Hardness amplification of weakly verifiable puzzles

Ran Canetti*, Shai Halevi, and Michael Steiner

IBM T.J. Watson Research Center, Hawthorne, NY, USA.
canetti@watson.ibm.com, shaih@alum.mit.edu, msteiner@watson.ibm.com

Abstract. Is it harder to solve many puzzles than it is to solve just one? This question has different answers, depending on how you define puzzles. For the case of inverting one-way functions it was shown by Yao that solving many independent instances simultaneously is indeed harder than solving a single instance (cf. the transformation from weak to strong one-way functions). The known proofs of that result, however, use in an essential way the fact that for one-way functions, verifying candidate solutions to a given puzzle is easy. We extend this result to the case where solutions are efficiently verifiable *only by the party that generated the puzzle.* We call such puzzles weakly verifiable. That is, for weakly verifiable puzzles we show that if no efficient algorithm can solve a single puzzle with probability more than ε, then no efficient algorithm can solve n independent puzzles simultaneously with probability more than ε^n. We also demonstrate that when the puzzles are not even weakly verifiable, solving many puzzles may be no harder than solving a single one.

Hardness amplification of weakly verifiable puzzles turns out to be closely related to the reduction of soundness error under parallel repetition in computationally sound arguments. Indeed, the proof of Bellare, Impagliazzo and Naor that parallel repetition reduces soundness error in three-round argument systems implies a result similar to our first result, albeit with considerably worse parameters. Also, our second result is an adaptation of their proof that parallel repetition of four-round systems may not reduce the soundness error.

1 Introduction

This work is concerned with the fundamental question of hardness amplification via parallel repetition. Suppose we knew that no efficient device succeeds in some computational task with probability much better than ε. Then what can we say about the success probability of efficient devices in performing n such tasks in parallel? The answer clearly depends on the type of task at hand. For the simple case where the task is inverting an efficiently computable function, the answer implicit in the groundbreaking work of Yao [13] is that the success probability cannot be much more than ε^n. A proof can be found in Goldreich's book [6,

* Supported by NSF CyberTrust Grant #0430450.

Chapter 2.3]. However, this proof relies heavily on the ability to efficiently verify the correctness of a candidate solution: That is, on input y in the range of the function f and candidate x from its domain, it is possible to efficiently verify that indeed $y = f(x)$. A natural question is whether parallel repetition amplifies hardness also for other types of puzzles, and in particular for the case where the entity posed with the puzzle cannot efficiently verify on its own the correctness of candidate solutions. (Following [12, 8], we use the term "puzzles" to denote somewhat-hard automatically-generated computational problems.)

We identify a more general class of puzzles for which parallel repetition indeed amplifies hardness. Specifically, we show that the same hardness amplification result holds even in the case where only the entity generating the puzzles can efficiently verify correctness of candidate solutions. More precisely, we consider the case where puzzles are generated (by some efficient algorithm) together with some "secret check information". Efficient verification of correctness of candidate solutions for a puzzle is guaranteed only if the corresponding secret check information is known. In particular, the entity posed with the puzzle may not be able to efficiently verify correctness of candidate solutions. We call such puzzles weakly verifiable. We show that, even in this setting, if no efficient algorithm can solve a single puzzle with probability much more than ε, then no efficient algorithm can simultaneously solve n puzzles with probability much more than ε^n, which is essentially optimal.[1] We also show that the weak verifiability property is essential for obtaining such a general hardness amplification result: We exhibit an example of puzzles that are not even weakly verifiable, and where the probability of solving multiple instances is the same as the probability of solving a single instance.

One example of weakly verifiable puzzles is the notion of computer-generated inverse Turing tests, or CAPTCHAs [11, 1]. These are distribution of puzzles that are easily solvable by humans, but are assumed to be solvable by computers only with small (albeit noticeable) probability. Automatically verifying a solution to a given CAPTCHA is typically just as hard as solving it, since the space of solutions is fairly small. Still, CAPTCHAs are weakly verifiable, as it is possible to efficiently generate a CAPTCHA together with its unique solution. In the work of von-Ahn et al. [1], they suggests *sequential* repetition as a method of hardness amplification for CAPTCHAs. Our work indicates that parallel repetition can be used as well. We note that puzzles that are only weakly verifiable can be constructed from one-way functions (e.g., the puzzle is $f(x)$ and the solution is a hard-core bit of x). On the other hand, our definition in Section 2 does not imply one-way functions. See Section 2.2 for more discussion on the definition of puzzles and relations to other notions of computational hardness.

1.1 Soundness amplification of argument systems

Bellare, Impagliazzo, and Naor [2] investigated the problem of reducing the soundness error of interactive argument systems (i.e., proof systems with compu-

[1] As usual, the analysis incurs slackness of negligible quantities. This in particular means that the amplification is only meaningful when ε is not negligible.

tational soundness). They showed that for three-round systems, n-fold parallel repetition reduces the soundness error exponentially in n, whereas for four-round systems parallel repetition may not reduce the soundness error at all. This problem turns out to be closely related to ours: For three-round systems, once the first prover message is fixed, the remaining two messages can be regarded as a weakly verifiable puzzle sent by the verifier to the prover, followed by a solution candidate sent by the prover. When there are more than three rounds, this puzzle may not be even weakly verifiable without additional communication. Indeed, the result in [2] for three-round argument systems implies that parallel repetition of weakly verifiable puzzles reduces the success probability exponentially. Similarly, their example of a four-round system whose soundness error is not reduced by parallel repetition can be translated to a family of puzzles (that are not weakly verifiable) where parallel repetition does not amplify hardness.

In terms of concrete parameters, however, the result in [2] is far from optimal. Specifically, they show that if no algorithm can solve a single puzzle much better than ε, then no algorithm can solve n independent puzzles simultaneously with probability much better than δ^n, where $\delta \approx \exp\left(\frac{-(1-\varepsilon)^2}{128}\right)$. Note that δ is quite close to one: we always have $\delta > \exp(-1/128) > 0.99$, regardless of ε. In particular, if ε is small (say, $\varepsilon = 1/\text{poly}$), then it may take many repetitions before any amplification whatsoever is guaranteed. Hence, although the bound in [2] suffices for an asymptotic result, it is not very useful for the case of amplifying hardness of moderately hard weakly verifiable puzzles. We remark that our improved bounds apply also to the problem of parallel repetition of argument systems. Also there, we obtain optimal bounds.

1.2 Our techniques

To show hardness amplification, we need to transform an algorithm A that solves n puzzles with probability ε^n into another algorithm A' that solves a single puzzle with probability ε. We consider the following matrix M that represents n-vectors of puzzles: The columns are labeled by all the possibilities for the first puzzle, and rows are labeled by all the possibilities for puzzles $2..n$, so each entry in the matrix corresponds to a particular n-vector of puzzles. Each entry consists of two bits, where the first bit is 1 if the answer-vector that A returns for these n puzzles includes a correct solution to the first puzzle, and the second bit is 1 if it includes correct solutions to all the puzzles $2..n$.

We make the following combinatorial observation. Assume that the fraction of (1,1) entries in M is at least some positive γ, and let α, β be positive numbers such that $\gamma = \alpha \cdot \beta$. Then, either M has some column with α-fraction of entries of the form $(\star, 1)$, or else the conditional probability of a (1,1) entry given that the entry is of the form $(\star, 1)$ is at least β.

We use this combinatorial observation with $\alpha = \varepsilon^{n-1}$, $\beta = \varepsilon$ and $\gamma = \alpha \cdot \beta = \varepsilon^n$, and indeed we know that the fraction of $(1, 1)$ entries in the matrix is at least ε^n. We thus conclude that either there exists a particular puzzle x, such that when x is the first puzzle, A correctly solves all the other puzzles with probability ε^{n-1}, or the conditional probability of a correct answer to the first puzzle when the answers to all the others are correct is at least ε. In the former case we trivially get an algorithm that solves $n-1$ puzzles with probability ε^{n-1} and we can continue by induction. In the latter case we directly get an algorithm that solves a single puzzle with probability ε: Simply choose many random $(n-1)$-vectors of puzzles, insert the input puzzle at the beginning to form an n-vector, and run A on the resulting vector. Repeat this process until we get correct answers to the last $(n-1)$ puzzles. Then we guess that we also have the right answer to the input puzzle and output that answer. The heart of the analysis is proving that this strategy indeed yields success probability (close to) ε.

Relations to xor-lemma proofs. We comment that there is some parallel between proofs of hardness amplification and proofs of xor lemmas, and indeed our proof is reminiscent of the xor-lemma proof of Myers [10]. In particular, he too has two asymmetric cases, where in one case we only get dimension reduction by one but it is essentially "for free", and in the other case we directly go to dimension one but pay some polynomial factor in complexity.

2 Notations and definitions

Below we use the term *efficient algorithms* as a synonym to (probabilistic) polynomial-time Turing machines. A function (from positive integers to positive real numbers) is *negligible* if it approaches zero faster than any inverse polynomial. (Also, we informally say that something is *noticeable* when it is larger than some inverse polynomial.) We use $\mathsf{negl}(\cdot)$ to denote an unspecified negligible function. We also use the notation $\tilde{O}(x)$ as a shorthand for $O(x \log^c x)$ for some constant c.

2.1 Puzzles

A system for weakly verifiable puzzles consists of algorithms for generating random puzzles and for verifying solutions to these puzzles. Specifically, it consists of a pair of efficient algorithms $\mathcal{Z} = (G, V)$, such that

- The puzzle-generator algorithm G, on security parameter k, outputs a random puzzle p along with some "check information" c, $(p, c) \xleftarrow{\$} G(1^k)$.
- The "puzzle verifier" V is a deterministic efficient algorithm that on input a puzzle p, check-information c, and answer a, outputs either zero or one, $V(p, c, a) \in \{0, 1\}$.

A *solver* for this puzzle system is an efficient algorithm S that gets a puzzle p as input and outputs an answer a. The *success probability* of S is the probability that the answer is accepted by the puzzle verifier,

$$\mathsf{succ}_{\mathcal{Z}}[S] \stackrel{\text{def}}{=} \Pr_{G,S}\left[(p, c) \xleftarrow{\$} G(1^k),\ a \xleftarrow{\$} S(p)\ :\ V(p, c, a) = 1\right]$$

where the probability is taken over the randomness of G and S. (Note that $\mathsf{succ}_{\mathcal{Z}}[S]$ is a function of the security parameter k.) The *hardness* of the puzzle system $\mathcal{Z}$ is a bound on the success probability of any efficient solver.

Definition 1 (Hardness of puzzles). *Let* $\varepsilon : \mathbb{N} \to [0,1]$ *be an arbitrary function. A puzzle-system* $\mathcal{Z}$ *is said to be* $(1-\varepsilon)$*-hard if for any efficient solver* S*, there is a negligible function* negl *such that* $\mathsf{succ}_{\mathcal{Z}}[S] \leq \varepsilon + \mathsf{negl}$.

Repetition. Let $\mathcal{Z} = (G, V)$ be a puzzle system, and let $n : \mathbb{N} \to \mathbb{N}$ be an arbitrary function. We denote by G^n the algorithm that on security parameter k runs $G(1^k)$ for $n(k)$ times and outputs all the n puzzles with their check information, $(\langle p_1, \ldots, p_n \rangle, \langle c_1, \ldots, c_n \rangle) \xleftarrow{\$} G^n(1^k)$. Similarly, we denote by V^n the function that gets three n-vectors $\boldsymbol{p}, \boldsymbol{c}, \boldsymbol{a}$ and outputs one if and only if $V(p_i, c_i, a_i) = 1$ for all $i \in \{1, ..., n\}$. The n-fold repetition of $\mathcal{Z}$ is the puzzle system $\mathcal{Z}^n = (G^n, V^n)$.

2.2 Discussion

We discuss some aspects of the definition of puzzles, and relate it to other notions of computational hardness. First, note that our definition of weakly-verifiable puzzles allows the veracity of answer a to puzzle p to depend on the check-information c. Namely, we allow the possibility of two outputs (p, c), and (p, c') in the support of G (with the same p but different c's), such that for some answer a it holds that $V(p, c, a) = 1$ but $V(p, c', a) = 0$. This extra generality may seems somewhat non-intuitive at first. In particular, it allows the hardness of solving the puzzle to be information-theoretic. (For example, consider a system where the puzzle is always the all-zero string, the check information is a random k-bit string, and an answer is accepted if it equals the check information.) We chose this more general formulation since defining things this way is slightly simpler, and because it captures also the soundness in proof systems where soundness is argued unconditionally. Also, it makes our result a bit stronger (since it works even for this wider class of puzzles).

Still, the interesting cases for hardness amplification are usually the ones where the veracity of the solution does not depend on the check information. Notice that in such systems the hardness can only be computational. (Indeed, an infinitely powerful solver, on input puzzle p, can exhaustively search for a pair (c, a) where (p, c) is in the support of G and $V(p, c, a) = 1$.) Below we therefore call such systems weakly-verifiable *computational* puzzle systems.

It may be instructive to relate the notion of weakly-verifiable computational puzzles to the notion of average-case hardness due to Levin [9]. Recall that according to Levin, a *distributional problem* is a pair $(\mathcal{P}, \mathcal{D})$, where $\mathcal{P}$ is a (search or decision) problem and $\mathcal{D}$ is a distribution on the instances of $\mathcal{P}$. Hence, weakly-verifiable puzzles are a special case of distributional search problems. Most of the literature concerning Levin's theory is focused on the study of the case where $\mathcal{P}$ is an NP-problem (either search or decision). From the perspective of the current work, this means that candidate solutions are always efficiently verifiable. Hence the previous proofs of Yao's theorem can be used just as well to prove hardness amplification for all these prior notions.

In contrast, in this work we consider search problems that *are not even in NP*. When cast as distributional search problems, a weakly-verifiable computational puzzle system is a distributional search problem $(\mathcal{P}, \mathcal{D})$ where $\mathcal{D}$ is efficiently sampleable and the relation $R_{\mathcal{P}} = \{(p, a) : a \text{ is a correct solution for } p\}$, is *not necessarily efficiently computable.* (When viewed as a language, $R_{\mathcal{P}}$ is itself in NP, with the witness roughly being the check information c.)[2] Hence, the class of distributional search problems that result from weakly-verifiable computational puzzle systems is a superset of the class $\langle \mathrm{NP}, \text{P-sampleable} \rangle$ of Ben-David et al. [3], in which $R_{\mathcal{P}}$ in an NP-relation (where pairs can be recognized in polynomial time).

The class $\langle \mathrm{NP}, \text{P-sampleable} \rangle$ is itself a superset of the class DistNP from Levin's work [9], in which the distribution $\mathcal{D}$ is P-computable. However, it was shown by Impagliazzo and Levin [7] that if $\langle \mathrm{NP}, \text{P-sampleable} \rangle$ contains hard problems, then so does DistNP. It is also easy to see that a hard weakly-verifiable computational puzzle can be transformed into a hard problem in $\langle \mathrm{NP}, \text{P-sampleable} \rangle$ (by changing the goal of the search problem from finding a to finding (a, c)). Hence, if any of these classes contains hard problems, then they all do. We finally comment that the existence of hard problems in these classes is not known to imply the existence of one-way functions.

3 Hardness amplification of puzzles

Theorem 1. *Let $\varepsilon : \mathbb{N} \to [0, 1]$ be an efficiently computable function, let $n : \mathbb{N} \to \mathbb{N}$ be efficiently computable and polynomially bounded, and let $\mathcal{Z} = (G, V)$ be a weakly verifiable puzzle system. If $\mathcal{Z}$ is $(1 - \varepsilon)$-hard, then $\mathcal{Z}^n$, the n-fold repetition of $\mathcal{Z}$, is $(1 - \varepsilon^n)$-hard.*

The core of the proof is a transformation that turns an algorithm A that solves (G^n, V^n) with probability δ^n (for some δ) into an algorithm A' that solves (G, V) with probability $\delta(1 - \frac{1}{q})$, where q is some "slackness parameter".[3] The running time of A' is polynomial in n, q, $1/\delta^n$, and the running times of A, G, and V.

Lemma 1. *Fix efficiently computable functions, $n, q : \mathbb{N} \to \mathbb{N}$, and $\delta : \mathbb{N} \to (0, 1)$. Also fix a puzzle system $\mathcal{Z} = (G, V)$, and denote the running times of G, V, by T_G, T_V, respectively. If there exists a solver A for $\mathcal{Z}^n$ with success probability δ^n and running time T, then there exists also a solver A' for $\mathcal{Z}$ with success probability $\delta(1 - \frac{1}{q})$ and running time $T' = \tilde{O}\left(\frac{nq^3}{\delta^{2n-1}}(T + nT_G + nT_V)\right)$.*

[2] Technically, the witness has to be randomness of the generator when generating (p, c). The reason is that even a computational puzzle system can have "invalid check information" c' such that $V(p, c', a) = 1$ for an incorrect answer a, as long as the generator G has probability zero of outputting (p, c').

[3] The parameter q is introduced in order to achieve an "optimal" hardness amplification result, from ε to ε^n. One can instead set, say, $q = 2$, in which case you can only prove amplification from ε to $(2\varepsilon)^n$.

3.1 The solver A'

Having input puzzle p the algorithm A' consists of two phases. Roughly, in a pre-processing phase, A' tries to find a puzzle p_1^*, such that when p_1^* is placed as the first puzzle in a vector, the algorithm A correctly solves all the other puzzles with probability at least δ^{n-1}. This pre-processing is the most time-consuming operation in the execution of A'. If such p_1^* is found, then A' makes a "recursive call to itself", using as a solver for $\mathcal{Z}^{n-1}$ the algorithm A with p_1^* hard-wired as the first puzzle.

If A' fails to find such puzzle p_1^*, it moves to the on-line phase, where it actually tries to solve its input puzzle p. This is done by repeatedly sampling $(n-1)$-vectors $(\boldsymbol{p}, \boldsymbol{c}) \leftarrow G^{n-1}(1^k)$, and running A on the n-vector $(p, \boldsymbol{p})$, getting the n-vector of answers $(a, \boldsymbol{a}) \leftarrow A(p, \boldsymbol{p})$. If the answers $2..n$ are correct (i.e., $V^{n-1}(\boldsymbol{p}, \boldsymbol{c}, \boldsymbol{a}) = 1$) then A' "hopes that the solution to p is also valid", and outputs the first answer a. (If too many trials have passed without getting a correct answers for puzzles $2..n$, then A' aborts.) A detailed description of A' follows, and the code for A' can be found in Figure 1.

Pre-processing phase: In the pre-processing phase, A' tries to find a prefix of $v \leq n-1$ puzzles that has high probability of residual success. That is, conditioned on this prefix, A solves the suffix of $n-v$ puzzles with probability at least δ^{n-v}. The pre-processing phase consists of iterations, where in the iteration i, A' already has a prefix of $i-1$ puzzles and it tries to add to it the i'th puzzle. This is done in a straightforward manner: let prefix be the prefix of length $i-1$ from the previous iteration. A' repeatedly chooses candidates to extend the prefix, and for each candidate p^* it estimates the residual-success probability of $\mathsf{prefix} \circ p^*$. (I.e., the probability that A, on input (prefix, p^*, $n-i$ random puzzles), solves correctly the last $n-i$ puzzles.) If A' finds a candidate p^* for which the estimated probability is at least δ^{n-i}, then it adds it to the prefix and continues to the next iteration. We stress that since A' generates these last "$n-i$ random puzzles" by itself, it also has the corresponding check information so it can verify the solutions to these puzzles.

In iteration i, A' tries at most $N_i = \left\lceil \frac{6q}{\delta^{n-i+1}} \ln\left(\frac{18qn}{\delta}\right) \right\rceil$ candidates. If none of them yields estimated probability of δ^{n-i} then A' terminates the pre-processing and moves to the on-line phase. For each candidate, A' estimates the probability up to additive accuracy of $\delta^{n-i}/6q$ with confidence of $\delta/18qnN_i$. Namely, $\Pr[|\mathsf{estimated} - \mathsf{actual}| > \delta^{n-i}/6q] < 2\delta/18qnN_i$. Using Chernoff bound, one can see that it is sufficient to sample $M_i = O(\frac{q^2}{\delta^{n-i}} \ln(\frac{qn}{\delta}))$ points to get these accuracy and confidence bounds.

On-line phase: Going into the on-line phase, A' has an input puzzle p, and a prefix of $v \leq n-1$ puzzles, and we know the residual-success probability of that prefix was estimated to be at least δ^{n-v}. Due to the accuracy and confidence bounds that were used in the estimation above, we can assume that the actual probability is at least $\delta^{n-v}(1-1/6q)$ (and this assumption holds expect with very small probability).

```
SOLVER A'(p):                                   // Parameters: k, n, q, δ
PREPROCESSING PHASE:
    initialize prefix ← empty-vector
    for i = 1 to n − 1
        p* ← EXTEND-PREFIX(prefix, i)
        if p* =⊥ then v ← i, goto ONLINE PHASE
        else prefix ← prefix ∘ p*
    v ← n, goto ONLINE PHASE

ONLINE PHASE:
    if v = n                                    // Base case, prefix has n − 1 puzzles
        a ← A(prefix, p)
        return a_n
    else                                        // prefix has v − 1 ≤ n − 2 puzzles
        repeat ⌈6q ln(6q) / δ^{n−v+1}⌉ times:
            (⟨p_{v+1}, ..., p_n⟩, ⟨c_{v+1}, ..., c_n⟩) ← G^{n−v}(1^k)
            a ← A(prefix, r, p_{v+1}, ..., p_n)
            if V(p_i, c_i, a_i) = 1 for all i ∈ {v + 1, ..., n} then return a_v
        if none of the repetitions succeeded then abort

EXTEND-PREFIX(prefix, i):                       // prefix has i − 1 puzzles
    N_i ← ⌈(6q / δ^{n−i+1}) ln(18qn / δ)⌉
    repeat N_i times:
        (p*, c*) ← G(1^k)
        μ̄_{p*} ← ESTIMATE-RES-SUCC-PROB(prefix ∘ p*, i)
        if μ̄_{p*} ≥ δ^{n−i} then return p*
    return ⊥                                    // No good extension found

ESTIMATE-RES-SUCC-PROB(prefix, i):              // prefix has i puzzles
    M_i ← ⌈(84q^2 / δ^{n−i}) ln(18qn·N_i / δ)⌉
    repeat M_i times:
        (⟨p_{i+1}, ..., p_n⟩, ⟨c_{i+1}, ..., c_n⟩) ← G^{n−i}(1^k)
        a ← A(prefix, p_{i+1}, ..., p_n)
        this sample is successful if V(p_j, c_j, a_j) = 1 for all j ∈ {i + 1, ..., n}.
    return number-of-successes/M_i
```

Fig. 1. The solver A' for $\mathcal{Z}$

If the prefix is of length $n-1$, then we know that A solves any single puzzle with probability at least $\delta(1-1/6q)$, so A' directly uses it to solve the input puzzle p. Otherwise, we know that the last iteration of the pre-processing phase failed to find an extension to the prefix that has estimated residual-success probability of δ^{n-v}. As we will show later, this means that with the given prefix, the conditional probability that A solves the v'th puzzle *given that it solves puzzles* $v+1,\dots,n$ is very close to δ. Thus, A' samples many random vectors of $n-v$ puzzles (with their check information) and use $A(\mathsf{prefix},\ p,\ \text{random puzzles})$ to try and solve the random puzzles. Since the overall residual-success probability with the given prefix is close to δ^{n-v+1} we expect to succeed after not much more than $1/\delta^{n-v+1}$ trials (but for technical reasons A' tries as many as $6q\ln(6q)/\delta^{n-v+1}$ times). Once A' gets an answer vector $\boldsymbol{a}$ that contains correct solutions to the random puzzles $v+1,\dots,n$, it outputs the answer a_v (i.e., the one that corresponds to the input puzzle). If all the trials fail, then A' aborts.

3.2 Analysis of A'

In the analysis we refer directly to the code of A' from Figure 1. We begin with the running time of A'. The pre-processing phase consists of at most $n-1$ calls to Extend-prefix, where the i'th call makes at most $N_i = \left\lceil \frac{6q}{\delta^{n-i+1}} \ln\left(\frac{18qn}{\delta}\right)\right\rceil$ calls to Estimate-res-succ-prob. The routine Estimate-res-succ-prob (when called during the i'th iteration) goes through its loop for $M_i = \left\lceil \frac{84q^2}{\delta^{n-i}} \ln\left(\frac{18qnN_i}{\delta}\right)\right\rceil$ times, and each loop makes one call to A and $n-i<n$ calls to G and V. Thus, the total time of the pre-processing phase is less than

$$n\cdot\frac{6q}{\delta^n}\cdot O\left(\ln(\frac{qn}{\delta})\right)\cdot\frac{84q^2}{\delta^{n-1}}\cdot O\left(\ln(\frac{qn}{\delta})\right)\cdot(T+nT_G+nT_V) \;=\; \tilde{O}\left(\frac{nq^3}{\delta^{2n-1}}(T+nT_G+nT_V)\right)$$

The on-line phase of A consists of at most $6q\ln(6q)/\delta^{n-v+1} \le 6q\ln(6q)/\delta^n$ repetitions of a loop, and each repetition makes one call to A and $n-v<n$ calls to G and V, so the time of this phase is $O(\frac{q\ln q}{\delta^n}(T+nT_G+nT_V))$. The total running time is therefore $\tilde{O}\left(\frac{nq^3}{\delta^{2n-1}}(T+nT_G+nT_V)\right)$, as stated in Lemma 1.

Next we analyze the success probability, and we begin with a few notations. For a vector $\boldsymbol{p}$ with $i\le n-1$ puzzles, we denote the residual-success probability of $\boldsymbol{p}$ by

$$\mathsf{rsp}_i[\boldsymbol{p}] \stackrel{\text{def}}{=} \Pr\left[\begin{array}{r}(\langle p_{i+1},\dots,p_n\rangle,\langle c_{i+1},\dots,c_n\rangle) \stackrel{\$}{\leftarrow} G^{n-i}(1^k),\boldsymbol{a} \stackrel{\$}{\leftarrow} A(\boldsymbol{p},\ p_{i+1},\dots,p_n)\\ : V(p_j,c_j,a_j)=1 \text{ for all } i+1\le j\le n\end{array}\right].$$

(In this notation, it is assumed that the security parameter k is implicit in the puzzles in $\boldsymbol{p}$.) For a vector $\boldsymbol{p}$ with $i-1\le n-2$ puzzles, we denote by $\mathsf{Ext}_i(\boldsymbol{p})$ the set of "good extensions" of $\boldsymbol{p}$, namely those puzzles p^* such that the residual-success probability of $\boldsymbol{p}\circ p^*$ is noticeably more than δ^{n-i},

$$\mathsf{Ext}_i(\boldsymbol{p}) \stackrel{\text{def}}{=} \{\ p^*:\ \mathsf{rsp}_i(\boldsymbol{p}\circ p^*) \ge \delta^{n-i}(1+\frac{1}{6q})\ \}$$

Next, we let prefix_i be the random variable describing the prefix of length i after the i'th iteration in the pre-processing phase, if there is one (otherwise we let $\mathsf{prefix}_i = \perp$). By convention, $\mathsf{prefix}_0 = \Lambda \neq \perp$.

We say that iteration i in the pre-processing phase *makes the wrong decision*, either if it returns an extension to prefix_{i-1} with residual-success probability that is too low, or if it fails to find an extension even though many good extensions exist. Formally, the event Wrong_i is defined when $\mathsf{prefix}_{i-1} \neq \perp$, and one of the following holds:

(a) either $\mathsf{prefix}_i \neq \perp$ (i.e., the routine $\textsc{Extend-prefix}(\mathsf{prefix}_{i-1}, i)$ returned some $p^* \neq \perp$), but $\mathsf{rsp}(\mathsf{prefix}_i) < \delta^{n-i}(1 - \frac{1}{6q})$,

(b) or $\mathsf{prefix}_i = \perp$ (i.e., the routine $\textsc{Extend-prefix}(\mathsf{prefix}_{i-1}, i)$ returned $\perp$), but it holds that $\Pr[(p, c) \xleftarrow{\$} G(1^k) \ : \ p \in \mathsf{Ext}_i(\mathsf{prefix}_{i-1})] \geq \frac{\delta^{n-i+1}}{6q}$.

Claim. For all $i \leq n-1$, $\Pr[\mathsf{Wrong}_i] \leq \frac{\delta}{6qn}$.

Proof. This claim essentially follows from the Chernoff bound. Recall that the Chernoff bound asserts that for any 0-1 random variable with mean μ, is we choose M independent samples of that variable and let $\bar{\mu}$ be their average, then for any $\gamma > 0$ we have

$$\Pr[\mu - \bar{\mu} > \gamma] < \exp\left(\frac{-M\gamma^2}{2\mu(1-\mu)}\right) < \exp\left(\frac{-M\gamma^2}{2\mu}\right)$$

(and the same expression also bounds $\Pr[\bar{\mu} - \mu > \gamma]$). Fix some i and assume that $\mathsf{prefix}_{i-1} \neq \perp$, which means that the pre-processing phase indeed calls $\textsc{Extend-prefix}(\mathsf{prefix}_{i-1}, i)$. For any possible value of prefix_{i-1}, we now bound the probability of Wrong_i conditioned on this value of prefix_{i-1}. For each puzzle p^* that can be chosen by $\textsc{Extend-prefix}$ in line 22, let $\mu_{p^*} = \mathsf{rsp}_i(\mathsf{prefix}_{i-1} \circ p^*)$. Note that the estimation routine uses $M_i = \left\lceil \frac{84q^2}{\delta^{n-i}} \ln\left(\frac{18qnN_i}{\delta}\right)\right\rceil$ samples to provide an estimate $\bar{\mu}_{p^*}$ for μ_{p^*}. Using the Chernoff bound, we have for any puzzle p^* such that $\mu_{p^*} \leq \delta^{n-i}(1 - \frac{1}{6q})$

$$\begin{aligned}\Pr\left[\bar{\mu}_{p^*} \geq \delta^{n-i}\right] \leq \Pr\left[\bar{\mu}_{p^*} - \mu_{p^*} > \delta^{n-i}/6q\right] \quad &< \quad \exp\left(-M_i \cdot \frac{\delta^{2(n-i)}}{72q^2\mu_{p^*}}\right) \\ < \exp\left(-M_i \cdot \frac{\delta^{n-i}}{72q^2}\right) \quad < \quad \exp\left(-\ln\left(\frac{18qnN_i}{\delta}\right)\right) \quad &= \quad \frac{\delta}{18qnN_i}\end{aligned}$$

and since the routine $\textsc{Extend-prefix}$ examines at most N_i candidates p^*, the probability that it gets an estimate $\bar{\mu}_{p^*} \geq \delta^{n-i}$ for any candidate p^* with $\mu_{p^*} \leq \delta^{n-i}(1 - 1/6q)$ is at most $\delta/18qn$. Hence the probability of sub-case (a) is at most $\delta/18qn$. Similarly, for a puzzle p^* such that μ_{p^*} exactly equals $\delta^{n-i}(1 + \frac{1}{6q})$ we have

$$\begin{aligned}&\Pr\left[\bar{\mu}_{p^*} < \delta^{n-i}\right] \leq \ \Pr\left[\bar{\mu}_{p^*} - \mu_{p^*} < -\delta^{n-i}/6q\right] < \ \exp\left(-M_i \cdot \frac{\delta^{2(n-i)}}{72q^2\mu_{p^*}}\right) \\ &< \exp\left(-M_i \cdot \frac{\delta^{n-i}}{72q^2(1+\frac{1}{6q})}\right) < \exp\left(\frac{-84}{72(1+\frac{1}{6q})} \ln\left(\frac{18qnN_i}{\delta}\right)\right) \leq \ \frac{\delta}{18qnN_i}\end{aligned}$$

where the last inequality holds since $\frac{84}{72(1+1/6q)} \geq \frac{84}{72(1+1/6)} = 1$. Clearly, if $\mu_{p^*} > \delta^{n-i}(1+1/6q)$ then the probability of $\bar{\mu}_{p^*} < \delta^{n-i}$ is even smaller. We see that when the routine Extend-prefix picks any $p^* \in \mathsf{Ext}(\mathsf{prefix}_{i-1})$ in line 22, it returns that p^* in line 24 with probability more than $1 - \frac{\delta}{18qnN_i}$. On the other hand, if the probability weight of $\mathsf{Ext}_i(\mathsf{prefix}_{i-1})$ is more than $\frac{\delta^{n-i+1}}{6q}$, then the probability that none of the N_i candidates that Extend-prefix picks belongs to $\mathsf{Ext}_i(\mathsf{prefix}_{i-1})$ is at most

$$\left(1 - \frac{\delta^{n-i+1}}{6q}\right)^{N_i} = \left(1 - \frac{\delta^{n-i+1}}{6q}\right)^{\frac{6q}{\delta^{n-i+1}}\ln(18qn/\delta)} < \exp(-\ln(\frac{18qn}{\delta})) = \frac{\delta}{18qn}$$

We conclude that the probability of sub-case (b) is at most $\frac{2\delta}{18qn}$, and therefore the overall probability of the event Wrong_i is at most $\frac{\delta}{18qn} + \frac{2\delta}{18qn} = \frac{\delta}{6qn}$.

In the analysis below of the online phase, we therefore assume that the pre-processing phase never makes the wrong decision, and this assumption effects the overall error probability of A' by at most $(n-1)\frac{\delta}{6qn} < \frac{\delta}{6q}$.

The online phase of A' Consider now the on-line phase of A'. Recall that this phase gets a prefix with $v-1$ puzzles, prefix_{v-1}, and an input puzzle p, and that these two are independent (since the pre-processing phase is independent of the input). Below we denote the input puzzle by p_v^*, and we denote by c_v^* the corresponding check information (that A' never actually sees, but may determines the veracity of A''s answer.) Assuming that the pre-processing phase did not make a wrong decision, we know that

$$\mathsf{rsp}(\mathsf{prefix}_{v-1}) \geq \delta^{n-v+1}(1 - \frac{1}{6q}) \tag{1}$$

(since iteration $v-1$ of the pre-processing returned some $p_{v-1} \neq \perp$). If $v = n$, this means that $\mathsf{rsp}(\mathsf{prefix}_{v-1}) \geq \delta(1-1/6q)$, so running $A(\mathsf{prefix}_{v-1}, p_v^*)$ and taking the last answer yields a correct solution to p_v^* with probability at least $\delta(1-1/6q)$.

The more interesting case to analyze is when $v < n$, which means that iteration v in the pre-processing phase failed to extend the prefix. Assuming again that this was not a wrong decision, it means that

$$\Pr[(p,c) \xleftarrow{\$} G(1^k)\ :\ p \in \mathsf{Ext}_v(\mathsf{prefix}_{v-1})] < \frac{\delta^{n-v+1}}{6q} \tag{2}$$

From now on, we fix some value for prefix_{v-1} for which Equations 1 and 2 hold. For convenience in the discussion below, we let E be the set of pairs (p,c) such that $p \in \mathsf{Ext}_v(\mathsf{prefix}_{v-1})$. Namely,

$$E \stackrel{\text{def}}{=} \{\ (p,c) :\ \mathsf{rsp}_v(\textbf{prefix}_{v-1} \circ p) \geq \delta^{n-v}(1 + \frac{1}{6q})\ \}$$

and from Equation 2 we know that

$$\Pr[(p,c) \xleftarrow{\$} G(1^k) \;:\; (p,c) \in E] < \frac{\delta^{n-v+1}}{6q}.$$

Consider the experiment where we choose at random a single vector of $n-v+1$ puzzles, $(p_j, c_j) \leftarrow G(1^k)$ for $j = v, \ldots, n$, and then run A to get $\boldsymbol{a} \leftarrow A(\mathsf{prefix}_{v-1}, p_v, \ldots, p_n)$. We are interested in the "success" event where $\boldsymbol{a}$ contains the right answers to all the puzzles $p_v, \ldots, p_n$, and in the "almost success" event where we only know that the answers to $p_{v+1}, \ldots, p_n$ are right. For any pair (p_v, c_v) we let $w(p_v, c_v)$, $s(p_v, c_v)$, $a(p_v, c_v)$, respectively, be the probability weight of that pair, and the probabilities of "success" and "almost success" conditioned on it.

$$\begin{aligned}
w(p_v, c_v) &\stackrel{\text{def}}{=} \Pr[G(1^k) = (p_v, c_v)] \\
s(p_v, c_v) &\stackrel{\text{def}}{=} \Pr\left[\begin{array}{l} (\langle p_{v+1}, \ldots, p_n \rangle, \langle c_{v+1}, \ldots, c_n \rangle) \leftarrow G^{n-v}(1^k), \\ \boldsymbol{a} \leftarrow A(\mathsf{prefix}_{v-1}, p_v, p_{v+1}, \ldots, p_n) \\ :\; V(p_j, c_j, a_j) = 1 \text{ for all } j \in \{v, \ldots, n\} \end{array}\right] \\
a(p_v, c_v) &\stackrel{\text{def}}{=} \Pr\left[\begin{array}{l} (\langle p_{v+1}, \ldots, p_n \rangle, \langle c_{v+1}, \ldots, c_n \rangle) \leftarrow G^{n-v}(1^k), \\ \boldsymbol{a} \leftarrow A(\mathsf{prefix}_{v-1}, p_v, p_{v+1}, \ldots, p_n) \\ :\; V(p_j, c_j, a_j) = 1 \text{ for all } j \in \{v+1, \ldots, n\} \end{array}\right]
\end{aligned}$$

Clearly, the overall probability of "success" is exactly the residual-success probability of prefix_{v-1}, namely,

$$\sum_{(p_v, c_v)} w(p_v, c_v) s(p_v, c_v) = \mathsf{rsp}(\mathsf{prefix}_{v-1}) \geq \delta^{n-v+1}(1 - \frac{1}{6q}). \tag{3}$$

Also, from what we know about E

$$a(p_v, c_v) \leq \delta^{n-v}(1 + \frac{1}{6q}) \text{ for any } (p_v, c_v) \notin E, \quad \text{and} \quad \sum_{(p_v, c_v) \in E} w(p_v, c_v) \leq \frac{\delta^{n-v+1}}{6q} \tag{4}$$

Recall that A', on input p_v^*, chooses at random many continuations $(p_{v+1}, \ldots, p_n)$ $(c_{v+1}, \ldots, c_n)$ until it finds an answer vector $\boldsymbol{a}$ that is an "almost success". Then A' outputs a_v, and this is correct only if $\boldsymbol{a}$ is also a "success". This means that conditioned on not aborting, the success probability of A' is exactly $s(p_v^*, c_v^*)/a(p_v^*, c_v^*)$. That is, for any fixed pair (p_v^*, c_v^*) we have

$$\Pr[V(p_v^*, c_v^*, A'(p_v^*)) = 1 \mid A'(p_v^*) \text{ does not abort}] \;=\; s(p_v^*, c_v^*)/a(p_v^*, c_v^*) \tag{5}$$

Next, let B (for <u>B</u>ad) be the set of input puzzles (and their associated check information) on which A' is unlikely to succeed. More specifically,

$$B \stackrel{\text{def}}{=} \{(p_v, c_v) \;:\; s(p_v, c_v) < \delta^{n-v+1}/6q\}. \tag{6}$$

It is easy to see that when $(p_v^*, c_v^*) \notin B$, then $A(p_v^*)$ almost never aborts. Indeed A' aborts only if it does not find puzzles $p_{v+1}, \ldots, p_n$ that A solves correctly after $\frac{6q\ln(6q)}{\delta^{n-v+1}}$ trials. As each trial success with probability $a(p_v^*, c_v^*) \geq s(p_v^*, c_v^*) \geq \frac{\delta^{n-v+1}}{6q}$, the probability that they all fail is at most $1/6q$,

$$\Pr_{A'}[A'(p_v^*) \text{ aborts}] \leq \frac{1}{6q}, \text{ for all } (p_v^*, c_v^*) \notin B \tag{7}$$

The last crucial observation that we need is that the sets B and E together cannot contribute too much to the probability of success. Namely,

$$\begin{aligned}\sum_{(p_v,c_v)\in B\cup E} w(p_v,c_v)s(p_v,c_v) &\leq \sum_{(p_v,c_v)\in B} w(p_v,c_v)s(p_v,c_v) \;+ \sum_{(p_v,c_v)\in E} w(p_v,c_v)s(p_v,c_v) \\ &\leq \sum_{(p_v,c_v)\in B} w(p_v,c_v)\delta^{n-v+1}/6q \;+ \sum_{(p_v,c_v)\in E} w(p_v,c_v) \\ &\leq \frac{\delta^{n-v+1}}{6q} + \frac{\delta^{n-v+1}}{6q} \;=\; \frac{\delta^{n-v+1}}{3q}\end{aligned}$$

and combined with Equation 3 we get

$$\sum_{(p_v,c_v)\notin B\cup E} w(p_v,c_v)s(p_v,c_v) \;\geq\; \delta^{n-v+1}(1-\frac{1}{6q}) - \frac{\delta^{n-v+1}}{3q} \;=\; \delta^{n-v+1}(1-\frac{1}{2q}) \tag{8}$$

Putting everything together, we have

$$\begin{aligned}&\Pr[A' \text{ answers correctly}] \;=\; \sum_{(p_v^*,c_v^*)} w(p_v^*,c_v^*)\cdot\Pr[V(p_v^*,c_v^*,A'(p^*))=1] \\ &\geq \sum_{(p_v^*,c_v^*)\notin B\cup E} w(p_v^*,c_v^*)\cdot(1-\Pr[A'(p_v^*) \text{ aborts}]) \\ &\qquad\qquad\cdot\Pr[V(p_v^*,c_v^*,A'(p_v^*))=1 \mid A'(p_v^*) \text{ does not abort}] \\ &\overset{(a)}{\geq} \sum_{(p_v^*,c_v^*)\notin B\cup E} w(p_v^*,c_v^*)\cdot(1-\frac{1}{6q})\cdot\frac{s(p_v^*,c_v^*)}{a(p_v^*,c_v^*)} \\ &\overset{(b)}{\geq} \sum_{(p_v^*,c_v^*)\notin B\cup E} w(p_v^*,c_v^*)\cdot(1-\frac{1}{6q})\cdot\frac{s(p_v^*,c_v^*)}{\delta^{n-v}(1+1/6q)} \\ &= \frac{1-1/6q}{\delta^{n-v}(1+1/6q)} \sum_{(p_v^*,c_v^*)\notin B\cup E} w(p_v^*,c_v^*)s(p_v^*,c_v^*) \\ &\overset{(c)}{\geq} \frac{1-1/6q}{\delta^{n-v}(1+1/6q)}\cdot\delta^{n-v+1}(1-1/2q) = \delta\cdot\frac{(1-1/6q)(1-1/2q)}{1+1/6q} > \delta(1-\frac{5}{6q})\end{aligned}$$

where inequality (a) is due to Equations 5 and 7, inequality (b) is due to (the first part of) Equation 4, and inequality (c) is due to Equation 8. We conclude that the probability of a wrong decision in the preprocessing phase is at most $\delta/6q$, and that if no wrong decisions were made then the online phase of A' solves the input puzzle with probability at least $\delta(1-5/6q)$, hence the overall success probability of A' is at least $\delta(1-5/6q) - \delta/6q = \delta(1-1/q)$. This completes the proof of Lemma 1. □

3.3 Proof of Theorem 1

All that is left now is to provide an asymptotic interpretation to the concrete bounds from Lemma 1. Let $\mathcal{Z}$ be a $(1-\varepsilon)$-hard puzzle system, and assume toward contradiction that there exists a T-time solver S^n that (for infinitely many k's) solves (G^n, V^n) with probability at least $\varepsilon(k)^n + 1/r(k)$, where both $T(\cdot)$, $r(\cdot)$ are polynomials.

Let us denote $q = 4nr$, and let δ be the solution to $\delta^n = \varepsilon^n + 1/r$. We note that since δ^n is noticeably larger than ε^n, then also δ is noticeably larger than ε, specifically $\delta > \varepsilon + 1/2rn$. To see this, denote $\gamma = \delta - \varepsilon$, and assume toward contradiction that $\gamma < 1/2rn < 1/n$. Then we have

$$\begin{aligned}
(\varepsilon+\gamma)^n = \varepsilon^n + \sum_{t=1}^{n}\binom{n}{t}\varepsilon^{n-t}\gamma^t \quad &= \quad \varepsilon^n + \gamma\sum_{t=1}^{n}\binom{n}{t}\varepsilon^{n-t}\gamma^{t-1} \\
< \varepsilon^n + \gamma\sum_{t=1}^{n}\binom{n}{t}\frac{\varepsilon^{n-t}}{n^{t-1}} \quad &< \quad \varepsilon^n + \gamma\sum_{t=1}^{n}\frac{n^t}{t!}\cdot\frac{\varepsilon^{n-t}}{n^{t-1}} \\
= \varepsilon^n + \gamma n\sum_{t=1}^{n}\varepsilon^{n-t}/t! \quad &< \quad \varepsilon^n + 2\gamma n
\end{aligned}$$

Thus, $\varepsilon^n + 2\gamma n > (\varepsilon+\gamma)^n = \delta^n = \varepsilon^n + 1/r$, and therefore $\gamma > 1/2rn$, contradiction.

Applying Lemma 1 with the given n, q, δ and T, we get an algorithm S for solving (G, V), with success probability

$$\delta(1-1/q) \;\geq\; (\varepsilon + 1/2rn)(1-1/4rn) \;>\; \varepsilon + 1/8rn,$$

which is noticeably larger than ε (since r, n are both polynomial in k). The running time of S is polynomial in n, q, T, T_G, T_V and $1/\delta^n$, which are all polynomial in k. (Note that $\delta^n = \varepsilon^n + 1/r$ is noticeable since r is polynomial in k.) This contradicts the $(1-\varepsilon)$ hardness of $\mathcal{Z}$, concluding the proof of Theorem 1. □

4 The weak verification property (informal)

As discussed in the introduction, the proof of Theorem 1 relies on the fact that we have efficient generation and verification algorithms, so that A' can generate puzzles and recognize correct solutions to these puzzles. We now show that without this property, hardness amplification is not guaranteed. In particular, we describe a "puzzle system" where the verification algorithm is not efficient, and show that solving a few puzzles in parallel is not any harder than solving just one.

The example is essentially the one that was used by Bellare, Impagliazzo and Naor, except that we do not need the last two flows of their protocol. Assume that we have a non-interactive, perfectly-binding commitment scheme for one bit, $C(\cdot)$, and fix some parameter n. The generator for the puzzle system picks at random a bit b, and outputs as puzzle a random commitment $c = C(b)$. A potential solution to this puzzle is a vector of $n-1$ commitments $c'_1, \ldots, c'_{n-1}$ such that (i) each c'_i is indeed a valid commitment to some bit b'_i, (ii) $c'_i \neq c$ for all i, and (iii) $b \oplus b'_1 \oplus \cdots \oplus b'_n = 0$. (Note that the last condition is well defined since the commitment scheme is perfectly binding.)

It is easy to see that if $C(\cdot)$ is non-malleable [5], then no efficient solver can solve this system with probability noticeably more than 1/2. Informally, since the solver cannot return $c'_i = c$, then the bits b'_i must be almost independent of b, so their sum must also be almost independent of b, and therefore $\Pr[b \oplus b'_1 \oplus \cdots \oplus b'_n = 0] \approx 1/2$.

On the other hand, a solver that gets n random puzzles $c^1, \ldots, c^n$ can return as a solution to puzzle c^i all the other puzzles $c^j, j \neq i$. To analyze the success probability of the solver, let b_i denote the committed bit defined by commitment c_i. (b_i is well defined since the commitment is perfectly binding.) Then, with probability one half we have that $b_1 \oplus \cdots \oplus b_n = 0$. In this case, the solver has solved *all* the puzzles. (Indeed, the probability that any two of the c^i's are the same is negligible.)

The only problem with this example is that there are no known provable constructions of non-interactive, perfectly-binding, non-malleable commitment schemes in the "bare model". Such schemes are only known to exist in the common-random-string model [4] (or the common-reference-string model, or the random-oracle model, etc.) At the current state of affairs, this example is therefore only valid with respect to one of these models. However, since we only need non malleability with respect to one specific relation, it may be possible to devise such scheme in the bare model. In particular, assuming that such scheme exist seems like a rather reasonable assumption. (We comment that the negative result of Bellare, Impagliazzo, and Naor, for four-round proofs also requires non-interactive, non-malleable commitment schemes.)

References

1. L. von Ahn, M. Blum, N. Hopper, and J. Langford. CAPTCHA: Using hard AI problems for security. In *Advances in Cryptology - EUROCRYPT'03*, volume 2656 of *Lecture Notes in Computer Science*, pages 294–311. Springer-Verlag, 2003.
2. M. Bellare, R. Impagliazzo, and M. Naor. Does parallel repetition lower the error in computationally sound protocols? In *38th Annual Symposium on Foundations of Computer Science* (FOCS'97), pages 374–383. IEEE, 1997.
3. S. Ben-David, B. Chor, O. Goldreich, and M. Luby. On the theory of average case complexity. *Journal of Computer and System Sciences*, 44(2):193–219, 1992. Preliminary version in STOC'89.
4. G. Di Crescenzo, Y. Ishai, and R. Ostrovsky. Non-interactive and non-malleable commitment. In *Proceedings of the thirtieth annual ACM symposium on theory of computing (STOC'98)*, pages 141–150. ACM Press, 1998.
5. D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. *SIAM J. on Computing*, 30(2):391–437, 2000. Preliminary version in STOC'91.
6. O. Goldreich. *Foundations of Cryptography, Basic tools.* Cambridge University Press, 2001.

7. R. Impagliazzo and L. A. Levin. No better ways to generate hard NP instances than picking uniformly at random. In *31st Annual Symposium on Foundations of Computer Science (FOCS'90)*, pages 812–821. IEEE, 1990.
8. A. Juels and J. Brainard. Client puzzles: A cryptographic defense against connection depletion attacks. In *Proceedings of the 1999 Network and Distributed System Security Symposium (NDSS'99)*, pages 151–165. Internet Society (ISOC), 1999.
9. L. A. Levin. Average case complete problems. *SIAM Journal of Computing*, 15(1):285–286, 1986. Preliminary version in STOC'84.
10. S. Myers. Efficient amplification of the security of weak pseudo-random function generators. *Journal of Cryptology*, 16(1):1–24, 2003. Extended Abstract appeared in EUROCRYPT 2001.
11. M. Naor. Verification of a human in the loop or identification via the Turing test. Manuscript, available on-line from `http://www.wisdom.weizmann.ac.il/~naor/PAPERS/human_abs.html`, 1996.
12. R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and time-released crypto. Technical Report MIT/LCS/TR-684, MIT laboratory for Computer Science, 1996.
13. A. C. Yao. Theory and applications of trapdoor functions. In *23rd Annual Symposium on Foundations of Computer Science*, pages 80–91. IEEE, Nov. 1982.

A Results for Proof systems

A computationally-sound proof system for a language $L \subseteq \{0,1\}^*$, is a pair $\mathcal{P} = (P, V)$ of polynomial time interactive Turing machines, who gets a common input string x, whose length is considered the security parameter. (The honest prover may also get some additional input that the verifier does not see.) Since we only care about soundness, then it is enough to consider only the verifier V. The success probability of a "cheating prover" B on input x is the probability that V accepts when interacting with B on common input x. The probability is taken over the randomness of B and V.

Definition 2 (Soundness error). *Fix a language L and a verifier V, and let $\varepsilon : \mathbb{N} \to (0,1)$ be some function. We say that V has soundness error at most ε if for any efficient prover B there is a negligible function* negl*, such that for any $x \notin L$, the success probability of $(B, V)(x)$ is at most $\varepsilon(|x|) + \mathsf{negl}(|x|)$.*

We comment that to be of interest, the proof system also has to satisfy some completeness property (say, for a prover that is given an NP witness for the membership of x in L). In this work, however, we are only interested in soundness.

The n-fold parallel repetition of a proof system $\mathcal{P} = (P, V)$ is defined in a straightforward manner: there are n independent copies of the verifier (and the honest prover), all running in "lock steps" in parallel on the same common input x. The n-fold parallel repetition of $\mathcal{P}$ is denoted $\mathcal{P}^n = (P^n, V^n)$.

A.1 Hardness amplification for three-round proof systems

A rather straightforward adaptation of our result from Section 3 yields:

Theorem 2. *Let $\varepsilon : \mathbb{N} \to (0, 1)$, $n : \mathbb{N} \to \mathbb{N}$ be efficiently computable functions, where n is polynomially bounded, and let $\mathcal{P} = (P, V)$ be an interactive proof system with three flows (or less) and soundness error at most ε. Then the proof system $\mathcal{P}^n$ has soundness error at most ε^n.*

Proof. (sketch) We again assume that there exists a cheating prover B such that for infinitely many common inputs $x \notin L$, the success probability of $(B, V^n)(x)$ is at least $\varepsilon(|x|)^n + 1/r(|x|)$ for some polynomial r. We show a cheating prover B' such that for those x'es, the success probability of $(B', V)(x)$ is noticeably more than ε.

Assume w.l.o.g. that the proof system has exactly three flows of communication, which means that the first flow is from the prover to the verifier. The prover B' first samples (polynomially) many first-flow messages of B, and for each one it estimates the success probability of $(B, V^n)(x)$ conditioned on that message. B' tries $N = poly(r, n)$ candidate first messages, and for each candidate it estimates the conditional success probability with accuracy $1/4r$ and confidence $1/32rnN$. Since the overall success probability of P is at least $\varepsilon^n + 1/r$, then P' can find a first-flow message $\boldsymbol{y}$ with estimated conditional success probability of at least $\varepsilon^n + 3/4r$ after trying only $N = poly(r, n)$ candidates. With the given accuracy and confidence bounds, it follows that except with probability $1/32rn$, P' indeed finds such first-flow message $\boldsymbol{y}$, and the conditional success probability of $\boldsymbol{y}$ is at least $\varepsilon^n + 1/2r$.

Now P' consider the puzzle system with the generating algorithm defined by the verifier V^n on input x and $\boldsymbol{y}$, and whose verifying function is the final verification procedure of V^n. (This does not quite satisfy our definition of the n-fold repetition of a puzzle system, since each copy now has a different generating and verifying procedures, because of the different first-flow messages. However, this difference does not change anything in the proof of Lemma 1.) P' applies the transformation from Lemma 1 with $\delta^n = \varepsilon^n + 1/2r$, and slackness $q = 8rn$, thus obtaining a strategy that convinces V with probability at least $\varepsilon + 1/16rn$. Since we have an error probability $1/32rn$ for choosing $\boldsymbol{y}$, then the overall success probability of $(P', V)(x)$ is at least $\varepsilon + 1/32rn$, which is noticeably more than ε.

A comment about the common-reference-string model Computationally-sound proofs were define in the work of Bellare et al. [2] somewhat more generally than above: essentially they defined proofs in the common-reference-string model. We note, however, that their "positive result" (as well as ours) does not extend to this model. Indeed, they only show hardness amplification when the common reference string is fixed (so the soundness error is defined with respect to a fixed reference string, rather than with respect to a random choice of that string).

For example, one may think of a cheating prover B that on ε^n fraction of the reference strings is able to convince the verifier with probability one, and on other strings it always fails. It is not hard to see that no black-box reduction can transform this prover to one that succeeds in a single proof with probability more than ε^n.