the following property: Suppose there exists auxiliary-input distributional inverters with error δ for all of the functions simultaneously. Then there exist efficient randomized algorithms A and B such that for every $x\in\mathcal{L}$ and $0\le i\le 2r$:*

- *For every $0\le k\le\ell$, if $\gamma\leftarrow\Pi^i_x(\gamma)$ $h\leftarrow\mathcal{H}_{\ell,k}$ and $y\leftarrow h(Coins_x(\gamma))$ then $A_{x,i,k}(h,y,\gamma)$ and $Coins_x(\gamma)$ conditioned on $h(Coins_x(\gamma))=y$ have statistical distance $O(\delta)$.*
- *For every $0\le k\le m$, if $\gamma\leftarrow\Pi^i_x(\gamma)$ $h\leftarrow\mathcal{H}_{m,k}$ and $y\leftarrow h(Msg_x(\gamma))$ then $B_{x,i,k}(h,y,\gamma)$ and $Msg_x(\gamma)$ conditioned on $h(Msg_x(\gamma))=y$ have statistical distance $O(\delta)$.*

If the inverter inverts only for auxiliary inputs of length $\mathcal{I}\subseteq\mathbb{N}$, then implementation of A and B is for every $x\in\mathcal{L}\cap(\cup_{t\in\mathcal{I}}\{0,1\}^t)$.

Proof. We construct A and B by induction on the message number. Specifically, if it is possible, given a transcript of length i, to sample coins that are consistent with a the transcript, then A and B can be constructed to sample coins or messages that hash to some value and are consistent with a transcript of length $i+1$. A for transcripts until round $i+1$ can then be used to sample coins that are consistent with a transcript of length $i+1$.

Basis $i=0$: In this case the transcript is of length 0. Thus we simply need to sample coins or messages that hash to some value. Let $f(h,\rho)=h(\rho)$. A distributional inverter for f with error δ can be used to sample random coins that hash to y by calling it on (h,y). Similarly let $g(h,\rho)=h(V_x(\emptyset;\rho))$ be the hash of the first verifier message induced by randomness ρ. A distributional inverter for g with error δ can be used to sample random coins that hash to y by calling it on (h,y) to receive ρ and outputting $V_x(\emptyset;\rho)$.

Induction Hypothesis: There exist $i+2$ efficiently computable functions such that: Suppose they are all simultaneously auxiliary-input distributionally invertible with error δ then there exist: Then there exist efficient randomized algorithms A^i and B^{i-1} such that for every $x\in\mathcal{L}$:

- For every $1\le j<i$, $0\le k\le\ell$, if $\gamma\leftarrow\Pi^j_x(\gamma)$ $h\leftarrow\mathcal{H}_{\ell,k}$ and $y\leftarrow h(Coins_x(\gamma))$ then $A^i_{x,j,k}(h,y,\gamma)$ and $Coins_x(\gamma)$ conditioned on $h(Coins_x(\gamma))=y$ have statistical distance at most $(2^{i+1}-1)\delta$.
- For every $1\le j<i-1$, $0\le k\le m$, if $\gamma\leftarrow\Pi^j_x(\gamma)$ $h\leftarrow\mathcal{H}_{m,k}$ and $y\leftarrow h(Msg_x(\gamma))$ then $B^i_{x,j,k}(h,y,\gamma)$ and $Msg_x(\gamma)$ conditioned on $h(Msg_x(\gamma))=y$ have statistical distance at most $(2^{i+1}-1)\delta$.

Induction Step $0 < i \leq 2r$**:** We have two cases. If transcripts that proceed untill round i end with a verifier message or a prover message. In both cases, let $M_{x,k}(\gamma) \triangleq A^i_{x,i,0}(\perp, \perp, \gamma)$. That is, A^i used with k set to 0, in which case the hash function and image are defined as $\perp$. $M_{x,k}(\gamma)$, then, samples from $Coins_x(\gamma)$ with error $\delta_1 = (2^{i+1} - 1)\delta$.

1. Verifier Message: Recall that the message distribution for transcripts that end in a verifier message always returns the last message. Thus, $B^{i+1}(h, y, \gamma)$ is trivial - it always returns the final message of the transcript. We show how to implement A^{i+1}. By Claim 4.6, using M, there exists a function f^{coins} such that if it is invertible, then A^{i+1} can be implemented with error $2\delta(2^{i+1} - 1) + \delta = (2^{i+2} - 1)\delta$.
2. Prover Message: In this case A^{i+1} is trivial. Suppose $\gamma = (\gamma', \beta)$ where β is the final prover message. Then $Coins_x(\gamma) \equiv Coins_x(\gamma')$. Therefore we define $A^{i+1} \equiv A^i$. We show how to implement B^{i+1}. By Claim 4.6, using M, there exists a function f^{msg} such that if it is invertible, then A^{i+1} can be implemented with error $2\delta(2^{i+1} - 1) + \delta = (2^{i+2} - 1)\delta$.

Claim 4.6. *Let $n \in \mathbb{N}$ and fix $1 \leq i \leq 2r(n)$. Suppose that there exists an algorithm M such that: For every $x \in \mathcal{L} \cap \{0,1\}^n$, given $\gamma_i \leftarrow \Pi^i_x(U_\ell)$ then $M_{x,k}(\gamma)$ and $Coins_x(\gamma)$ are δ_1-close. Then there exist two functions f^{coins} and f^{msg} with the following property. For every $x \in \mathcal{L}$:*

- *If f^{coins} is auxiliary-input distributionally invertible with error δ_2, then there exists a randomized algorithm A^{i+1}. For every $0 \leq k \leq \ell$, if $\gamma \leftarrow \Pi^{i+1}_x(\gamma)$ $h \leftarrow \mathcal{H}_{\ell,k}$ and $y \leftarrow h(Coins_x(\gamma))$ then $A^{i+1}_{x,k}(h, y, \gamma)$ and $Coins_x(\gamma)$ conditioned on $h(Coins_x(\gamma)) = y$ are $(2\delta_1 + \delta_2)$-close.*
- *If f^{msg} is auxiliary-input distributionally invertible with error δ_2, then there exists a randomized algorithm B^{i+1}.For every $0 \leq k \leq m$, if $\gamma \leftarrow \Pi^i_x(\gamma)$ $h \leftarrow \mathcal{H}_{m,k}$ and $y \leftarrow h(Msg_x(\gamma))$ then $B^{i+1}_{x,k}(h, y, \gamma)$ and $Msg_x(\gamma)$ conditioned on $h(Msg_x(\gamma)) = y$ are $(2\delta_1 + \delta_2)$-close.*

Proof. We begin by showing A^{i+1}. Fix x, i and $\gamma = (\alpha_1, \beta_1, \ldots, \alpha_i, \beta_i)$. We define distribution $D^i_{x,\gamma,k}$ to be the output of the following process: Choose $h \leftarrow \mathcal{H}_{\ell,k}$ and $\rho \leftarrow Coins_x(\gamma)$ and output $(h, \rho, V_x(g; \rho))$. Additionally define functions:

- $Samp^i_{x,\gamma,k}(h, r)$: Let $\rho \leftarrow M_{x,k}(\gamma)$. Output h, ρ.
- $Dep^{coins}_{x,\gamma,k}(h, \rho)$: Output $h, V_x(\gamma; \rho), h(\rho)$.
- $g^{coins}_{x,k}(h, \rho)$: Output ρ.

Notice that since M is efficient, all of the functions defined are efficient. Additionally, observe the following properties, supposing that $Samp$ uses τ random coins:

1. $\Delta\left(Samp^i_{x,\gamma,k}(\mathcal{H}_{\ell,k}, U_\tau), D_{x,\gamma,k}\right) \leq \delta_1$.
2. $Dep_{x,\gamma,k}(D)$ is identical to the distribution output by the process that samples $h \leftarrow \mathcal{H}_{\ell,k}$, $\rho \leftarrow Coins_x(\gamma)$ and outputs h, $\alpha = V(\gamma; \rho)$ and $y = h(\rho)$.
3. $g_{x,k}(D)$ is equivalent to $Coins_x(\gamma)$.

Using Claim 4.2, if the function $Dep(Samp(\cdot))$ is invertible with error δ_2 then there exists an algorithm A^{i+1} such that: Given auxiliary input γ and regular inputs (α, y) drawn from the distribution $Dep(D)$ returns a value distributed $(2\delta_1 + \delta_2)$-close to D conditioned on $Dep(D) = (\alpha, y)$. D conditioned on $Dep(D) = (\alpha, y)$ reduces to $(\mathcal{H}_{\ell,k}, Coins_x(\gamma))$ conditioned on $V_x(\gamma; Coins_x(\gamma)) = \alpha$ and $h(Coins_x(\gamma)) = y$. Observe that the set coins that are consistent with γ and lead to α is exactly the set of coins that lead to the transcript (γ, α). Thus, when looking only at the "coin" part of it, as g does, the distribution reduces to $Coins_x(\gamma, \alpha)$ conditioned on $h(Coins_x(\gamma, \alpha)) = y$.

We now show B^{i+1}. Fix x, i and $\gamma = (\alpha_1, \beta_1, \ldots, \alpha_i, \beta_i)$. We define distribution $D^i_{x,\gamma,k}$ to be the output of the following process: Choose $h \leftarrow \mathcal{H}_{\ell,k}$ and $\rho \leftarrow Coins_x(\gamma)$ and output $(h, \rho, V_x(g; \rho))$. Additionally define functions:

- $Samp^{i}_{x,\gamma,k}(h,r)$: Let $\rho \leftarrow M_{x,k}(\gamma)$. Output h, ρ.
- $Dep^{coins}_{x,\gamma,k}(h,\rho)$: Output $h, h(V_x(\gamma;\rho))$.
- $g^{coins}_{x,k}(h,\rho)$: Output $V_x(\gamma;\rho)$.

Notice that since M is efficient, all of the functions defined are efficient. Additionally, observe the following properties, supposing that $Samp$ uses τ random coins:

1. $\Delta\left(Samp^{i}_{x,\gamma,k}(\mathcal{H}_{\ell,k}, U_\tau), D_{x,\gamma,k}\right) \leq \delta_1$.
2. $Dep_{x,\gamma,k}(D)$ is identical to the distribution output by the process that samples $h \leftarrow \mathcal{H}_{\ell,k}$, and outputs h, $h(Msg_x(\gamma))$.
3. $g_{x,k}(D)$ is equivalent to $Msg_x(\gamma)$.

As in the previous case, we use Claim 4.2, which implies that if the function $Dep(Samp(\cdot))$ is invertible with error δ_2 then there exists an algorithm A^{i+1} such that: Given auxiliary input γ and regular inputs (α, y) drawn from the distribution $Dep(D)$ returns a value distributed $(2\delta_1 + \delta_2)$-close to D conditioned on $Dep(D) = (\alpha, y)$. D conditioned on $Dep(D) = (\alpha, y)$ reduces to $(\mathcal{H}_{\ell,k}, Coins_x(\gamma))$ conditioned on $h(V_x(\gamma; Coins_x(\gamma))) = y$. □

□

4.2 Implementing Subset

As mentioned in the overview, our approach to emulation requires splitting the space of messages into subsets, where belonging to a subset implies a bound on the number of coins that are consistent with the message. Recall that $\mathsf{Subset}^{\delta}_{D,\mathbb{D}}$, defined in Section 3.1, is as follows: Parametrized by a distribution D over $\{0,1\}^\ell$ and a partitioning of D, $\mathbb{D} = \{D_i\}_{i=1}^{I}$. Receives as input a value $\rho \leftarrow D$. With probability $1-\delta$ returns $i \in [I]$ such that $\rho \in Supp(D_i)$.

In this section we show that if the prover has access to the $\mathsf{HashInv}$ oracles described in Section 4.1, then it can implement $\mathsf{Subset}_{D,\mathbb{D}}$ where, given input x and partial transcript γ, the distribution D will be the verifier next message distribution $Msg_x(\gamma)$ (see Definition 2.11) and $\mathbb{D}$ will be a partitioning of the support of the message distribution such that for every $\alpha \in Supp(Msg_x(\gamma))$, if α belongs to subset i, then the number of coins that are consistent with the transcript (γ, α) is between $(1+\varepsilon)^{i-2}$ and $(1+\varepsilon)^{i+1}$. This partitioning defines "approximate clusters" of the message distribution:

Proposition 4.7. *Let $\Pi = \langle P, V\rangle$ be an interactive proof. Fix $x \in \{0,1\}^n$ and $i \in [r]$. Then for $0 < \delta \leq 1$ there exists an oracle-aided PPTM that for $\gamma \leftarrow \Pi^i_x(U_\ell)$ implements $\mathsf{Subset}^{\delta'}_{Msg_x(\gamma),\mathbb{D}}$ for error $\delta' = 2^{-\frac{1}{\varepsilon}} \cdot \left(\frac{\varepsilon}{m\ell}\right)^8 + O\left(\left(\frac{m\ell}{\varepsilon}\right)^{15} \cdot \delta\right)$, and a partitioning $\mathbb{D} = \{D_i\}_{i=1}^{\log_{1+\varepsilon}(2^\ell)}$ such that for every $\alpha \in D_i$:*

$$(1+\varepsilon)^{i-2} \leq |Coins_x(\gamma, \alpha)| \leq (1+\varepsilon)^{i+1}$$

The algorithm receives access to an oracle that for every γ' acts as $\mathsf{HashInv}^{\delta}_{Coins_x(\gamma')}$. Moreover, a call to $\mathsf{Subset}^{\delta'}_{Msg_x(\gamma),\mathbb{D}}$ takes time $O\left(\left(\frac{m\ell}{\varepsilon}\right)^{15}\right)$ and makes $O\left(\left(\frac{m\ell}{\varepsilon}\right)^{15}\right)$ oracle calls.

Remark 4.8. The requirement that the oracle acts as $\mathsf{HashInv}^{\delta}_{Coins_x(\gamma')}$ for any γ' can be weakened to only work when γ' is a random transcript drawn from $\Pi^{i+1}_x(U_\ell)$.

Proof. On input α, all that the algorithm will call the cardinality approximation procedure trying to approximate the number of coins consistent with the transcript γ, α and use that to find to which subset the message belongs. In more detail: the algorithm for $\mathsf{Subset}^{\delta'}_{Msg_x(\gamma),\mathbb{D}}$ on input $\alpha \leftarrow Msg_x(\gamma)$ is as follows:

1. Execute the cardinality approximation procedure (Lemma 3.18) with approximation parameter ε and failure probability $2^{-\frac{1}{\varepsilon}} \cdot \left(\frac{\varepsilon}{m\ell}\right)^8$, using $\mathsf{HashInv}^{\delta}_{Coins_x(\gamma,\alpha)}$. Denote the value output by the procedure as a.
2. Output $\max\left\{1, 1 + \min\left\{\log_{1+\varepsilon}(2^\ell), \left\lfloor \log_{1+\varepsilon}(a) \right\rfloor\right\}\right\}$.

Let a be the value output by $\mathsf{Subset}^{\delta'}_{Msg_x(\gamma),\mathbb{D}}$ on an execution on $\alpha \leftarrow Msg_x(\gamma)$. Then by Lemma 3.18, since we use $\mathsf{HashInv}^{\delta}_{Coins_x(\gamma,\alpha)}$ which has error δ, we have that $a \in (1 \pm \varepsilon)\left|Coins_x\left(\gamma,\alpha\right)\right|$ except with probability:

$$2^{-\frac{1}{\varepsilon}} \cdot \left(\frac{\varepsilon}{m\ell}\right)^8 + O\left(\frac{\ell^4}{\varepsilon^6} \log\left(2^{\frac{1}{\varepsilon}} \cdot \left(\frac{m\ell}{\varepsilon}\right)^8\right) \cdot \delta\right) = 2^{-\frac{1}{\varepsilon}} \cdot \left(\frac{\varepsilon}{m\ell}\right)^8 + O\left(\left(\frac{m\ell}{\varepsilon}\right)^{15} \cdot \delta\right)$$

In the case that this event does occur and a is in the correct range, we have that:

$$(1+\varepsilon)^{-1}a \leq \left|Coins_x\left(\gamma,\alpha\right)\right| \leq (1-\varepsilon)^{-1}a \leq (1+\varepsilon)^2 a$$

Fix α and let $j = \max\left\{1, 1 + \min\left\{\log_{1+\varepsilon}(2^\ell), \left\lfloor \log_{1+\varepsilon} a \right\rfloor\right\}\right\}$ be the output of the algorithm with regards to input α. Then conditioned on a being bounded as described above, we show that $(1+\varepsilon)^{j-2} \leq \left|Coins_x(\gamma,\alpha)\right| \leq (1+\varepsilon)^{j+1}$. We have three number of cases:

- $1 \leq a \leq 2^\ell$: Then $j = \left\lfloor \log_{1+\varepsilon} a \right\rfloor + 1$. In this case, $(1+\varepsilon)^{j-1} \leq a < (1+\varepsilon)^j$, and so:

  $$(1+\varepsilon)^{j-2} \leq (1+\varepsilon)^{-1}a \leq \left|Coins_x\left(\gamma,\alpha\right)\right|$$

  and

  $$\left|Coins_x\left(\gamma,\alpha\right)\right| \leq (1+\varepsilon)^2 a < (1+\varepsilon)^{j+1}$$

- $2^\ell < a$: Then $j = \log_{1+\varepsilon}(2^\ell) + 1$. Since $\left|Coins_x\left(\gamma,\alpha\right)\right| \leq 2^\ell$, this implies:

  $$(1+\varepsilon)^{j-2} \leq (1+\varepsilon)^{-1}2^\ell < (1+\varepsilon)^{-1}a \leq \left|Coins_x\left(\gamma,\alpha\right)\right|$$

  and

  $$\left|Coins_x\left(\gamma,\alpha\right)\right| \leq 2^\ell < (1+\varepsilon)^{j+1}$$

- $a < 1$: Then $j = 1$. Since $1 \leq \left|Coins_x\left(\gamma,\alpha\right)\right|$ this implies:

  $$(1+\varepsilon)^{j-2} < 1 \leq \left|Coins_x\left(\gamma,\alpha\right)\right|$$

  and

  $$\left|Coins_x\left(\gamma,\alpha\right)\right| \leq (1+\varepsilon)^2 a < (1+\varepsilon)^2 = (1+\varepsilon)^{j+1}$$

In every one of the cases is is true that $(1+\varepsilon)^{j-2} \leq \left|Coins_x(\gamma,\alpha)\right| \leq (1+\varepsilon)^{j+1}$.

Thus, we have that if α is a random message consistent with γ, then with probability at least

$$1 - 2^{-\frac{1}{\varepsilon}} \cdot \left(\frac{\varepsilon}{m\ell}\right)^8 + O\left(\left(\frac{m\ell}{\varepsilon}\right)^{15} \cdot \delta\right)$$

the classification of α by $\mathsf{Subset}_{Msg_x(\gamma),\mathbb{D}}$ will imply the required bounds on the number of coins leading to α. The running time of the cardinality approximation procedure is $O\left(\frac{\ell^4}{\varepsilon^6} \log\left(2^{\frac{1}{\varepsilon}} \cdot \left(\frac{m\ell}{\varepsilon}\right)^8\right)\right) = O\left(\left(\frac{m\ell}{\varepsilon}\right)^{15}\right)$ and it makes $O\left(\left(\frac{m\ell}{\varepsilon}\right)^{15}\right)$ calls to $\mathsf{HashInv}$.

Two remarks should be made about the above procedure. The first is that since it is randomized the algorithm might return different indices for the same message- even when it does not fail. It therefore does not induce a partitioning. This is easily remedied by keeping a list of messages and their subsets,

The Emulation Protocol

Joint Input: $x \in \{0,1\}^n$

Parameters: Sampling error $0 < \varepsilon < 1$

Prover Oracle Access: For every γ, $\mathsf{HashInv}^{\delta}_{Coins_x(\gamma)}$ and $\mathsf{HashInv}^{\delta}_{Msg_x(\gamma)}$.

Protocol:

1. Let $\gamma_0 = \emptyset$ and $N_0 = 2^\ell$.
2. For $i = 1$ to r:
   (a) The prover uses $\mathsf{HashInv}^{\delta}_{Coins_x(\gamma)}$ in order to implement $\mathsf{Subset}^{\delta'}_{Msg_x(\gamma),\mathbb{D}}$ for $\delta' = 2^{-\frac{1}{\varepsilon}} \cdot \left(\frac{\varepsilon}{m\ell}\right)^8 + O\left(\left(\frac{m\ell}{\varepsilon}\right)^{15} \cdot \delta\right)$, as described in Proposition 4.7.
   (b) Execute the sampling via subsets protocol (Corollary 3.17) with parameters ε and $I = \log_{1+\varepsilon}(2^\ell)$ for the distribution $Msg_x(\gamma_{i-1})$ and the partitioning $\mathbb{D}$. The prover is given access to $\mathsf{HashInv}^{\delta'}_{Msg_x(\gamma_{i-1})}$ and $\mathsf{Subset}^{\delta'}_{Msg_x(\gamma_{i-1}),\mathbb{D}}$, and input $\mu = (1+\varepsilon)^3$. The function f is implemented as follows: Receive as input i and p, return $p(1+\varepsilon)^{-(i+1)} \cdot N_{i-1}$.
   (c) If the verifier does not reject, the protocol will end with both parties agreeing on some message $\alpha_i \in \{0,1\}^m$ and a subset index $j \in \{1,\ldots,I\}$.
   (d) The prover sends $\beta_i = P_x(\gamma_{i-1}, \alpha_i)$ to the verifier
   (e) Both parties output $\gamma_i = (\gamma_{i-1}, \alpha_i, \beta_i)$ and $N_i = \max\left\{1, 2^{j-2}\right\}$
3. The final iteration ends with N_r and $\gamma_r = \alpha_1\beta_1, \ldots, \alpha_{r-1}, \beta_{r-1}, \rho$. The verifier accepts if $N_r = 1$, $\forall t \le r:\ \alpha_t = V_x(\gamma_{t-1}; \rho)$, and $V_x(\gamma_r; \rho)$ accepts.

Figure 6: The Emulation Protocol

and beginning by checking if a message belongs to the list. If it does, return whatever was returned the previous time the algorithm was queried with this message. Otherwise, execute the cardinality approximation procedure and enter the message and result into the list. The second remark is that since the algorithm is randomized, the partitioning $\mathbb{D}$ is not defined a-priory, but only during execution. The important property is that there *exists* some partitioning of the message space that agrees (with probability δ') with answers given by $\mathsf{Subset}_{Msg_x(\gamma),\mathbb{D}}$ for which all messages in subset i, the number of coins leading to them is within the correct range. □

4.3 The Emulation Protocol

Proposition 4.9. *Let $\Pi = \langle P, V\rangle$ be an r-round interactive proof for language $\mathcal{L}$ with ℓ bits of randomness, m-bit messages, completeness c, soundness error s, prover running time t_P and verifier running time t_V. Suppose that Π ends with the verifier sending its entire randomness to the prover. Then the emulation protocol described in Figure 6, with parameter $0 < \varepsilon < \frac{1}{2}$ is a public-coin interactive proof with the following properties:*

Verifier Efficiency: Running time $O(t_V)$, Randomness $O(rm)$

Prover Efficiency: Running time $t_P + O\left(\left(\frac{m\ell}{\varepsilon}\right)^{22} \cdot r\right)$, Oracle Calls: $O\left(\left(\frac{m\ell}{\varepsilon}\right)^{22} \cdot r\right)$

Number Of Messages: $2r+1$ ($r+1$ rounds)

Completeness: *For every $x \in \mathcal{L}$, if for every γ, the prover has access to $\mathsf{HashInv}^{\delta}_{Coins_x(\gamma)}$ and $\mathsf{HashInv}^{\delta}_{Msg_x(\gamma)}$ then the probability that the verifier accepts when interacting with the honest prover on input x is at least $c - \left(28r\varepsilon + O\left(\left(\frac{m\ell}{\varepsilon}\right)^{23} \cdot r\delta\right)\right)$.*

Soundness: *If $x \notin \mathcal{L}$ then for any (computationally unbounded) cheating prover M^*, the probability that the verifier accepts when interacting with M^* on input x is at most $O\left(\varepsilon^{-5} r\ell \cdot s^{\frac{1}{r}}\right)$.*

Remark 4.10. For the completeness property it is possible to relax the requirement on the oracles $\mathsf{HashInv}^{\delta}_{Coins_x(\gamma)}$ and $\mathsf{HashInv}^{\delta}_{Msg_x(\gamma)}$ such that they are only required to work for random transcripts rather than for all γ, i.e. for any $i \in [r]$ they are only required to work if $\gamma \leftarrow \Pi^i_x(U_\ell)$. This observation is true since in Claim 4.13 we analyze the protocol as if the previous transcript was chosen at random, and this is the only place in which these oracles are used in the analysis.

Remark 4.11. Proposition 4.9 assumes that the protocol being emulated ends with the verifier sending over its entire randomness to the prover. Any protocol can be trivially transformed to one of this form. Therefore transforming protocols with r rounds that do not end with the verifier revealing its randomness yields public-coin protocols with the same parameters as in Proposition 4.9, except with $2r + 3$ messages ($r + 2$ rounds).

Proposition 4.9 when taking into consideration the above remark, and Propositions 4.4 and 4.5 yields the following theorems showing that for every protocol, there are functions such that whenever they are invertible emulation is possible:

Theorem 5 (Emulation of Doubly-Efficient Interactive Proofs)**.** *Let $\langle P, V\rangle$ be a doubly-efficient interactive proof for language $\mathcal{L}$ with completeness c, soundness s, verifier randomness ℓ, r rounds, prover running time t_P and verifier running time t_V which ends with the verifier sending its randomness to the prover. Then there exists a single function f such that if f has an auxiliary-input distributional inverter with error δ then for every $0 < \varepsilon < \frac{1}{2}$, $\langle P, V\rangle$ can be transformed into a public-coin interactive proof with the following properties:*

Verifier Efficiency: *Running time $O(t_V)$, Randomness $O(rm)$*

Prover Efficiency: *Running time $t_P + \mathrm{poly}(n)$*

Number Of Messages: *$2r + 3$ ($r + 2$ rounds)*

Completeness: *For every $x \in \mathcal{L}$ the probability that the verifier accepts when interacting with the honest prover on input x is $c - \left(28r\varepsilon + O\left(\left(\frac{m\ell}{\varepsilon}\right)^{23} \cdot r\delta\right)\right)$.*

Soundness: *If $x \notin \mathcal{L}$ then for any (computationally unbounded) cheating prover M^*, the probability that the verifier accepts when interacting with M^* on input x is at most $O\left(\varepsilon^{-5} r\ell \cdot s^{\frac{1}{r}}\right)$.*

If the inverter for f inverts only for auxiliary inputs of length $\mathcal{I} \subseteq \mathbb{N}$, then the proof is complete for $x \in \mathcal{L} \cap (\cup_{t\in\mathcal{I}}\{0,1\}^t)$.

Theorem 6 (Emulation of Constant-Round Interactive Proofs)**.** *Let $\langle P, V\rangle$ be a constant-round interactive proof for language $\mathcal{L}$ with completeness c, soundness s, verifier randomness ℓ, r rounds, prover running time t_P and verifier running time t_V which ends with the verifier sending its randomness to the prover. Then there exists a sequence of $2r + 3$ functions $(f_1, \ldots, f_{2r+3})$ where f_i is dependent on inversion oracles for $(f_1, \ldots, f_{i-1})$, such that if every function is simultaneously auxiliary-input distributionally invertible with error δ then for every $0 < \varepsilon < \frac{1}{2}$, $\langle P, V\rangle$ can be transformed into a public-coin interactive proof with the following properties:*

Verifier Efficiency: *Running time $O(t_V)$, Randomness $O(rm)$*

Prover Efficiency: Running time $t_P + \text{poly}(n)$

Number Of Messages: $2r + 3$ *($r + 2$ rounds)*

Completeness: For every $x \in \mathcal{L}$ the probability that the verifier accepts when interacting with the honest prover on input x is $c - \left(28r\varepsilon + O\left(\left(\frac{m\ell}{\varepsilon}\right)^{23} \cdot r\delta\right)\right)$.

Soundness: If $x \notin \mathcal{L}$ then for any (computationally unbounded) cheating prover M^*, the probability that the verifier accepts when interacting with M^* on input x is at most $O\left(\varepsilon^{-5} r\ell \cdot s^{\frac{1}{r}}\right)$.

If the inverters invert simultaneously only for auxiliary inputs of length $\mathcal{I} \subseteq \mathbb{N}$, then the proof is complete for $x \in \mathcal{L} \cap (\cup_{t\in\mathcal{I}}\{0,1\}^t)$.

Remark 4.12. Theorem 6 works only for constant-round proofs due to a dependency on the functions $(f_1, \ldots, f_{2r+3})$. In more detail, for every i, the running time of f_i is polynomially related to the running time of f_{i-1}. Thus the running time of f_{2r+3} is exponential in r. Since the honest prover uses this function in its computations, this limits the number of rounds that can be supported while keeping the prover overhead polynomial.

4.3.1 Efficiency

The protocol makes r iterations in Step 2. In every iteration, the running time and number of random bits used by the verifier while running the sampling via subsets protocol are $O(m)$, and the running time of the honest prover in the sampling via subsets protocol is $O\left(\frac{\mu m^3 I^4}{\varepsilon^3}\right) = O\left(\frac{m^4\ell^4}{\varepsilon^7}\right)$ and it makes the same amount of oracle calls to $\mathsf{HashInv}^{\delta'}_{Mess_x(\gamma)}$ and $\mathsf{Subset}^{\delta'}_{Msg_x(\gamma_{i-1}),\mathbb{D}}$. By Proposition 4.7, due to how we implement it, each call to $\mathsf{Subset}^{\delta'}_{Msg_x(\gamma_{i-1}),\mathbb{D}}$ takes time $O\left(\left(\frac{m\ell}{\varepsilon}\right)^{15}\right)$ and makes the same number of oracle calls. Additionally, the prover executes P once, taking time $t_{P,i}$. Therefore in iteration i, the prover takes time $t_{P,i} + O\left(\left(\frac{m\ell}{\varepsilon}\right)^{22}\right)$ Thus, in Step 2 altogether, the verifier uses randomness and runs in time $O(rm)$, and the prover runs in time $t_P + O\left(\left(\frac{m\ell}{\varepsilon}\right)^{22} \cdot r\right)$ and makes $O\left(\left(\frac{m\ell}{\varepsilon}\right)^{22} \cdot r\right)$ calls to its oracles. Finally, in Step 3 the verifier tests that the transcript generated is consistent and accepting, which takes time t_V. Noting that $rm \leq t_V$, altogether, the verifier runs in time $O(t_V)$, and the verifier runs in time $t_P + O\left(\left(\frac{m\ell}{\varepsilon}\right)^{22} \cdot r\right)$.

The sampling via subsets protocol is a 3 message protocol, starting and ending with a prover message. The prover then sends β, which can be merged with the final message of the sampling via subsets protocol, and so the protocol has 3 messages in each iteration, beginning and ending with a prover message. We can merge the prover's messages at the beginning and ending of each iteration, except for the first prover message. Since the final message of the original protocol is assumed to be a verifier message (its randomness), there is no final prover message. Therefore there are $2r + 1$ messages in total.

4.3.2 Completeness

Our goal is to show that if the prover is honest, the transcript that is generated behaves like a transcript in the original protocol, and that in addition the size claim (the N values) is always a lower-bound on the number of coins consistent with the protocol so far.

This will be done by induction - assuming that γ_{i-1} is close to a random transcript up till round $i-1$, and that N_{i-1} is a lower-bound on the number of coins that are consistent with γ_{i-1}, we show that γ_i is close to a random transcript up till round i, and N_i is a lower-bound on the number of coins consistent with γ_i.

Claim 4.13. *Fix $i \in [r]$ and let $d = 14\varepsilon + O\left(\left(\frac{m\ell}{\varepsilon}\right)^{23} \cdot \delta\right)$. Suppose that $\gamma_{i-1} \leftarrow \Pi_x^{2(i-1)}(U_\ell)$ is a random partial transcript of the original protocol up to round $i-1$, and $N_{i-1} \leq |Coins_x(\gamma_{i-1})|$. Then with probability $1 - d$:*

1. *The γ_i is drawn from a distribution that is of statistical distance at most d from $\Pi_x^{2i}(U_\ell)$.*
2. $1 \leq N_i \leq |Coins_x(\gamma_i)|$.

Remark 4.14. It is actually true that (γ_i, N_i) is d-close to a distribution (W_1, W_2) such that W_1 is a random transcript consistent with γ_{i-1} and if γ', N' are drawn from (W_1, W_2) then $N' \leq |Coins_x(\gamma')|$. Using this description will result in a slightly better bound, but the above description is easier to digest.

Proof. We wish to use Corollary 3.17 to show that the output of the sampling via subsets protocol is close to sampling from $Msg_x(\gamma_{i-1})$ along with its subset index. In order to invoke Corollary 3.17, it must be the case that for every $k \in [I]$:

1. $(1+\varepsilon)^3 = \mu \geq \frac{\max_{\alpha \in Supp(D_k)} \Pr[Msg_x(\gamma_{i-1})=\alpha]}{\min_{\alpha \in Supp(D_k)} \Pr[Msg_x(\gamma_{i-1})=\alpha]}$.
2. There exists a distribution Z_k over $Supp(D_k)$ such that $\Delta(D_k, Z_k) \leq 10\varepsilon$.
3. For every p such that $0 < p \leq \Pr[D \in Supp(D_k)]$: $p(1+\varepsilon)^{-(k+1)} N_{i-1} = f(k,p) \leq 2^{H_\infty(Z_k)}$ where Z_k is the distribution from the previous item.

Fix $k \in [I]$. The first property is true by the definition of the partitioning: The prover has oracle access to $\mathsf{HashInv}^{\delta}_{Coins_x(\gamma_{i-1})}$ and $\mathsf{HashInv}^{\delta}_{Mess_x(\gamma_{i-1})}$, and so by Section 4.2 this allows it to implement $\mathsf{Subset}^{\delta'}_{Msg_x(\gamma_{i-1}),\mathbb{D}}$ for $\delta' = 2^{-\frac{1}{\varepsilon}} \cdot \left(\frac{\varepsilon}{m\ell}\right)^8 + O\left(\left(\frac{m\ell}{\varepsilon}\right)^{15} \cdot \delta\right)$ and a partitioning such that if $\alpha \in Supp(D_k)$, then:

$$(1+\varepsilon)^{k-2} \leq |Coins_x(\gamma_{i-1}, \alpha)| \leq (1+\varepsilon)^{k+1}$$

Property 1 is true since the number of coins that lead to a message is precisely equivalent to its probability in the message distribution. We have yet to show that items 2 and 3 are true. We define Z_k as the uniform distribution over all messages given in $Supp(D_k)$. We use Claim 2.15 to show that D_k is close to flat. Set $z = (1+\varepsilon)^{k+1}$ and let $x_\alpha = |Coins_x(\gamma_{i-1}, \alpha)|$. Notice that:

$$(1-5\varepsilon)x_\alpha < x_\alpha \leq z_\alpha \leq x_\alpha(1+\varepsilon)^3 \leq x_\alpha(1+5\varepsilon)$$

Where $(1+\varepsilon)^3 < 1+5\varepsilon$ since $\varepsilon < \frac{1}{2}$. Further notice that for every $\alpha \in Supp(D_k)$, $\Pr[Z_k = \alpha] = \frac{z}{\sum_{\alpha' \in Supp(D_k)} z}$ and $\Pr[D_k = \alpha] = \frac{x_\alpha}{\sum_{\alpha' \in Supp(D_k)} x_{\alpha'}}$. Therefore by Claim 2.15, D_k is 10ε-close to Z_k, the flat distribution over the same support, proving Item 2.

We now turn to showing that Item 3 is true, namely that, given $p \leq \Pr[Msg_x(\gamma_{i-1}) \in Supp(D_k)]$:

$$-\log\left(p(1+\varepsilon)^{-(k+1)} N_{i-1}\right) = \log f(k,p) \leq H_\infty(Z_k)$$

There are at least N_{i-1} coins which are consistent with γ_{i-1}, which since $p \leq \Pr[Msg_x(\gamma_{i-1}) \in Supp(D_k)]$ implies that there are at least pN_{i-1} different coins leading to the support of D_k. This is because the message distribution is defined as the probability that a message is output by choosing a random consistent coin, and the distribution on coins is uniform. Each message in D_k has at most $(1+\varepsilon)^{k+1}$ consistent coins leading to it and so the support of D_k is of size at least $\frac{pN_{i-1}}{(1+\varepsilon)^{k+1}}$. Z_k and D_k have identical supports, and since Z_k is uniform, its min-entropy is equal to the logarithm of the inverse of the size of its support. Therefore the min-entropy of Z_k is at least $-\log\left(p(1+\varepsilon)^{-(k+1)} N_{i-1}\right)$.

We have shown that all of the requirements for Corollary 3.17 are held, which implies that the pair (α_i, j) are drawn from a distribution of statistical distance at most

$$\begin{aligned} 10\varepsilon + 3\varepsilon + O\left(\left(\frac{mI\mu}{\varepsilon}\right)^4 \cdot \delta'\right) &= 13\varepsilon + O\left(\left(\frac{m\ell}{\varepsilon}\right)^8 \cdot \delta'\right) \\ &= 14\varepsilon + O\left(\left(\frac{m\ell}{\varepsilon}\right)^{23} \cdot \delta\right) \\ &= d \end{aligned}$$

from the distribution of drawing a message from $Msg_x(\gamma_{i-1})$ along with the subset to which it belongs.

Thus, except with probability d, $\alpha_i \in Supp(D_j)$, meaning that $(1+\varepsilon)^{j-2} \leq |Coins_x(\gamma_{i-1}, \alpha_i)|$. Noting that it is always true that $1 \leq |Coins_x(\gamma, \alpha)|$ (since α_i is consistent with γ_{i-1}), setting $N_i = \max\{1, (1+\varepsilon)^{i-2}\}$ we have that, indeed, $N_i \leq |Coins_x(\gamma_{i-1}, \alpha_i)|$. Since the prover's messages do not affect the number of coins consistent with the transcript, $Coins_x(\gamma_{i-1}, \alpha_i) = Coins_x(\gamma_{i-1}, \alpha_i, \beta_i)$. Together we have that except with probability d, $N_i \leq |Coins_x(\gamma_i)|$.

Since (α_i, j) are close to being drawn from $Msg_x(\gamma_{i-1})$ and the subset to which it belongs, we have that α_i is of statistical distance at most d from $Msg_x(\gamma_{i-1})$. Then γ_i is distributed at most d-far from $(\gamma_{i-1}, Msg_x(\gamma_{i-1}), P(\gamma_{i-1}, Msg_x(\gamma_{i-1})))$ (i.e. the distribution of continuing the transcript consistently with γ_{i-1}). Since γ_{i-1} was a random transcript up till round $i-1$, this distribution is identical to $\Pi_x^i(U_\ell)$. □

As in Claim 4.13, we let $d = 14\varepsilon + O\left(\left(\frac{m\ell}{\varepsilon}\right)^{23} \cdot \delta\right)$. Since γ_0 is empty in both the real and the emulated execution, and all 2^ℓ coins are consistent with the empty transcript, the requirement needed to use Claim 4.13 for $i = 1$ is always true. Thus by induction, with probability $1 - rd$:

1. The distribution from which γ_r is drawn and the distribution of random transcripts, $\Pi_x(U_\ell)$ are of statistical distance at most rd.
2. $1 \leq N_r \leq |Coins_x(\gamma_r)|$

We now turn to analysing the probability that in Step 3 of the protocol the verifier accepts if $\gamma_r = (\alpha_1, \beta_1, \ldots, \alpha_{r-1}, \beta_{r-1}, \rho)$ is a random transcript and if N_r is a correct lower bound. The verifier tests that $N_r = 1$, that for every $t \leq r$, α_t is the next verifier message implied by the randomness ρ and transcript up to round $t-1$. The transcript γ_r defines prefix transcripts γ_i for every $0 \leq i < r$. If γ_r is a random transcript of the protocol, then it is certainly true that for every i: $V(\gamma_{i-1}; \rho) = V(\alpha_1, \beta_1, \ldots, \alpha_{i-1}, \beta_{i-1}; \rho) = \alpha_i$. Therefore this test of the verifier will pass. To see why $N_r = 1$ recall that a transcript of the protocol ends with the verifier sending its random coins ρ. Therefore there can only be one set of coins that are consistent with a full transcript. Therefore $|Coins_x(\gamma_r)| = 1$. Since N_r is a correct lower bound of $|Coins_x(\gamma_r)|$ it must be that $N_r = 1$. Finally, the verifier tests that the original verifier accepts on transcript γ_r and randomness ρ. Since γ_r is a random transcript, by the completeness of the original protocol, the verifier will accept with probability c.

Taking into account that γ_r is not random, but only rd-close to random, and that the event that N_r is a good bound only happens with probability $1 - rd$, we have that the verifier accepts with probability at least:

$$c - 2rd = c - \left(28r\varepsilon + O\left(\left(\frac{m\ell}{\varepsilon}\right)^{23} \cdot r\delta\right)\right)$$

4.3.3 Soundness

Recall that for a transcript γ, the set $Acc_x(\gamma)$ contains the coins that are consistent with γ such that if the verifier were to use them in the original protocol, the prover would have a winning strategy. That is:

$$Acc_x(\gamma) = \{\rho | \rho \in Supp(Coins_x(\gamma)) | \exists \gamma' \text{ s.t. } V(\gamma, \gamma'; \rho) = 1 \text{ and } \rho \in Supp(Coins_x(\gamma, \gamma'))\}$$

By the soundness property of the original protocol, $|Acc_x(\emptyset)| \cdot 2^{-\ell} \leq s$. In each round of the emulation protocol, the prover helps the verifier to sample a new message. A malicious prover needs the protocol to end with the verifier holding a string in $Acc_x(\emptyset)$, and should therefore try to steer the verifier in such a way as to maximize the probability that this happens. For each round, $0 \leq i \leq r$, we define the random variable of the gap $g_i = \frac{N_i}{|Acc_x(\gamma_i)|}$. We note that $g_0 = \frac{N_0}{|Acc_x(\emptyset)|} \geq \frac{1}{s}$. In the final stage of the protocol, the verifier tests that $N_r = 1$ and will otherwise reject. Since there is only one set of coins consistent with γ_r (i.e. $|Acc_x(\gamma_r)| \leq 1$), in order for the verifier to accept it must be that $g_r = 1$. We additionally note that in order for the verifier to accept it must be that ρ_r is consistent with all of the previous partial transcripts. This implies that all the partial transcripts are consistent with each other- there exists some set of random

coins which can explain each and every step (in particular, the random coins which have made the verifier accept).

In Claim 4.15 we show that for every round, with high probability the gap will not shrink by much. Since the gap is unlikely to shrink, the prover is unlikely to be able to bring the gap all the way from $\frac{1}{s}$ to 1.

Claim 4.15. *Fix $i \in [r]$, a partial transcript γ_{i-1} and $N_{i-1} \in [2^\ell]$. Set $g_{i-1} = \frac{N_{i-1}}{|Acc_x(\gamma_{i-1})|}$. Then for every $0 < \lambda \leq 1$:*

$$\Pr\left[g_i \leq \lambda g_{i-1}\right] = O\left(\varepsilon^{-5}\ell\lambda\right)$$

where $g_i = \frac{N_i}{|Acc_x(\gamma_i)|}$ is the gap after the end of the iteration and the probability is over the randomness of the verifier in the iteration of the protocol.

Proof. Let $I = \log_{1+\varepsilon}(2^\ell)$. We would like to upper bound the probability that the prover's claim shrinks by a λ-factor relative the number of consistent accepting coins. For $1 \leq j \leq I$ define

$$S_j = \left\{ \alpha \in \{0,1\}^m \text{ s.t. } \frac{\max\left\{1, (1+\varepsilon)^{j-2}\right\}}{|Acc_x\left(\gamma_{i-1}, \alpha\right)|} \leq \lambda g_{i-1} \right\}$$

The set S_j contains all the messages which would decrease the gap by too much if subset j was chosen. Invoking the soundness property of the sampling via subsets protocol, Corollary 3.17, we have that for every $1 \leq j^* \leq I$ the probability that the protocol outputs (j^*, ρ) such that $\rho \in S_{j^*}$ is bounded from above by

$$\frac{8}{\varepsilon^3} \cdot \frac{|S_{j^*}|}{\min_{p' > \frac{\varepsilon}{6I}} f(j^*, p')} = \frac{48I}{\varepsilon^4} \cdot \frac{|S_{j^*}|(1+\varepsilon)^{i+1}}{N_{i-1}}$$

We therefore focus on finding an upper bound on $|S_j|$ for every possible value of j. Note that S_1 and S_2 are identical and so we only need to analyze values of j of size at least 2.

By definition every $\alpha \in S_j$ has $|Acc_x\left(\gamma_{i-1}, \alpha\right)| \geq \frac{(1+\varepsilon)^{j-2}}{\lambda g_{i-1}}$. Since there are at most $|Acc_x\left(\gamma_{i-1}\right)|$ accepting coins consistent with γ_{i-1}, the number of messages in S_j is at most $(1+\varepsilon)^{-(j-2)}\lambda g_{i-1}\left|Acc_x\left(\gamma_{i-1}\right)\right|$. Substituting $g_{i-1} = \frac{N_{i-1}}{|Acc_x(\gamma_{i-1})|}$ this gives us that $|S_j| \leq (1+\varepsilon)^{-(j-2)}\lambda N_{i-1}$. Therefore, if the value j is chosen, the probability that the verifier ends up with an element in S_j is bounded by

$$\frac{48I}{\varepsilon^4} \cdot \frac{|S_j|(1+\varepsilon)^{i+2}}{N_{i-1}} \leq \frac{48I(1+\varepsilon)^3}{\varepsilon^4}\lambda = O\left(\varepsilon^{-4}I\lambda\right)$$

Recalling that $I = \log_{1+\varepsilon}(2^\ell) = O(\frac{\ell}{\varepsilon})$, this value is bounded by $O\left(\varepsilon^{-5}\ell\lambda\right)$.

□

Now, using Claim 4.15 we can bound the probability that the prover manages to make the verifier accept. Since the prover must have $g_r = 1$ in order to make the verifier accept, we will bound the probability that the gap reduces from $g_0 = \frac{1}{s}$ all the way to $g_r = 1$. Using the union bound:

$$\Pr\left[\exists 1 \leq i \leq r : \ g_i \leq \lambda \cdot g_{i-1}\right] \leq O\left(\varepsilon^{-5}r\ell\lambda\right)$$

Therefore,

$$\begin{aligned}\Pr\left[g_r \leq \lambda^r \cdot g_0\right] &\leq Pr\left[\exists 1 \leq i \leq r : \ g_i \leq \lambda \cdot g_{i-1}\right] \\ &= O\left(\varepsilon^{-5}r\ell\lambda\right)\end{aligned}$$

Since $g_0 = \frac{1}{s}$, and we wish for $g_r > 1$, we can set $\lambda = \Theta(s^{\frac{1}{r}})$ and get

$$\Pr\left[g_r \leq 1\right] = O\left(\varepsilon^{-5}r\ell s^{\frac{1}{r}}\right)$$

We reiterate that if $g_r > 1$, then the verifier will reject. Therefore the verifier will accept with probability at most $O\left(\varepsilon^{-5}r\ell s^{\frac{1}{r}}\right)$.

Piecemeal Emulation Protocol

Joint Input: $x \in \{0,1\}^n$

Parameter: $0 < \varepsilon \leq \frac{1}{2}$

Prover Oracle Access: *Approx*, an ε-approximator for the protocol being emulated.

1. Let $\gamma_0 = \emptyset$, and $N_0 = 2^\ell$.
2. For $i = 1$ to $2rm$:
   (a) Prover: Let $N_i^0 \leftarrow \mathit{Approx}(\gamma_{i-1}, 0)$ and $N_i^1 \leftarrow \mathit{Approx}(\gamma_{i-1}, 1)$. If $i = 2rm$ then for every bit b such that $N_i^b > 1$, set $N_i^b = 1$. Send N_i^0 and N_i^1 to the verifier.
   (b) Verifier:
      i. Test that $\frac{N_{i-1}}{N_i^0 + N_i^1} \leq 1 + 3\varepsilon$. Reject otherwise.
      ii. Randomly choose a bit $b \in \{0,1\}$ from the Bernoulli distribution where 0 is chosen with probability $\frac{N_i^0}{N_i^0 + N_i^1}$. Send b to the prover.
   (c) Both parties set $N_i = N_i^b$ and $\gamma_i = (\gamma_{i-1}, b)$.
3. Parse γ_{2rm} into m-bit chunks $\gamma_{2rm} = \alpha_1, \beta_1, \ldots, \alpha_{r-1}, \beta_{r-1}, \rho$, where ρ is a set of random coins.
4. The verifier accepts if $0 < N_{rm} \leq 1$, $V_x(\gamma_{rm})$ accepts and if for every $j \leq r$: $\alpha_j = V_x(\alpha_1, \beta_1, \ldots, \alpha_{j-1}, \beta_{j-1}; \rho)$.

Figure 7: Piecemeal Emulation Protocol

5 Piecemeal Emulation Protocol

The piecemeal emulation protocol is built of iterations, where in each iteration the two parties add a single bit to the transcript. Each iteration starts with a partial transcript γ_{i-1} and a claim N_{i-1} on the number of coins that are consistent with γ_{i-1}, $|\mathit{Coins}_x(\gamma)|$.[8] The prover sends the verifier two values N_i^0 and N_i^1, where each one is supposed to be the number of random coins consistent with the choice of the corresponding bit. The verifier tests that the sum of the values is not too far from N_{i-1} and chooses one of the the bits based on their relative weight. The bit corresponding to the chosen value b is added to the transcript, and N_i is set to equal N_i^b. This is done until the two parties have constructed an entire transcript of the protocol. We begin by defining what type of approximation algorithm the honest prover uses, and then state the theorem.

Definition 5.1 (ε-Approximator)**.** *Let $\mathcal{L}$ be a language and $\langle P, V \rangle$ be an interactive proof for $\mathcal{L}$. An ε-approximator for $\langle P, V \rangle$, denoted Approx, is a randomized algorithm such that for every $x \in \mathcal{L}$ and partial transcript γ:*

$$\mathit{Approx}(\gamma) \in (1 \pm \varepsilon)\,|\mathit{Coins}_x(\gamma)|$$

Theorem 7. *Let $\mathcal{L}$ be a language, $\langle P, V \rangle$ be an interactive proof for $\mathcal{L}$ with r rounds, completeness c, soundness error s, message length m, verifier randomness ℓ and verifier running time t_V and let $0 \leq \varepsilon \leq \frac{1}{2}$. Suppose there exists an ε-approximator for $\langle P, V \rangle$ with running time t_A. Then there exists a public-coin interactive proof for $\mathcal{L}$ with the following properties:*

Verifier Efficiency: Running time $t_V + O(rm\ell)$, Randomness $O(rm\ell)$.

Prover Efficiency: Running time $O(rm \cdot t_A)$, Oracle Calls $O(rm)$.

[8] recall that $\mathit{Coins}_x(\gamma)$ is the set of coins that are consistent with γ, see Definition 2.9 for more details

Number Of Messages: $4rm$ *(*$2rm$ *rounds).*

Completeness: For every $x \in \mathcal{L}$ *the probability that the verifier accepts when interacting with the honest prover is at least* $c - 6rm\varepsilon$.

Soundness: For any $x \notin \mathcal{L}$ *and (unbounded) cheating prover* M^* *the probability that the verifier accepts when interacting with* M^* *on input* x *is at most* $s \cdot (1+3\varepsilon)^{2mr}$.

Remark 5.2. It is possible to have a trade-off between the number of rounds, completeness and soundness and the rest of the parameters if rather than constructing the transcript bit-by-bit it is constructed $t = O(\log n)$ bits at a time. The prover sends 2^t values in each round, and the verifier similarly to the bit-wise case tests that their sum is not too far from the previous claim. The verifier then chooses a set of t bits randomly. This would result in a public-coin protocol with $\frac{2rm}{t}$ rounds, completeness $c - \frac{6rm\varepsilon}{t}$, soundness error $s \cdot (1+3\varepsilon)^{\frac{2mr}{t}}$ prover running time $O(2^t rm \cdot t_A)$, $O(2^t rm)$ prover oracle calls and verifier running time $t_V + O(2^t rm\ell)$.

5.1 Efficiency

The protocol is dominated by $2rm$ iterations. In each iteration the verifier receives 2 strings of length $O(\ell)$. It then does a simple test, and chooses one of the strings via weighted Bernoulli distribution. The verifier, then, runs in time $O(mr\ell)$ in Step 2. In Step 4 the verifier tests consistency and tests that V accepts, which takes time t_V. Altogether the verifier runs in time $t_V + O(mr\ell)$ and uses $O(mr\ell)$ bits of randomness. The prover, in each iteration makes two calls to *Approx* takes time $2t_A$ and returns strings of length ℓ. Therefore the prover runs in time $O(mr\ell \cdot t_A)$.

In each iteration of Step 2, there is one prover message, and one verifier message. Since there are $2rm$ iterations, there are a total of $4rm$ messages sent in the protocol.

5.2 Completeness

We start our analysis by showing that if we begin iteration i of Step 2 of the protocol with γ_{i-1} and N_{i-1} such that N_{i-1} is a good approximation of $|Coins_x(\gamma_{i-1})|$, then the parties will end up with γ_i and N_i such that N_i is a good approximation of $|Coins_x(\gamma)|$:

Claim 5.3. *Fix* $x \in \mathcal{L}$ *and* $1 \le i \le 2rm$. *Let* $\gamma_{i-1} \leftarrow \langle P, V\rangle(x, U_\ell)_{1,\ldots,i-1}$ *be the first* i *bits of a random transcript of the original protocol, and* $N_{i-1} \in (1 \pm \varepsilon)\,|Coins_x(\gamma_{i-1})|$. *Then the verifier will not reject in round* i *and end up with a transcript* γ_i *and value* N_i *such that:*

- γ_i *is drawn from a distribution that is of statistical distance at most* 3ε *from* $\langle P, V\rangle(x, U_\ell)_{1,\ldots,i}$ *conditioned on the first* $i-1$ *bits equalling* γ_{i-1}.
- $N_i \in (1 \pm \varepsilon)\,|Coins_x(\gamma_i)|$.

Proof. By assumption on *Approx*, $N_i^0 \in (1 \pm \varepsilon)\,|Coins_x(\gamma_{i-1}, 0)|$ and $N_i^1 (1 \pm \varepsilon)\,|Coins_x(\gamma_{i-1}, 1)|$. Recalling that $N_{i-1} \in (1\pm)\,|Coins_x(\gamma_{i-1})|$ and since $Coins_x(\gamma_{i-1}) = Coins_x(\gamma_{i-1}, 0) \cup Coins_x(\gamma_{i-1}, 1)$, it is true that:

$$\frac{N_{i-1}}{N_i^0 + N_i^1} \le \frac{1+\varepsilon}{1-\varepsilon} \cdot \frac{|Coins_x(\gamma_{i-1})|}{|Coins_x(\gamma_{i-1}, 0)| + |Coins_x(\gamma_{i-1}, 1)|} = \frac{1+\varepsilon}{1-\varepsilon} \le 1 + 3\varepsilon$$

Therefore the verifier's test in Step 2.*b.i* will pass. Now note that whichever bit b is chosen, it is true that $N_i = N_i^b \in (1 \pm \varepsilon)\,|Coins_x(\gamma_{i-1}, b)|$.

All that remains is to show that the transcript is distributed close to a random one. Let p_0 be the probability that $\langle P, V\rangle(x, U_\ell)_{1,\ldots,i}$ ends with 0 conditioned on the first $i-1$ bits being equal to γ_{i-1}. Then:

$$p_0 = \frac{|Coins_x(\gamma_{i-1}, 0)|}{|Coins_x(\gamma_{i-1}, 0)| + |Coins_x(\gamma_{i-1}, 1)|}$$

Due to the fact that:

$$\frac{N_i^0}{N_i^0+N_i^1} \in \left(\frac{1-\varepsilon}{1+\varepsilon}, \frac{1+\varepsilon}{1-\varepsilon}\right) \cdot \frac{|Coins_x(\gamma_{i-1},0)|}{|Coins_x(\gamma_{i-1},0)|+|Coins_x(\gamma_{i-1},1)|}$$

We have that the probability that 0 is chosen is within the range $(\frac{1-\varepsilon}{1+\varepsilon}, \frac{1+\varepsilon}{1-\varepsilon})p_0$, which is contained within the $(1 \pm 3\varepsilon)p_0$. Defining p_1 similarly to p_0 but for the bit 1 we have that the probability that 1 is chosen is within the range $(1 \pm 3\varepsilon)p_1$. Then the statistical distance between the choice of bit for a real transcript and the distribution from which the bit would be chosen in the real protocol conditioned on γ_{i-1} is at most:

$$\frac{1}{2}\left|(1+3\varepsilon)p_0 - p_0 + (1+3\varepsilon)p_1 - p_1\right| < 3\varepsilon$$

The final iteration, when $i = 2rm$, is very slightly different, since the verifier may change the values N_i^0 and N_i^1 if they are larger than 1. For every bit $b' \in \{0,1\}$, $|Coins_x(\gamma_{i-1}, b'|$ is either 0 or 1. Therefore if $N_i^{b'} \in (1\pm\varepsilon)\left|Coins_x(\gamma_{i-1},b')\right|$ and $N_i^{b'} > 1$, it must be that $|Coins_x(\gamma_{i-1},b')| = 1$, and so the approximation is correct. The rest of the analysis is identical. □

We now note that in round $i = 0$, γ_0 is distributed identically to that of the original transcript as it is the empty transcript, and there are always 2^ℓ strings consistent with this transcript. Thus, using the the above claim inductively where in round i the previous transcript is $3(i-1)\varepsilon$-close to random, we get that the final transcript, after $2mr$ rounds is $6rm\varepsilon$-close to being random. All that remains is to show that the verifier's test in Step 4 will pass supposing $\gamma_{2rm} = \alpha_1, \beta_1, \ldots, \alpha_{r-1}, \beta_{r-1}, \rho$ is a random transcript. For a random transcript of the original protocol it will always be the case that for every $j \le r$: $\alpha_j = V_x\left(\alpha_1, \beta_1, \ldots, \alpha_{j-1}, \beta_{j-1}; \rho\right)$. By the completeness property of the original protocol the verifier will accept with probability at least c. Furthermore, due to the honest prover making sure in Step 1 that in the final iteration both N_{2rm}^0 and N_{2rm}^1 are bounded by 1, this test of the verifier will also pass. Thus if γ_{2rm} were a random transcript, the verifier would accept with probability at least c. Since the transcript is $6rm\varepsilon$-close to random, this event happens with probability at least $c - 6rm\varepsilon$.

5.3 Soundness

We begin by showing in Claim 5.4 that in a given round if the ratio between the number of accepting coins and the number of coins that the prover claims are consistent with the transcript transcript is small, then the probability that the verifier ends up accepting is small. Recall that $Acc_x(\gamma)$ is the set of coins for which the protocol has a legal accepting transcript (see Definition 2.10). The claim implies that the probability that the verifier accepts by the end of the protocol is at most $\frac{|Acc(\gamma_0)|}{N_0} \cdot (1+3\varepsilon)^{2rm}$. Noting that $\frac{|Acc(\gamma_0)|}{N_0} = \frac{|Acc(\emptyset)|}{2^\ell} = s$, this implies that the probability that a malicious prover convinces the verifier is at most $s \cdot (1-3\varepsilon)^{2rm}$.

Claim 5.4. *Fix $1 \le i \le 2rm$, N_{i-1} and γ_{i-1}. Then the probability that the verifier accepts is at most $\frac{|Acc(\gamma_{i-1})|}{N_{i-1}} \cdot (1+3\varepsilon)^{2rm-i}$.*

Proof. We prove the claim by reverse induction, starting with $i = 2rm$. Before beginning note that a full transcript is only accepting if it is also *consistent* with the string at the end. This means that for every partial transcript γ, $Acc(\gamma) = Acc(\gamma,0) \cup Acc(\gamma,1)$.

Basis: We analyze the case of $i = 2rm$ and show by case analysis that the probability that the verifier accepts is at most $(1+3\varepsilon) \cdot \frac{|Acc(\gamma_{i-1}|}{N_{i-1}}$. Since we are deriving the final bit of the transcript, we have that $|Acc(\gamma_{i-1})| \in \{0,1,2\}$ - i.e. the number of choices that will lead to an accepting transcript is 0, 1 or 2. We have a number of options:

1. $\frac{|Acc(\gamma_{i-1})|}{N_{i-1}} = 0$: This implies that $|Acc(\gamma_{i-1})| = 0$ and so there is no extension to to the transcript that will make the verifier accept. Then no matter what the prover does the verifier will reject.

2. $\frac{|Acc(\gamma_{i-1})|}{N_{i-1}} \geq \frac{1}{1+3\varepsilon}$: In this range of values, the claim is trivial as $(1+3\varepsilon) \cdot \frac{|Acc(\gamma_{i-1}|}{N_{i-1}} \geq 1$.

3. $0 < \frac{|Acc(\gamma_{i-1})|}{N_{i-1}} < \frac{1}{1+3\varepsilon}$: In order to cause the verifier to accept, the prover must send N_i^0 and N_i^1 that satisfy $\frac{1}{N_i^0+N_i^1} \leq \frac{1}{N_{i-1}} \cdot (1+3\varepsilon)$. There are two options for the value of $|Acc(\gamma_{i-1})|$:

   (a) $|Acc(\gamma_{i-1})| = 1$: Then there is only one bit b' for which the verifier will accept. The probability that bit b' is chosen (and therefore that the verifier accepts) is:

   $$\frac{N_i^{b'}}{N_i^0+N_i^1} \leq \frac{1}{N_i^0+N_i^1} \leq \frac{1}{N_{i-1}} \cdot (1+3\varepsilon) = \frac{|Acc(\gamma_{i-1})|}{N_{i-1}} \cdot (1+3\varepsilon)$$

   (b) $|Acc(\gamma_{i-1})| = 2$: In this case we have that $N_{i-1} > 2 \cdot (1+3\varepsilon)$. Since $\frac{N_{i-1}}{N_i^0+N_i^1} \leq 1+3\varepsilon$ this implies that $N_i^0 + N_i^1 > 2$. Recall that in order to cause the verifier to accept, it must be that $0 \leq N_{2rm} \leq 1$. Since $N_i^0 + N_i^1 > 2$, only one of the two values can be smaller or equal to than 1 simultaneously and so, as in the previous case, there is only one bit b' for which the verifier will accept. Finally, the probability that the verifier accepts is:

   $$\frac{N_i^{b'}}{N_i^0+N_i^1} \leq \frac{1}{N_i^0+N_i^1} \leq \frac{1}{N_{i-1}} \cdot (1+3\varepsilon) < \frac{|Acc(\gamma_{i-1})|}{N_{i-1}} \cdot (1+3\varepsilon)$$

Induction Step: Suppose that in iteration $i+1$, the verifier accepts with probability at most $\frac{|Acc(\gamma_i)|}{N_i} \cdot (1+3\varepsilon)^{2rm-(i+1)}$. We show that this implies that in round i the verifier accepts with probability at most $\frac{|Acc(\gamma_{i-1})|}{N_{i-1}} \cdot (1+3\varepsilon)^{2rm-i}$. The prover sends two values, N_i^0 and N_i^1. If $\frac{N_{i-1}}{N_i^0+N_i^1} > 1+3\varepsilon$ then the verifier rejects, and so this must not be the case. Suppose that the verifier chose bit b, which happens with probability $\frac{N_i^b}{N_i^0+N_i^1}$. Then by the induction hypothesis, the probability that the verifier accepts is at most $\frac{|Acc(\gamma_{i-1},b)|}{N_i^b} \cdot (1+3\varepsilon)^{2rm-(i+1)}$. Therefore the probability that the verifier chooses bit b and the prover accepts afterwards is at most:

$$\begin{aligned}\frac{N_i^b}{N_i^0+N_i^1} \cdot \frac{|Acc(\gamma_{i-1},b)|}{N_i^b} \cdot (1+3\varepsilon)^{2rm-(i+1)} &= \frac{|Acc(\gamma_{i-1},b)|}{N_i^0+N_i^1} \cdot (1+3\varepsilon)^{2rm-(i+1)} \\ &\leq \frac{|Acc(\gamma_{i-1},b)|}{N_{i-1}} \cdot (1+3\varepsilon)^{2rm-i}\end{aligned}$$

Where the final inequality is due to the fact that $\frac{N_{i-1}}{N_i^0+N_i^1} > 1+3\varepsilon$. Then, to conclude:

$$\begin{aligned}\Pr[\text{Verifier Accepts}] &= \sum_{b\in\{0,1\}} \Pr[b \text{ chosen}] \Pr[\text{Verifier Accepts when b chosen}] \\ &\leq \left(\frac{|Acc(\gamma_{i-1},0)|}{N_{i-1}} + \frac{|Acc(\gamma_{i-1},1)|}{N_{i-1}}\right) \cdot (1+3\varepsilon)^{2rm-i} \\ &= \frac{|Acc(\gamma_{i-1})|}{N_{i-1}} \cdot (1+3\varepsilon)^{2rm-i}\end{aligned}$$

□

6 Bipartiteness With Public-Coins Via the Piecemeal Protocol

In this section we give a concrete language and private-coin protocol with an efficient prover in the Interactive Proofs of Proximity (IPP) model that can be transformed using our piecemeal emulation protocol into a public-coin protocol. An ε-Proof of Proximity is similar to an interactive proof for a promise problem, except that the verifier is given limited access to the input. For further discussion and motivation on the IPP model see [RVW13]. We begin by introducing the bounded degree model:

Private-Coin Bipartiteness Protocol [RVW13]

Joint Input: A graph G on n vertices and bounded degree d given in the bounded-degree model.

1. Set $L = \Theta(\log n)$.
2. Verifier: Pick a uniformly random vertex $s \leftarrow [n]$, and perform a random walk of length L on G starting at s and ending at v. Let $b \in \{0,1\}$ be the parity of real (non "self-loop") steps in the walk. Send (s, v) to the Prover.
3. Prover: Receive (s, v). Let $p \in \{0,1\}$ be the parity of the shortest path between s and v. Send p to the Verifier.
4. Verifier: Accept if and only if $b = p$. Send all randomness used to the prover.

Figure 8: Private-Coin Bipartiteness Protocol

Definition 6.1 (The Bounded Degree Model)**.** *The input is an undirected n-vertex graph $G = ([n], E)$, of (constant) maximum degree d. The graph is represented as a function $g : [n] \times \{1, \ldots, d\} \to ([n] \cup \perp)$, which on input (v, i) outputs $u \in [n]$ if u is the i-th neighbour of v (or $\perp$ if v has fewer than i neighbours). Since the degree is bounded by d, the number of (undirected) edges is at most $d \cdot n/2$. The distance between two graphs the fraction of edges on whose presence or absence they disagree. Note that by this definition, the distance between two different graphs is at least $\frac{2}{nd}$ and at most 1.*

Definition 6.2 (Random Walk)**.** *For this section, a random walk on a graph is always as follows: In each step stay at the current vertex with probability $1/2$ and otherwise walk a random neighbour. Note that every step in the walk takes at most $\log d + 1$ bits of randomness. Let Step be the function that takes in a vertex $u \in [n]$ and $\log d + 1$ bits of randomness and outputs the next step in a walk from u.*

Definition 6.3 (Well-Mixing Graph)**.** *We say that a graph G with n nodes is well-mixing for random walks of length $L = L(n)$ if for any nodes $s, v \in [n]$, the probability that a random walk of length L starting at s reaches node v is at least $\frac{1}{2n}$.*

Consider the following promise problem $\Pi_\varepsilon = (\Pi_\varepsilon^Y, \Pi_\varepsilon^N)$ parametrised by ε due to Rothblum, Vadhan and Widgerson [RVW13]. The input is a graph G with n vertices and bounded degree $d \in \mathbb{N}$.

- Π_ε^Y: "YES" instances of the problem are bipartite graphs.
- Π_ε^N: "NO" instances are graphs that are ε-far from bipartite and are in addition well-mixing for walks of length $\log n$.

Remark 6.4. We stress that the notion of well-mixing considered in this work refers to random walks which *include self-loops* (as described in Definition 6.2). This definition of random walks is the standard one in prior work on this problem [GR99, GR02, RVW13]. Note that a bipartite graph can be "well mixing" for such walks, and as such the additional constraint that NO cases are also ε-far from bipartite is crucial (otherwise the YES and NO cases might intersect).

There exists a private-coin prover-efficient ε-Proof of Proximity for Π_ε:

Theorem 8 ([RVW13] From the proof of Theorem 5.1)**.** *For any specified $\varepsilon = \varepsilon(n)$, the protocol described in Figure 8 is a private-coin interactive proof of ε-proximity in the bounded degree model for Π_ε with the following properties:*

Verifier Efficiency: Running time $\Theta(\log n)$, Randomness $\Theta(\log n)$, Query complexity $\Theta(\log n)$.

Prover Efficiency: Running time $\mathrm{poly}(n)$.

Number Of Messages: 3.

Communication Complexity: $\Theta(\log n)$.

Completeness: For every $G \in \Pi_\varepsilon^Y$ *the probability that the verifier accepts when interacting with the honest prover is* 1.

Soundness: For any $G \in \Pi_\varepsilon^N$ *and (unbounded) cheating prover* P^* *the probability that the verifier accepts when interacting with* P^* *on input* x *is at most* $1 - \Theta(\varepsilon)$.

Now consider how to convert the protocol of Figure 8 to use public-coins with an efficient prover. Suppose we want to use Goldwasser and Sipser's protocol [GS86]. Their protocol is prover-efficient when the verifier randomness is logarithmic. Using this emulation on an r-round protocol with initial soundness error s and ℓ bits of randomness, the soundness error of the new public-coin protocol is $\Theta(\ell \cdot s^{\frac{1}{r}})$. In order for this error to be smaller than 1, it must be that $s < \ell^{-r}$ (up to constant factors). Due to this fact, we would like to shrink the soundness error of the original protocol. This can be done by repeating the protocol in parallel a number of times before emulation. Since, for prover-efficiency, we want to keep the number of verifier random coins logarithmic, we can only repeat the protocol a constant number of times. Plugging in the properties of the protocol described in Figure 8, we have that after parallel repetition, in order to keep this error under 1, it must be that $\varepsilon \geq 1 - 1/\operatorname{polylog}(n)$. We do not know whether this emulation can be used to transform the protocol to be public-coin while preserving prover efficiency when $\varepsilon < 1 - 1/\operatorname{polylog}(n)$. Nevertheless, using the *piecemeal protocol* described in Theorem 7, we transform the proof of proximity described in Figure 8 into a public-coin proof of proximity for Π_ε for any $\varepsilon = \varepsilon(n)$ with polynomial prover running time. For completeness, we describe the resulting protocol, after slight reordering and optimization for readability, in Figure 9.

We first show a general corollary of Theorem 7, showing that private-coin IPPs with logarithmic verifier randomness can always be emulated by the public-coin described in Theorem 7 without affecting completeness or the soundness error of the protocol.

Corollary 6.5 (Theorem 7 Applied on IPs with $\log n$ Randomness)**.** *Let* $\mathcal{L}$ *be a language,* $\langle P, V \rangle$ *be an interactive proof for* $\mathcal{L}$ *with* r *rounds, completeness* c*, soundness error* s*, message length* m*, verifier randomness* $O(\log n)$*, verifier query complexity* q *and verifier running time* t_V*. Then there exists a public-coin interactive proof for* $\mathcal{L}$ *with the following properties:*

Verifier Efficiency: Running time $t_V + O(rm \log n)$*, Randomness* $O(rm \log n)$*, Query complexity* q.

Prover Efficiency: Running time $O(rm \operatorname{poly}(n))$.

Number Of Messages: $4rm$ *(*$2rm$ *rounds).*

Completeness: For every $x \in \mathcal{L}$ *the probability that the verifier accepts when interacting with the honest prover is at least* c.

Soundness: For any $x \notin \mathcal{L}$ *and (unbounded) cheating prover* M^* *the probability that the verifier accepts when interacting with* M^* *on input* x *is at most* s.

Proof. We first note that the protocol described in Theorem 7 works also for interactive proofs of proximity. The verifier only queries the input at the end of the protocol when it makes sure that the transcript is legal. This is done by running the private-coin verifier on the transcript, which means that the public-coin verifier makes exactly the same number of queries to the input as the private-coin one. Since there are only $O(\log n)$ bits of verifier randomness, a 0-Approximator that runs in $\operatorname{poly}(n)$ time can be built for the transcript: Given a transcript prefix γ, enumerate over all possible choices of random coins for the verifier, and output the number of coins that lead to transcripts that have γ as their prefix. □

Remark 6.6. Notice that everything in the above proof remains identical if the prover has only oracle-access to its input. Hence, Theorem 7 works also for proofs of proximity.

Using Corollary 6.5 and Remark 6.6 on the protocol given in Theorem 8 we obtain a public-coin protocol for the "bipartiteness" problem Π_ε:

Theorem 9 (Public-Coin Protocol for Π_ε)**.** *For any specified $\varepsilon = \varepsilon(n)$, there is a* public-coin *interactive proof of ε-proximity in the bounded degree model for Π_ε with the following properties:*

Verifier Efficiency: Running time $\mathrm{polylog}(n)$*, Randomness* $\mathrm{polylog}(n)$*, Query complexity* $\Theta(\log n)$*.*

Prover Efficiency: Running time $\mathrm{poly}(n)$*.*

Number Of Messages: $\Theta(\log n)$*.*

Communication Complexity: $\mathrm{polylog}(n)$*.*

Completeness: For every $G \in \Pi_\varepsilon^Y$ the probability that the verifier accepts when interacting with the honest prover is 1*.*

Soundness: For any $G \in \Pi_\varepsilon^N$ and (unbounded) cheating prover P^* the probability that the verifier accepts when interacting with P^* on input x is at most $1 - \Theta(\varepsilon)$.

Proof. We plug the properties described in Theorem 8 into Corollary 6.5. □

Public-Coin Bipartiteness Protocol

Joint Input: A graph G on n vertices given in the bounded-degree model.

Set $L = \Theta(\log n)$.

Choose source:

1. Verifier: Choose $s \leftarrow [n]$ and send it to the Prover.

Choose target:

1. Prover: Go over all $(2d)^L$ different possible random coins. For every set of coins, follow a walk on G of length L starting at s such that in every step we to each neighbour with probability $\frac{1}{2d}$, and with the remaining probability stay at the current vertex. Let S be the set of all vertices reached during these walks. Notice that $|S| = O(\log n)$. For every $u \in S$ let N_u be the number of walks which led to a path that ended in vertex u. Send the set S and all the values N_u to the Verifier.
2. Verifier: Test that $\sum_{u \in S} N_u = (2d)^L$. If false, reject. Otherwise choose v from the distribution where $u \in S$ is chosen with probability $\frac{N_u}{\sum_{t \in S} N_t}$. Record N_v and send v to the Prover.

Parity Claim:

1. Prover: Send the parity $p \in \{0, 1\}$ of the shortest path between s and v to the Verifier.

Guided Random Walk:

1. Set $w = s$, $N = N_v$.
2. For $j = 0$ to L:
   (a) Prover: Let $\Gamma(w)$ be the neighbours of w. For every $z \in \Gamma(w) \cup \{w\}$ send N^z, the number of random coins that imply random walks leading from w to v in exactly $L - j$ steps to the verifier.
   (b) Verifier: Receive the values N_z. Test that $N = \sum_z N^z$. If false, reject. Otherwise choose a random $w' \in \Gamma(w) \cup \{w\}$ where w' is chosen with probability $\frac{N^{w'}}{\sum_z N^z}$. Send w' to the prover.
   (c) Both parties update $w = w'$ and $N = N^{w'}$.
3. Let π be the walk of length L taken in the previous step. The verifier queries the graph on the walk π. If any part of the walk is illegal, or the path does not start at s and end with t, reject. If the path is legal, then accept if $N = 1$, and the parity of the non-self-loop steps in π is equal to p. Otherwise reject.

Figure 9: Public-Coin Bipartiteness Protocol

Acknowledgements

We would like to thank Ron Rothblum for making us aware of Kilian's ideas which in turn developed into our piecemeal emulation protocol. We would also like to thank Zvika Brakerski and Moni Naor for helpful comments on the presentation of this work.

References

[Bab85] László Babai. Trading group theory for randomness. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, pages 421–429, 1985.

[BGG+88] Michael Ben-Or, Oded Goldreich, Shafi Goldwasser, Johan Hρastad, Joe Kilian, Silvio Micali, and Phillip Rogaway. Everything provable is provable in zero-knowledge. In *Advances in Cryptology - CRYPTO '88, 8th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1988, Proceedings*, pages 37–56, 1988.

[BM89] László Babai and Shlomo Moran. Proving properties of interactive proofs by a generalized counting technique. *Inf. Comput.*, 82(2):185–197, 1989.

[BRSV18] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. Proofs of work from worst-case assumptions. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, pages 789–819, 2018.

[FGM+89] Martin Fürer, Oded Goldreich, Yishay Mansour, Michael Sipser, and Stathis Zachos. On completeness and soundness in interactive proof systems. *Advances in Computing Research*, 5:429–442, 1989.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, pages 186–194, 1986.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 113–122, 2008.

[GL16] Oded Goldreich and Maya Leshkowitz. On emulating interactive proofs with public coins. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:66, 2016.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, pages 291–304, 1985.

[GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 174–187, 1986.

[Gol08] Oded Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, New York, NY, USA, 1 edition, 2008.

[Gol18] Oded Goldreich. On doubly-efficient interactive proof systems. *Foundations and Trends in Theoretical Computer Science*, 13(3):158–246, 2018.

[GR99] Oded Goldreich and Dana Ron. A sublinear bipartiteness tester for bounded degree graphs. *Comb.*, 19(3):335–373, 1999.

[GR02] Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002.

[GS86] Shafi Goldwasser and Michael Sipser. Private coins versus public coins in interactive proof systems. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, pages 59–68, 1986.

[HK13] Thomas Holenstein and Robin Künzler. A protocol for generating random elements with their probabilities. *CoRR*, abs/1312.2483, 2013.

[HMX10] Iftach Haitner, Mohammad Mahmoody, and David Xiao. A new sampling protocol and applications to basing cryptographic primitives on the hardness of NP. *Electronic Colloquium on Computational Complexity (ECCC)*, 17:1, 2010.

[IL89] Russell Impagliazzo and Michael Luby. One-way functions are essential for complexity based cryptography (extended abstract). In *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*, pages 230–235, 1989.

[ILL89] Russell Impagliazzo, Leonid A. Levin, and Michael Luby. Pseudo-random generation from one-way functions (extended abstracts). In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*, pages 12–24, 1989.

[Imp95] Russell Impagliazzo. A personal view of average-case complexity. In *Proceedings of the Tenth Annual Structure in Complexity Theory Conference, Minneapolis, Minnesota, USA, June 19-22, 1995*, pages 134–147, 1995.

[LFKN92] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992.

[LR88] Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Comput.*, 17(2):373–386, 1988.

[MP06] Silvio Micali and Rafael Pass. Local zero knowledge. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006*, pages 306–315, 2006.

[OW93] Rafail Ostrovsky and Avi Wigderson. One-way fuctions are essential for non-trivial zero-knowledge. In *Second Israel Symposium on Theory of Computing Systems, ISTCS 1993, Natanya, Israel, June 7-9, 1993, Proceedings*, pages 3–17, 1993.

[RVW13] Guy N. Rothblum, Salil P. Vadhan, and Avi Wigderson. Interactive proofs of proximity: delegating computation in sublinear time. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 793–802, 2013.

[Sha92] Adi Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, 1992.

[Sto83] Larry J. Stockmeyer. The complexity of approximate counting. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 118–126, 1983.

[Sud07] Madhu Sudan. Advanced Complexity Theory (Lecture 14): `http://people.csail.mit.edu/madhu/ST07/scribe/lect14.pdf`. Scribe notes by Nate Ince and Krzysztof Onak, 2007.

[Vad00] Salil P. Vadhan. On transformation of interactive proofs that preserve the prover's complexity. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 200–207, 2000.

[Vad12] S.P. Vadhan. *Pseudorandomness*. Foundations and Trends(r) in T. Now Publishers, 2012.

A Black-Box Transformations

In this section we show that generic black-box reductions from private-coin proofs to public-coin ones do not preserve the running time of the prover even in very restricted regimes. Black-box transformations between private-coin and public-coin proofs are defined as follows:

Definition A.1 (Black-Box Reduction From IP to AM)**.** *A black-box reduction from private-coin protocol to public-coin protocol is a pair of oracle-aided algorithms $\langle M^{P,V}, A^V \rangle$ such that for every language $\mathcal{L}$ and every interactive proof $\langle P, V \rangle$ for $\mathcal{L}$, $\langle M^{P,V}, A^V \rangle$ constitutes a public-coin IP for $\mathcal{L}$ where on input $x \in \{0,1\}^n$ M and A are given input 1^n, and oracle access to $P(x)$ and $V(x)$.*

Note that in the above definition M and A are not given direct access to the input. Vadhan [Vad00] showed that if one-way functions exist, then there is no black-box private to public coin reduction that preserves the running time of the prover. We extend this result to show that it is true for even when constraining the reduction to work on a much smaller class of private-coin proofs, namely proofs that are doubly-efficient.

Theorem 10. *Assuming the existence of one-way functions, there is no black-box transformation from doubly-efficient private-coin proofs to doubly-efficient public-coin proofs. Moreover this impossibility extends to:*

- *Transformations that result in public coin* arguments.
- *Transformations that allow the prover black-box access to the verfier-inversion function, i.e. for every x and partial transcript γ the prover has black-box access to sampling from $V_x^{-1}(\gamma)$.*

Theorem 10 states that there exist interactive proofs for which the prover is efficient, but it is still hard to transform them in a black-box manner to public-coin proofs. Doubly-efficient proofs exist only for languages in $\mathcal{BPP}$, and so this result shows that black-box transformations from private to public coin are incredibly difficult - they are hard even when restricted on languages which are easy to decide.

Organization: We begin in Appendix A.1 by giving an overview of our proof. In Appendix A.2 we define a promise problem, and in Appendix A.3 show how to construct two distributions of instances of the aforementioned promise problem for which prover efficiency-preserving reductions will be impossible. Finally, in Appendix A.4 we prove Theorem 10.

A.1 Overview of The Black-Box Impossibility Result

Recall that a doubly-efficient interactive proof is a proof with the additional requirement that the honest prover runs in polynomial time. In we show an impossibility result for transformations from private-coin doubly-efficient proofs to public-coin doubly-efficient proofs that work in a black-box manner. Our result is inspired by the ideas of Vadhan [Vad00]]. We proceed to overview the ideas behind this result, which is shown by giving a specific promise problem and a private-coin doubly-efficient interactive proof for this problem, and showing that in any black-box emulation of the proof the prover must run in super-polynomial time. d We call the promise problem used to prove our impossibility result "Weak Disjoint Support". An instance of this problem consists of three circuits, X_0, X_1 and I. "YES" instances are those such that the supports of X_0 and X_1 are disjoint, and "NO" instances are triples of circuits such that the distributions

induced by giving X_0 and X_1 a uniformly random input are identical. The circuit I acts as a "trapdoor" for this problem: For any input y, $I(y)$ returns whether or not y belongs to the support of X_1. Distinguishing between "YES" and "NO" cases is, then, easy: Set $y = X_0(0^n)$ and return that it is a "YES" instance if $I(y)$ indicates that $y \notin Supp(X_1)$. This language has a simple private-coin doubly-efficient interactive proof with completeness 1 and soundness error $\frac{1}{2}$: The verifier samples $r \leftarrow U_n$ and $b \leftarrow \{0,1\}$ and sends $X_b(r)$ to the prover. The honest prover receives y, and uses I to discover to which support y belongs, and sends this to the verifier. The verifier accepts if the prover was correct.[9]

Consider what it would mean to have black-box access to the protocol: The new prover has access to P and V and the new verifier has access to V. The function V simply receives randomness r and a bit b and returns $X_b(r)$, and so is equivalent to access to X_0 and X_1 on any input. P, given input y returns 1 if $y \in Supp(X_1)$ and 0 otherwise, and so access to P reduces to access to an oracle that returns whether an element is in the support of X_1 or not - equivalent to access to I. Suppose there exist two distributions D_Y and D_N over circuits (X_0, X_1, I) such that: (a) Circuits drawn from D_Y are "YES" instances of weak disjoint support (b) Circuits drawn from D_N are "NO" instances of weak disjoint support (c) No efficient randomized algorithm E can, given black-box access to (X_0, X_1, I) drawn from D_Y, generate $y \in Supp(X_0) \cup Supp(X_1)$ that is not the result of a previous query made by E to X_0 or X_1. (d) It is hard to distinguish between D_Y and D_N in polynomial time given black-box access only to X_0 and X_1. Using these distributions it is possible to show that there is no private-coin to public-coin transformation that preserves the prover's running time.

Suppose towards contradiction that there exists a black-box private-coin to public-coin transformation, $\langle M^{P,V}, A^V \rangle$ where M is efficient. As described above, access to V can be replaced with access to the inputs X_0 and X_1 and access to P can be replaced with oracle access to I. Thus we can treat the protocol as $\langle M^{X_0,X_1,I}, A^{X_0,X_1} \rangle$. We will show that when $(X_0, X_1, I) \leftarrow D_Y$, the prover $M^{X_0,X_1,I}$ can be replaced with an efficient prover $\widetilde{M}^{X_0,X_1}$ that does not access the circuit I, while preserving the probability that the verifier accepts when interacting with it on these inputs. Notice now two facts: (1) If (X_0, X_1, I) are drawn from D_Y then in $\left\langle \widetilde{M}^{X_0,X_1}, A^{X_0,X_1} \right\rangle$ the verifier accepts with high probability and (2) If (X_0, X_1, I) are drawn from D_N then in an execution of $\left\langle \widetilde{M}^{X_0,X_1}, A^{X_0,X_1} \right\rangle$ the verifier accepts with small probability. We defer proof of Fact (1) to later. Fact (2) follows directly from the soundness of the protocol $\langle M^{X_0,X_1,I}, A^{X_0,X_1} \rangle$ and property (b) of the distributions. Together, these two facts are sufficient to contradict property (d) of the distributions as follows: We build a distinguisher B^{X_0,X_1} that can tell whether its oracles come from D_Y or D_N. B emulates the protocol $\left\langle \widetilde{M}^{X_0,X_1}, A^{X_0,X_1} \right\rangle$ and indicates that the inputs came from D_Y if the verifier ends up accepting and that they came from D_N otherwise. Due to Fats (1) and (2), B can tell what distribution its inputs came from with non-negligible probability. Note additionally that since $\widetilde{M}$ and A are efficient so is B.

It remains to specify the prover $\widetilde{M}$ and prove Fact (1). $\widetilde{M}$ is defined as follows: Given access to oracles X_0 and X_1 and input 1^n emulate M on input 1^n. For every query q made by M to X_b, add the tuple $(X_b(q), b)$ to a list of prior queries and return $X_b(q)$ as the result to M. Given a query y to I, $\widetilde{M}$ first checks whether its list of prior queries contains (y, b) for some b. If so, it returns b and otherwise returns 0. We now show that when (X_0, X_1, I) are drawn from D_Y, $\widetilde{M}^{X_0,X_1}$ convinces the verifier with similar probability to $M^{X_0,X_1,I}$. The only difference between $\widetilde{M}$ and M is how they treat queries y to I that are in $Supp(X_0) \cup Supp(X_1)$ and are not the result of a previous call that the prover made to the circuits X_0 and X_1. Suppose towards a contradiction of property (c) of the distributions that M interacting with A can generate a query $y \in Supp(X_0) \cup Supp(X_1)$ which is not the result of a previous query M made to X_0 or X_1. Let q be a bound on the number of queries that M makes to I. We show an algorithm E that contradicts property (c) of the distributions with probability $1/q$: $E^{X_0,X_1,I}$ chooses $i \leftarrow [q]$, runs M giving it random

[9] A natural question that arises is why the verifier needs the prover, and does not simply use I directly to decide the language. While in this example it is true that the verifier gains no speedup by delegating the work to the prover, it suffices for the impossibility result. It is possible to redefine the language in a way that it takes time $\Theta(n^c)$ to solve trivially, but where the verifier in the interactive proof still runs in time $O(n)$. The proof of impossibility remains nearly identical.

coins as the messages from A and outputs the i'th query that M makes. Let us inspect the distribution of M's view when interacting with A in the protocol. Recall that the view of a party consists of its internal randomness and all messages received. Since the protocol is *public-coin*, during the interaction with the prover all that the verifier does is sample uniformly random coins and send them, and in particular makes no queries to X_0 and X_1. Therefore the view of the prover simulated by E is identical to that of the real one. Notice that whereas in the public-coin setting the view of the prover is simply random coins, in a private-coin protocol the view of the prover can depend on unknown queries to X_0 or X_1, made by the verifier (indeed, the protocol for Weak Disjoint Support described in the beginning of this section uses has property). Now, whenever M makes a query to $y \in Supp(X_0) \cup Supp(X_1)$ that has not been previously queried by it, E succeeds if it chooses i correctly, which happens with probability $1/q$, which is at least $1/\operatorname{poly}(n)$. This gives a contradiction to property (c) of the distributions. Since we have shown that whenever (X_0, X_1, I) are drawn from D_Y, M cannot generate queries to I for which $\widehat{M}$ would emulate the answers incorrectly, we conclude that, indeed, $\widehat{M}$ emulates M correctly. This, in turn, means that for such inputs, the verifier accepts with the same probability in $\left\langle \widetilde{M}^{X_0,X_1}, A^{X_0,X_1} \right\rangle$ as in $\left\langle M^{X_0,X_1,I}, A^{X_0,X_1} \right\rangle$ which, by property (a) of the distributions and by completeness, is high. This proves Fact (1) as described above.

All that remains is to show that the distributions D_Y and D_N exist. This means constructing X_0, X_1 and I. At high-level this is done by constructing a family of pseudo-random functions which are invertible (given the secret key). In the "YES" case we must make sure that X_0 and X_1 are disjoint and in the "NO" case they are essentially the same circuit with an additional randomization so that it will be hard to find two inputs that lead to the same place. Since these functions are invertible it is easy to construct I. We can make sure that property (c) of the distributions will be true with high probability since X_0 and X_1 will be expanding to a high degree, and so it will be hard for any efficient party to find a string in the image of the circuits that has non-zero probability (i.e. it will be hard to find y such that $X_0^{-1}(y)$ and $X_1^{-1}(y)$ are non-empty).

Finally, we add two observations. The first is that if the circuits X_0 and X_1 are injective, then the "trapdoor" circuit I is almost identical to finding preimages of the verifier function V. Specifically, on input y rather than returning whether it is in the support of X_1 or not, I could return $X_0^{-1}(y)$ and $X_1^{-1}(y)$. Given this alteration to I, it is possible to simulate black-box access to V^{-1} using I, and so a prover that is given access to I and V^{-1} can be simulated by one given access only to I. Together with the impossibility result described above this shows that black-box transformations are hard even when the prover is able to easily invert V. Our second observation is that the impossibility holds even if the resulting protocol is only a computationally-sound argument rather than a (statistically-sound) proof. We do this by looking closely at the proof of impossibility. To prove that B will reject instances from D_N with high probability we claimed that M given access to X_0, X_1 and the efficiently simulated I cannot convince the public-coin verifier. Since X_0 and X_1 are efficient, this hybrid prover is efficient. Therefore even if we only assume that the emulated protocol is secure against computationally bounded provers we get the same contradiction.

Remark A.2. Vadhan also extended his impossibility to apply to (honest-verifier) zero-knowledge, where the parties additionally have access to the simulator. Getting a similar statement for doubly-efficient zero-knowledge is intriguing, but the notion of zero-knowledge is not clear in this setting (see e.g. [MP06] or [BRSV18] for works in this direction), and is beyond the scope of this work.

A.2 Weak Disjoint Support

Vadhan's result shows that if one-way functions exist, then no black-box reduction exists that can convert any private-coin interactive proof to a public-coin one while preserving the running time of the prover. A weakness of his result is that the original private-coin proof for the Disjoint Support language does not seem to have an efficient prover. This raises the question of whether this result still stands if the original proof was doubly-efficient (i.e. the honest prover of the private-coin protocol runs in polynomial time for any input). We answer this question in the affirmative, and extend this impossibility to scenarios where the prover is given access to other oracles. We do this by a defining a variant of the Disjoint Support problem, which we call "Weak Disjoint Support", which has a doubly-efficient private-coin interactive proof but for which,

Interactive Proof for wDS

Joint Input: Circuits $(X_0, X_1, I) \in wDC$

Protocol:

1. Verifier: Choose $b \leftarrow \{0,1\}$ and $r \leftarrow \{0,1\}^n$ uniformly and send $y = X_b(r)$.
2. Prover: Receive y and calculate $r_0 = I(y,0)$ and $r_1 = I(y,1)$. Reply with $b' = 0$ if $X_0(r_0) = y$ and otherwise send $b' = 1$. In addition, also send r_0 and r_1.

   The values r_0 and r_1 are sent to give the black-box reduction that has oracle access to P "more power", but are not used by the verifier in the original protocol.
3. Verifier: Receive b', r_0, r_1. Accept if and only if $b' = b$.

Figure 10: Private-Coin Interactive Proof for wDS^τ

assuming one-way functions exist, there exists no black-box transformation to a public-coin interactive proof (indeed, not even a computationally sound argument).

We proceed to define the "Weak Disjoint Support" promise Recall that the disjoint support problem receives as input two circuits X_0 and X_1. In "YES" instances, the distributions implied by X_0 and X_1 are disjoint, and in "NO" instances they are identical. In weak disjoint support, we add as input another circuit, I, which is formed in such a way that I can be used to recover the preimage of any element in $Supp(X_0)$ or $Supp(X_1)$. This will allow the prover in the private-coin interactive proof to always run in polynomial time, making this a doubly-efficient interactive proof. Below is a formal definition of the promise problem:

Definition A.3 (Weak Disjoint Support)**.** *The Weak Disjoint Support problem,* $wDS = (wDS_Y, wDS_N)$ *is as follows: The input is of the form of three circuits* (X_0, X_1, I) *such that* $X_0, X_1 : \{0,1\}^n \to \{0,1\}^{3n}$ *and* $I : \{0,1\}^{3n} \times \{0,1\} \to \{0,1\}^n$ *and:*

- For the "YES" instances, wDS_Y: $Supp(X_0) \cap Supp(X_1) = \emptyset$.
- For the "NO" instances, wDS_N: $X_0(U_n) \equiv X_1(U_n)$.
- In both cases X_0 and X_1 are injective and for any $y \in \{0,1\}^{3n}$, and $b \in \{0,1\}$:

$$I(y,b) = \begin{cases} X_b^{-1}(y) & y \in Supp(X_b) \\ 0^n & o.w. \end{cases}$$

Denote by t_0, t_1 and t_I the time required to evaluate X_0, X_1 and I respectively. It is immediate that this promise problem is in $\mathcal{P}$, since it can be decided by the simple procedure of running X_0 on some arbitrary value, say 0^n, receiving some y, testing whether $y \in Supp(X_1)$ by evaluating $w = I(y,1)$, and then testing whether indeed $X_1(w) = y$. This procedure takes $O(t_{X_0} + t_{X_1} + t_I)$ time. This problem also has a doubly-efficient IP with correctness 1 and soundness error $\frac{1}{2}$, shown in Figure 10, where the prover running time is $O(t_{X_0} + t_{X_1} + t_I)$ and the verifier running time is $O(\max\{t_{X_0}, t_{X_1}\})$. Black-box access to the verifier amounts to black-box access to X_0 and X_1, and black-box access to the prover represents the following functionality:

$$P(y) = \begin{cases} \left(0, X_0^{-1}(y), X_1^{-1}(y)\right) & y \in Supp(X_0) \cap Supp(X_1) \\ \left(0, X_0^{-1}(y), 0^n\right) & y \in Supp(X_0) \setminus Supp(X_1) \\ \left(1, 0^n, X_1^{-1}(y)\right) & y \in Supp(X_1) \setminus Supp(X_0) \\ (1, 0^n, 0^n) & o.w. \end{cases}$$

A.3 Constructing the Distributions

Similarly to Vadhan, we define two efficiently sampleable distributions, D_Y^n and D_N^n. D_Y^n returns "YES" instances of wDS and D_N^n outputs "NO" instances. Recall that instances of wDS are of the form X_0, X_1, I. The distributions will be defined so that given only X_0 and X_1 as oracles, no efficient adversary will able to distinguish instances sampled from D_Y^n to ones sampled from D_N^n. We also need an additional property, that given X_0 and X_1 as oracles, it is unfeasible to find an element in the support of either circuit in polynomial time, unless it is the result of a previous query to one of the circuits. We begin by defining a family of functions, F, and then define the actual distributions using F and some additional tools.

A.3.1 Defining the Function Family F

Let $P = \{P_n\}$ be a family of strong pseudo-random permutations where for each $n \in \mathbb{N}$, all functions in P_n are keyed permutations with range $\{0,1\}^n$ and with keys drawn from $\{0,1\}^{s(n)}$. Define $F = \{F_n\}$, where every $f_k \in F_n$ is a keyed function from domain $\{0,1\}^n$ to image $\{0,1\}^{3n}$ with keys drawn from $\{0,1\}^{s(n)+3n}$ and is defined as follows:

$$f_{k,y}(x) = p_k(x \| 0^{2n}) \oplus y$$

Where p_k is a keyed PRP in P_n. That is, sampling from F_n reduces to sampling a PRP key from P_n and a uniformly random string of length $3n$. The inverse of $f_{k,y}$ is given by:

$$f_{k,y}^{-1}(t) = \begin{cases} p_k^{-1}(t_1 \oplus y_1)_{1\ldots n} & p_k^{-1}(t_1 \oplus y_1) \text{ ends with } 0^{2n} \\ \perp & o.w. \end{cases}$$

Where $p_k^{-1}(t_1 \oplus y_1)_{1\ldots n}$ are the first n bits of $p_k^{-1}(t_1 \oplus y_1)$. We now wish to show a number of properties of this function family:

Claim A.4 (Properties of the function family F)**.** *If pseudo-random permutations exist then there exists a function family $F = \{F_n\}_{n\in\mathbb{N}}$ as above such that:*

1. *Every $f_k \in F_n$ is one-to-one.*
2. *Let A be a PPTM. Then*

$$\left| \Pr_{f_k \leftarrow F_n}\left[A^{f_k, f_k^{-1}}(1^n) = 1\right] - \Pr_{g \leftarrow G_n}\left[A^{g, g^{-1}}(1^n) = 1\right] \right| = negl(n)$$

   where G_n is the set of random one-to-one functions from $\{0,1\}^n$ to $\{0,1\}^{3n}$.
3. *For every fixed $x \in \{0,1\}^n$, $f_k(x)$ is uniform in $\{0,1\}^{3n}$ over the choice of k.*
4. *For every PPTM A, the success probability of A in the following experiment is negligible in n:*
   (a) *Choose $f_k \leftarrow F_n$*
   (b) *Run $A^{f_k}(1^n)$ to obtain y.*
   (c) *A succeeds if y is in the range of f_k and A did not obtain y as a response to a query to f_k.*

Proof. We prove each item separately:

1. Every f_k is one-to-one since p_k is a permutation, and so it is easy to invert given the correct key.
2. This is proven via a hybrid. Consider some $g \in G_n$. Then there exists a permutation π' such that $\pi'(x\|0^{2n}) = g(x)$ for every x. Therefore, it is impossible to distinguish a randomly drawn function from G_n to the construction of f, if π were truly random. Since π is pseudo-random, this implies that no efficient adversary can distinguish between functions drawn from F_n and funcitons drawn from G_n.

3. Recall the definition of $f_{k,y}(x) = \pi_k(x) \oplus y$. Since y is chosen uniformly from $\{0,1\}^{3n}$, $f_k(x)$ is uniform over its entire domain.

4. We show that succeeding in this test with non-negligible probability invalidates property 2. Consider the same experiment, when given access to g, a random one-to-one function from $\{0,1\}^n$ to $\{0,1\}^{3n}$. Making only poly(n) queries, it is impossible to predict another element in the support of g. Therefore no efficient A will be able to do this for functions drawn from F_n.

□

A.3.2 The Distributions

Lemma A.5. *If one-way functions exist, then there are ensembles $\{D_Y^n\}$ and $\{D_N^n\}$ on triples of circuits such that there exists $N \in \mathbb{N}$ such that for any $n > N$:*

1. *D_Y^n and D_N^n only produce pairs (X_0, X_1, I) such that X_0, X_1 both map $\{0,1\}^n$ to $\{0,1\}^{3n}$, I is a function from $\{0,1\}^{3n} \times [n] \times \{0,1\}$ to $\{0,1\}$ and the circuits are of size at most poly(n).*
2. *$\Pr[D_N^n \in wDS_N] = 1$ and $\Pr[D_Y^n \in wDS_Y] = 1 - 2^{-n}$.*
3. *For every PPTM B:*

$$\left| \Pr_{(X_0,X_1,I) \leftarrow D_Y^n} \left[B^{X_0,X_1}(1^n) = 1 \right] - \Pr_{(X_0,X_1,I) \leftarrow D_N^n} \left[B^{X_0,X_1}(1^n) = 1 \right] \right| = negl(n)$$

4. *For every PPTM M, the probability that M succeeds in the following experiment is bounded by a negligible function in n:*
   (a) *Select $(X_0, X_1, I) \leftarrow D_Y^n$*
   (b) *Run $M^{X_0,X_1}(1^n)$ to obtain output x (note that M does not get access to I).*
   (c) *M succeeds if $x \in Supp(X_0) \cup Supp(X_1)$ and M did not obtain x as a response to a query to its oracles.*

Proof. If one-way functions exist then, by Lemmas 2.32 and 2.34, both pseudo-random functions and pseudo-random permutations exist, and so does the function family given in Claim A.4. Let $F = \{F_n\}$ be the function family guaranteed by Claim A.4 and $P = \{P_n\}$ be a family of pseudo-random permutations, where P_n are permutations over $\{0,1\}^n$. The distributions $\{D_Y^{n,\tau}\}$ and $\{D_N^{n,\tau}\}$ are defined as follows:

- $D_Y^{n,\tau}$: Select f_0 and f_1 independently from F_n. Let X_0 and X_1 be the circuits evaluating these functions. Additionally, let X_0^{-1} and X_1^{-1} be the circuits calculating the functions f_0^{-1} and f_1^{-1} respectively. Choose ψ independently from Ψ_n and set I to be the circuit that:
  - Receives inputs y, b.
  - Tests that $y \in Supp(X_b)$ using X_b^{-1}. If true returns $X_b^{-1}(y)$.
  - Otherwise returns 0^n.

  Output (X_0, X_1, I).
- $D_N^{n,\tau}$: Select f randomly from F_n and π randomly from P_n. Let X_0 be the circuit evaluating $f \circ \pi$ and let X_1 be the circuit evaluating f. Set I as in the "YES" instance. Output (X_0, X_1, I).

We now prove that all the properties required by Lemma A.5 hold. The circuit sizes evaluating these functions are bounded by poly(n) by the efficiency of P and F. Thus Property 1 holds.

For Property 2 note first that I always calculates the required function and that f and $f \circ \pi$ both induce the same distribution on the range of f, and so $\Pr[D_N^n \in wDS_N^c] = 1$. To see that $\Pr[D_Y^n \in wDS_Y] = 1 - 2^{-n}$,

note that for every $x \neq y$, by Property 3 of Claim A.4 it holds that $f_0(x) = f_1(y)$ with probability 2^{-3n}. Hence, the probability that the range of f_0 and f_1 intersect is at most $(2^n)^2 \cdot 2^{-3n} = 2^{-n}$, so Property 2 holds.

Property 3 holds from the hardness properties of f_0, f_1 and π. For the distribution D_Y^n the black-box access to X_0 and X_1 is indistinguishable from access to two random one-to-one functions from $\{0,1\}^n$ to $\{0,1\}^{3m}$. For the distribution D_N^n the black-box access to X_0 and X_1 is indistinguishable from access to a random one-two-one function g from $\{0,1\}^n$ to $\{0,1\}^{3n}$ and $g \circ \pi$ where π is a random permutation. These two cases are indistinguishable for efficient adversaries.

Property 4 follows from Property 4 of Claim A.4. The experiment is identical to that of Claim A.4, but with two independent functions from F_n. Assume towards contradiction that there does exist such adversary M. Then it would be possible to break property 4 of Claim A.4 - the adversary A receives oracle access to f, chooses a new $f' \leftarrow F_n$ uniformly, chooses uniformly which of f, f' will be X_0 and which will be X_1 and then simulates M. f and f' are drawn identically to f_0 and f_1 in D_Y^n, and so M will return some value in $Supp(f) \cup Supp(f')$. Given that it succeeded, The probability that it returns an element in f is $\frac{1}{2}$, since A chose uniformly in which location to put f and which location to put f'. □

A.4 Proving Theorem 10

Assume towards contradiction that there exists a black-box transformation $\left\langle M^{P,V,V^{-1}}, A^V \right\rangle$ from private-coin doubly-efficient proofs to doubly-efficient public-coin arguments, where the prover is additionally given access to V^{-1}. Then substituting the deIP for wDS described in Figure 10, we have that $\left\langle M^{X_0,X_1,P,V^{-1}}, A^{X_0,X_1} \right\rangle$ is a doubly-efficient public-coin argument for wDS, with completeness c and soundness s, where $c-s$ is non-negligible. We assume without loss of generality that the argument begins with a communication phase, in which all the verifier A does is toss uniformly random coins and send them to the prover, and the prover sends its replies. When the communication phase is over the verifier looks at the transcript that was generated, possibly makes calls to its oracles, and decides whether to accept or reject.

We begin by showing that the oracle for V^{-1} is superfluous when given oracle access to P. Recall that access to P gives the following functionality:

$$P(y) = \begin{cases} \left(0, X_0^{-1}(y), X_1^{-1}(y)\right) & y \in Supp(X_0) \cap Supp(X_1) \\ \left(0, X_0^{-1}(y), 0^n\right) & y \in Supp(X_0) \setminus Supp(X_1) \\ \left(1, 0^n, X_1^{-1}(y)\right) & y \in Supp(X_1) \setminus Supp(X_0) \\ (1, 0^n, 0^n) & o.w. \end{cases}$$

In order to emulate V^{-1} on input y using P the prover can do the following: Call $P(y)$ and receive result (b, x_0, x_1). The prover then tests whether $X_0(x_0) = y$ and $X_1(x_1) = y$. If the answer is yes to only one of these tests, the prover returns the answer x which passed. If both tests pass, it randomly chooses $b' \leftarrow \{0,1\}$ and returns $x_{b'}$. If neither passes the test, it returns $\perp$. Therefore any access to V^{-1} can be simply converted to access to P, and so we can without loss of generality remove the access that M has to V^{-1}.

Claim A.6. *There exists efficient oracle-aided prover $\widetilde{M}$ such that*

$$\left| \Pr_{(X_0,X_1,I) \leftarrow D_Y^n} \left[\left\langle M^{X_0,X_1,P}, A^{X_0,X_1} \right\rangle = 1 \right] - \Pr_{(X_0,X_1,I) \leftarrow D_Y^n} \left[\left\langle \widetilde{M}^{X_0,X_1}, A^{X_0,X_1} \right\rangle = 1 \right] \right| = negl(n)$$

Proof. By Lemma A.5, the probability that (X_0, X_1, I) are "YES" instances of wDS with probability $1 - negl(n)$. Conditioned on them being a "YES" instance, we have that $Supp(X_0)$ and $Supp(X_1)$ are disjoint. Consider the prover's access to P when the supports are disjoint:

$$P(y) = \begin{cases} \left(0, X_0^{-1}(y), 0^n\right) & y \in Supp(X_0) \\ \left(1, 0^n, X_1^{-1}(y)\right) & y \in Supp(X_1) \\ (1, 0^n, 0^n) & o.w. \end{cases}$$

Given access to oracles X_0 and X_1 and input 1^n emulate M on input 1^n. For every query q made by M to X_b, record the tuple $(q, X_b(q), b)$ and return $X_b(q)$ as the result to M. Given a query y to P, $\widetilde{M}$ first checks whether its list contains (q, y, b) for some q and b. If this is the case and $b = 0$, return $(0, q, 0^n)$. If it is the case and $b = 1$ return $(1, 0^n, q)$. Finally, if there is no such entry in the list, return $(1, 0^n, 0^n)$. Clearly, whenever y is the output of a previous query to X_0 or X_1, $\widetilde{M}$ gives M the same answer as P would. The same is true for any $y \notin Supp(X_0) \cup Supp(X_1)$. The final case is that M generates a query $y \in Supp(X_0) \cup Supp(X_1)$ that is not the result of a previous query to either circuit. We show in Lemma A.7 that such a query is generated by M only with negligible probability, implying that $\widetilde{M}$ emulates M correctly except with negligible probability, concluding the proof.

Lemma A.7. *When $(X_0, X_1, I) \leftarrow D^n_Y$, the prover $M^{X_0,X_1,P}$ cannot generate a query $y \in Supp(X_0) \cup Supp(X_1)$ that is not the result of a previous query to X_0 or X_1 except with negligible probability.*

Proof. Recall that we are only looking at instances drawn from D^n_Y. Suppose towards contradiction that M can generate such a query with non-negligible probability ε while making at most q queries. We show an algorithm D that contradicts Property 4 of Lemma A.5 with probability ε/q: $D^{X_0,X_1,I}$ chooses $i \leftarrow [q]$, runs M giving it random coins as the messages from A and outputs the i'th query that M makes. Let us inspect the distribution of M's view when interacting with A in the protocol. Recall that the view of a party consists of its internal randomness and all messages received. Since the protocol is *public-coin*, during the interaction with the prover all that the verifier does is sample uniformly random coins and send them, and in particular makes no queries to X_0 and X_1. Therefore view of the prover simulated by D is identical to that of the real one. Thus, whenever M makes a query to $y \in Supp(X_0) \cup Supp(X_1)$ that has not been previously queried by it, D succeeds if it chooses i correctly, which happens with probability $1/q$ which, since M is efficient is at least $1/\operatorname{poly}(n)$. Thus D breaks returns a query $y \in Supp(X_0) \cup Supp(X_1)$ which was not the answer of a previous query to X_0 or X_1 with probability at least $\varepsilon/\operatorname{poly}(n)$ which is non-negligible, breaking Property 4 of Lemma A.5. Therefore with overwhelming probability, in a real execution of M no such query is generated. □

□

Since $\widetilde{M}$ emulates M for elements drawn from wDS^n_Y, and M accepts instances drawn from this distribution with probability $c - negl(n)$:

$$\Pr_{(X_0,X_1,I)\leftarrow D^n_Y}\left[\left\langle \widetilde{M}^{X_0,X_1}, A^{X_0,X_1}\right\rangle = 1\right] = c - negl\,(n)$$

Additionally, since $\Pr\left[D^n_N \in wDS_N\right] = 1$, and because the protocol is sound against efficient provers, which includes $\widetilde{M}$:

$$\Pr_{(X_0,X_1,I)\leftarrow D^n_N}\left[\left\langle \widetilde{M}^{X_0,X_1}, A^{X_0,X_1}\right\rangle = 1\right] \leq s$$

Now consider the PPTM B^{X_0,X_1} which simply emulates the protocol $\left\langle \widetilde{M}^{X_0,X_1}, A^{X_0,X_1}\right\rangle$ using its oracles. Then

$$\Pr_{(X_0,X_1,I)\leftarrow D^n_Y}\left[B^{X_0,X_1} = 1\right] \geq c - negl\,(n)$$

and

$$\Pr_{(X_0,X_1,I)\leftarrow D^n_N}\left[B^{X_0,X_1} = 1\right] \leq s$$

Since $c - s$ is non-negligible, B breaks Property 3 of Lemma A.5. This implies that our initial assumption, that there exists doubly-efficient public-coin argument with black-box access to the prover and verifier of the deIP for wDS is false.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il