Cryptographic Puzzles and DoS Resilience, Revisited

Bogdan Groza[1] and Bogdan Warinschi[2]

[1] Faculty of Automatics and Computers,
Politehnica University of Timisoara,
Bd. V. Parvan nr. 2,
Timisoara, Romania
bogdan.groza@aut.upt.ro
[2] Computer Science Department,
University of Bristol,
Woodland Road,
Bristol, BS8 1UB,
United Kingdom.
bogdan@cs.bris.ac.uk

Abstract. Cryptographic puzzles (or client puzzles) are moderately difficult problems that can be solved by investing non-trivial amounts of computation and/or storage. Devising models for cryptographic puzzles has only recently started to receive attention from the cryptographic community as a first step toward rigorous models and proofs of security of applications that employ them (e.g. Denial-of-Service (DoS) resistance). Unfortunately, the subtle interaction between the complex scenarios for which cryptographic puzzles are intended and typical difficulties associated with defining concrete security easily leads to flaws in definitions and proofs. Indeed, as a first contribution we exhibit shortcomings of the state-of-the-art definition of security of cryptographic puzzles and point out some flaws in existing security proofs. The main contribution of this paper are new security definitions for puzzle difficulty. We distinguish and formalize two distinct flavors of puzzle security which we call optimality and fairness and in addition, properly define the relation between solving one puzzle vs. solving multiple ones. We demonstrate the applicability of our notions by analyzing the security of two popular puzzle constructions.
We briefly investigate existing definitions for the related notion of security against DoS attacks. We demonstrate that the only rigorous security notion proposed to date is not sufficiently demanding (as it allows to prove secure protocols that are clearly not DoS resistant) and suggest an alternative definition. Our results are not only of theoretical interest: the better characterization of hardness for puzzles and DoS resilience allows establishing formal bounds on the effectiveness of client puzzles which confirm previous empirical observations. We also underline clear practical limitations for the effectiveness of puzzles against DoS attacks by providing simple rules of thumb that can be easily used to discard puzzles as a valid countermeasure for certain scenarios.

1 Introduction

Background. Cryptographic puzzles are moderately difficult problems that can be solved by investing non-trivial amounts of computation and/or memory. A typical use for puzzles is to balance participants costs during the execution of some protocols. For example, many papers addressed their use against resource depletion in SSL/TLS [10], TCP/IP [20], general authentication protocols [3, 15], spam combat [12], [11], [17]. The use of puzzles reaches beyond balancing resources: they can be used as proof-of-work in other applications (like timestamping) or through a clever application in encryption into the future [27]. Puzzles are accounted under various names: cryptographic puzzles, client puzzles, computational puzzles or proofs of work, we prefer the first one since the puzzles that we study are intrinsically based on cryptographic functions.

Most of the puzzle-related literature concentrates on providing constructions, often with additional, innovative properties. For example puzzles that are non-parallelizable prevent an adversary from using distributed computations to solve them. Examples of constructions include the well known time-lock puzzle [27], the constructions proposed by Tritilanunt et al. in [31] and later by Jeckmans [18], Ghassan and Čapkun [21], Tang and Jeckmans [30]. All of these constructions can ensure that a puzzle-solver spends computation cycles before a server engages in any expensive computation. To alleviate computational disparities between solvers, Abadi et al. [1] build puzzles that rely on memory usage rather than on CPU speed, this leading to a more uniform behaviour between devices. For completeness, in Appendix A we make a brief survey over puzzle properties from related work.

Given the wide-range of applications for puzzles and the number of proposed constructions it is probably surprising that devising formal security notions for puzzles has received rather little attention so far, with only two notable exceptions. Chen et al. [9] initiate the formal study of security properties for puzzles. They identify two such properties. *Puzzle difficulty* requires that no adversary can solve *a single puzzle* faster than some prescribed bound, whereas *puzzle unforgeability* requires that no adversary can produce a valid-looking puzzle. While this latter property is not required by all scenario usages for puzzles, the former one is critical. In a recent paper, Stebila et al. [28] notice that single-puzzle difficulty may not suffice to guarantee security when puzzles are used in real applications, since here it may be needed that an adversary does not solve *multiple puzzles* faster than some desired bound, and the relation between single-puzzle difficulty and multi-puzzle difficulty is unclear at best, and completely inexistent at worst.

To fix this, Stebila et al. [28] propose a notion of puzzle difficulty that accounts for multiple puzzles being solved at once and prove that two existing constructions HashInversion (initially used by Juels and Brainard [20]) and HashTrail (initially used in the Hashcash system [4]) meet this notion. The main motivation for the work in this paper is that the proposed security definition is problematic: the notion defined is incomplete since it does not account for the tightness of the bounds and, strictly speaking, it cannot be met by any existing scheme. This does not contradict the security proofs mentioned above as the claims rely on faulty analysis: the difficulty bound provided for the HashInversion puzzle is wrong while for HashTrail is largely overestimated.

Our results. The main contribution of our paper are new security notions for puzzle difficulty. We distinguish between several different flavors of puzzle difficulty. The first property demands that no adversary can solve the puzzle faster than by using the "prescribed" algorithm (i.e. the puzzle-solving algorithm that is associated to the puzzle). We call such puzzles *optimal*. The formulations of this notion is in the multi-puzzle setting which, as correctly observed in [28], is the case relevant for most practical applications. While it is not true in general that for a puzzle construction solving n puzzles takes n times the resources needed for solving one puzzle, this is clearly a desirable property. We capture this intuition through a property that we call *difficulty preserving*, an attribute which is directly linked to *optimality* showing the later to be the desired property for a puzzle. Having fixed the definitions we move to the analysis of two popular puzzle systems HashTrail and HashInversion. We prove that, in the random oracle model, these puzzles are optimal and difficulty preserving for concrete difficulty bounds that we derive. Finally, we look at existing work on using puzzles for provable DoS resistance. Unfortunately, we discovered that the formal definition for DoS resilience proposed by [28] is not strong enough as it allows for clear attacks against protocols that are provably secure according to the definition. We then design and justify a new security definition that does not suffer from the problems that we have identified.

The work in this paper extends and refines our previous results [16]. The main contribution are several new security definitions that are stronger, more convenient, and of higher practical relevance. The newly introduced notions stem from the contrast between the *sequential* and *concurrent* solving games that allow us to formally define *difficulty relations* between puzzles. These also set way for a more rigorous treatment of difficulty preserving puzzles which does not require optimality and relies strictly on properties of the adversary rather than on properties of the solving algorithm. A new relevant puzzle property is also added: *fairness*. This addresses both the probability for an adversary to falsely claim that he solved the puzzle, i.e., *fairness in answering*, as well as the average vs. the maximum solving time, i.e., *fairness in solving*. We also take further our analysis on DoS-resistance with client puzzles. While in [16] our scrutiny was limited to the simplest proof-of-work approach, here we depict practical limitations for two more advanced proof-of-work schemes that use filters to separate between clients and adversaries. Finally, our work here includes more proofs and details that were absent in our previous work due to space limitations.

Before we move on, we note that getting the security definitions for puzzles and DoS security right is quite important as more and more works in this direction have appeared (a book chapter in [8] and also [25] and [29]) and all seem to have inherited the weaknesses in the definition of [28].

2 Shortcomings of existing definitions and proofs

The first attempt to formalize puzzle properties, and in particular puzzle difficulty, was by Chen et al. in [9]. Recently, Stebila et al. [28], motivated by the observation that the security notion of [9] does not guarantee that solving n puzzles is n times harder than solving one, introduced a new definition of puzzle difficulty. In brief, a puzzle is deemed $\epsilon_{k,d,n}(\cdot)$-strongly difficult if the success probability of an adversary is less or equal to $\epsilon_{k,d,n}(\cdot)$ and $\epsilon_{k,d,n}(t) \leq \epsilon_{k,d,1}(t/n)$ (this later condition enforcing stronger difficulty w.r.t. n puzzles). Here k is a security parameter, d is the difficulty level and n denotes the number of solved puzzles.

2.1 Shortcomings of existing definitions

There are several weak points in the difficulty definition outlined above. One shortcoming is that the property of a puzzle of being strongly difficult [28] is in fact a property of the function ϵ that upper-bounds the success of the adversary. However, ϵ is an upper bound on the hardness of the puzzle, but not necessarily the tightest possible (for example if one sets $\epsilon_{k,d,n} = 1$ any puzzle is $\epsilon_{k,d,n}$-strongly difficult). A natural question is then what if one can find a bound that deems the puzzle strongly difficult, while for some other tighter bounds this property does not hold anymore. Should we consider such a puzzle strongly difficult or not? Note that in contrast, Chen et al. in [9] clearly state that any puzzle that is ϵ difficult is $\epsilon + \mu$ difficult and the most accurate difficulty bound is the *infimum* of ϵ. The point is not that one would find such a bound on purpose, but rather as security reductions are not trivial one could find a good bound with respect to which the puzzle is strongly difficult, just to turn out that the puzzle is not strongly difficult for a tighter bound. There is an important qualitative distinction between puzzles and the majority of cryptographic primitives when it comes down to the tightness of the security bounds. For most cryptographic constructions the tightness of bound matters, since it impacts the parameters and hence the efficiency of schemes. Even with loose bounds, secure instantiations are possible at the expense of efficiency. For cryptographic puzzles, tight bounds are more important since a loose bound does not necessarily means that the time to solve the puzzle increases proportional over multiple instances even in the case when the bound does so.

The following example of a time-lock puzzle shows that the tightness of the bound matters. We skip the formalism as we want to keep this example as intuitive as possible. Set m to be an RSA-like modulus (sufficiently large to rule out any insecurity) and assume that solving one puzzle means given $x \in_R [0..2^{k-1}]$ to compute $x^{2^d} \bmod m$. We assume the usual hypothesis that this computation cannot be done faster than d squarings unless one knows the factorization of the modulus. Suppose the adversary can get 1 or 2 fresh values x and has to compute $x^{2^d} \bmod m$ for each of them with no prior knowledge of the modulus. We can say that the success probability of the adversary is upper bounded by $\epsilon_{k,d,n}(t) = \frac{t}{n \cdot d}, \forall n \in \{1, 2\}$. To check for correctness, indeed, if $n = 1$ the probability to find the output for less than d steps (one step means one squaring) is almost 0 assuming a sufficiently large modulus and 1 at d steps. While for $n = 2$, for less than d steps the probability is 0, at d steps the adversary has solved the first puzzle, while the probability that the second is also solved is 2^{-k} due to the possibility of colliding x_1, x_2, and 2^{-k} is lower than 1/2 claimed by the upper bound. Thus the bound holds and one can also verify that $\epsilon_{k,d,1}(t/2) = \epsilon_{k,d,2}(t)$ so the puzzle is $\epsilon_{k,d,n}(t)$-strongly difficult. We set some artificially small parameters just to easily exhibit some calculation. Let $k = 16$ and $d = 2^{16}$

(the bound holds for these values as well). One would expect that solving the two puzzles requires $2 \times 2^{16} = 131072$ steps. However, due to the possibility of colliding inputs the average number of steps is actually $2^{16} - 1 = 131071$, that is, one step is missing. The numbers given here are artificially small and the variation is not very relevant, but it has the sole purpose to show that the criterion has some deficiencies. The problem here is that the bound is not tight enough. More precise bounds that should have been used are: $\epsilon_{k,d,1} = 0$ if $t \in [0, d)$, $\epsilon_{k,d,1} = 1$ if $t = d$ and $\epsilon_{k,d,2} = 0$ if $t \in [0, d)$, $\epsilon_{k,d,2} = 2^{-k}$ if $t \in [d, 2d)$ and $\epsilon_{k,d,2} = 1$ if $t = d$. For these bounds indeed $\epsilon_{k,d,1}(t/2) \leq \epsilon_{k,d,2}(t)$ which shows that in fact the puzzle is not strongly difficult. These bounds are also informal and we used them just as an intuition, indeed for any $t < d$ the adversary can still guess the solution with negligible (but non-zero) probability. We can prove, and we specify this in a remark that follows, that if the bound is tight then the condition from [28] is sufficient to make a puzzle difficulty preserving.

But, one may further ask if this condition is really necessary. The answer is negative. In fact, quite surprisingly, the HashTrail puzzle does not satisfy it and neither does the HashInversion puzzle (while both of them can be proved to be difficulty preserving). We call HashInversion the generic puzzle which consists in the partial inversion of a hash function, that is given $x'', H(x'||x'')$ find x'. Also, we refer HashTrail as the generic puzzle which consists in finding an input to $H(r||\cdot)$ such that the result has a fixed number of trailing zeros. Both these constructions are frequently used in many proposals. The first one is used by Jules and Brainard in [20] and the second by Back in the Hashcash system [4]. We prefer the generic names HashInversion and HashTrail as these suggest better what means to solve the puzzle as well as we are not interested in the specific details for the construction of the puzzles used in [4], [20].

Figure 1 shows the advantages for both these puzzles at $d = 8$ for $n = 1$ and $n = 3$. For $n = 1$ we divide the number of steps with 3 as indicated in the criterion from [28]. Note that the intersection of the bounds is obvious, thus $\epsilon_{k,d,n}(t) \leq \epsilon_{k,d,1}(t/n)$ is not satisfied. The bounds in [28] satisfy the criterion, but we show these bounds to be wrong.

Moreover, and this is another weakness for the definition of [28], the criterion $\epsilon_{k,d,n}(t) \leq \epsilon_{k,d,1}(t/n)$, can never hold in general. The reason is that in the game that defines security of multiple puzzle it is possible with some (negligible) probability that the challenge puzzles contain *two identical* puzzles. In this case solving n puzzles should *always* require less effort than n times the effort required to solve a single puzzle, at least up to negligible factors. The definition should therefore allow for this kind of slack, i.e. it should require that $|\epsilon_{k,d,n}(t) - \epsilon_{k,d,1}(t/n)| \leq k^{-\omega(1)}$. The time-lock puzzle seems to satisfy such a criterion, but note that this is certainly not the case for the hash-based puzzles above which are the most commonly employed solution in practice.

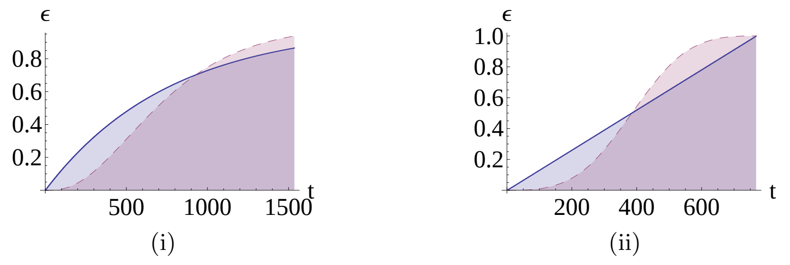

Fig. 1: Adversary advantage at $n = 1$, $d = 8$, i.e., $\epsilon_{k,8,1}(t/3)$, (continuous line) and $n = 3$, $d = 8$, i.e., $\epsilon_{k,8,3}(t)$, (dotted line) for HashTrail (i) and HashInversion (ii) puzzles

2.2 Shortcomings of existing proofs

In light of the above comments, it is natural to ask how tight is the bound obtained in [28]. By inspecting the security proofs it turns out that besides the conceptual shortcoming in judging the hardness of n puzzle instances, the bound used for the HashTrail puzzle is extremely loose while the bound for the HashInversion puzzle is wrong (these puzzles are difficulty preserving as we show later in the paper, but unfortunately the proofs provided in [28] are wrong). Figure 2 depicts the loose bound in (i) and the wrong bound in (ii) for the case of $n = 3$ puzzles of difficulty $d = 8$ bits. Note that in (ii) the adversary advantage is well underestimated.

We give a short numerical example to illustrate this. The difficulty bound claimed in [28] for the HashInversion puzzle is $\epsilon_{k,d,n} = (\frac{q+n}{n \cdot 2^d})^n$ and the puzzle is deemed strongly difficult with respect to this bound. Just to show that this bound is wrong consider the trivial case of $n = 2$, $d = 3$, i.e., the case of solving 2 puzzles each having 3 bits. Consider an adversary running at most 11 steps. According to the aforementioned bound, one would expect that the advantage of the adversary is less than $(\frac{11+2}{2 \cdot 2^3})^2 = (\frac{13}{16})^2 \approx 0.66$. Consider the naive (yet the best) algorithm that successively walks trough the set $\{0, 1, 2, ..., 7\}$ in order to solve each puzzle. The success probability of this algorithm is actually bigger than 0.66 as one can easily show. The naive algorithm can solve two puzzles in 11 steps if, given x'_1 and x'_2 the two solutions, it holds that $x'_1 + x'_2 \leq 9$. That is, there exists 1 solutions for 2 steps (the pair $\{(0,0)\}$), 2 solutions for 3 steps (the pairs $\{(0,1),(1,0)\}$) and so on, $k-1$ solutions for k steps up to $k = 9$ steps. From there on, one can note that for 10 steps given the set of pairs $\{(0,8),(1,7),(2,6),(3,5),(4,4),(5,3),(6,2),(7,1),(8,0)\}$ one must discard the first and the last pair (since 8 is not a valid value for the 3 bit guess) while for 11 steps one must discard the first 2 and the last 2 pairs. Summing up, the naive algorithm succeeded in $1 + 2 + 3 + ... + 8 + 7 + 6 = 36 + 13 = 49$ out of the obvious $2^3 \times 2^3$ variants which gives a success probability of $\frac{49}{64} \approx 0.76$.

Thus the naive algorithm does better than the success probability of the adversary considered in [28] and the discrepancy is due to the flawed security proof. The difference is not big in this example, but obviously it gets significant when one increases the values of n and d.

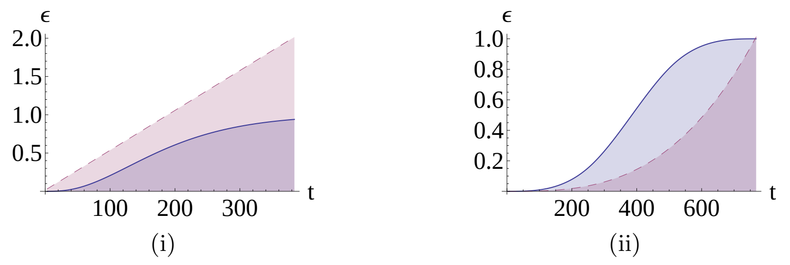

Fig. 2: Adversary advantage at $n = 3$, $d = 8$ for HashTrail (i) and HashInversion (ii) puzzles according to Stebila et al. (dotted line) and in this paper (continuous line)

2.3 Single puzzle difficulty does not imply multiple puzzle difficulty

Solving multiple puzzles is a form of parallel repetition which is well known not to always lower the error [5]. The following examples show that this is indeed the case for puzzles.

To begin with, we briefly enumerate the examples given in [28]. We skip the details and underline just the arguments of the authors in the three examples. First, the generic construction from [9] is shown not be strongly difficult since finding the master key of the server (of k bits) allows building for free any number of puzzles and solutions (since the master key is known), which is easier than solving a puzzle of difficulty k bits. Second, a time-lock puzzle built on 512 bit RSA modulus is shown not to be strongly difficult since for a puzzle difficulty of 2^{20} and 2^{30} puzzles, it may be easier to factor the modulus instead of solving the puzzles. Third, for a puzzle based on signature forgery it is again argued that factoring can be easier than solving a large number of puzzles. However, these examples basically show that whenever n instances of difficulty d exceed the security level k of the underlying primitive it is easier to solve the corresponding hard problem, e.g., breaking MAC codes, factoring, etc. One interpretation is that they are based on a poorly chosen security level k rather than on the fact that difficulty is not preserved. In practice, one would generally expect that, given k as security parameter, the underlying hard problem is at least sub-exponentially hard to solve, e.g., $2^{f(k)}$ steps, while the number of instances n is bounded by some polynomial, e.g., $p(k)$ clients, and the difficulty d still within reach to be eventually solved by clients, e.g., 2^d steps for some reasonably small d. Bottom line, for all of the previous examples a correctly chosen k would have void the examples of any practical significance, e.g., fix an 4096 bits RSA modulus (rather than 512) and there is no practically relevant n and d for which the construction would not preserve hardness (in the previously mentioned circumstances). Generally, if one forces sufficiently many instances n for any difficulty level d and security parameter k there is no puzzle construction to satisfy hardness amplification. These examples would have been more relevant if, for rigorous treatment, the security parameter k would have been included in the bound. But in [28] for HashTrail the bound is set to $\epsilon_{k,d,n} = (\frac{q+n}{n \cdot 2^d})$ while for HashInversion it is set to $\epsilon_{k,d,n} = (\frac{q+n}{n \cdot 2^d})^n$ - obviously k is missing from any of these bounds. Note that the weaker difficulty bounds from [9] do include the security parameter. It is worth mentioning that in [28] there is also a fourth example for showing that solving more puzzles can be easier. MicroMint is given as example, here it is commonly known that finding more collisions is easier than finding one collision (due to the birthday paradox). This example however, as the authors from [28] also note, shows that finding more solutions for a puzzle is easier than finding one. Thus it has less to do with solving more than one puzzle.

Are there more natural examples which show that single puzzle difficulty does not imply multiple puzzle difficulty without jeopardizing the difficulty bound? Yes. Not directly on the client puzzles discussed here, but there is a whole line of research on whether parallel composition amplifies or not difficulty. Roughly, one can view solving multiple puzzles (the concurrent solving game as we call it further) as parallel composition. We give an example inspired by the work in [5]. This is an artificial example, but it is useful to understand why parallel solving may be easier even without trying to solve the puzzle at all. Consider f some publicly known function that is easily computable by the server side but otherwise difficult for the client (it is easy to come up with such a function, consider for example integer factorization as the trapdoor, and exponentiation for some large exponent 2^d as the function, i.e., the time-lock puzzle). Say puzzle solving means that the server hides bit b by computing $f(r,b)$ and the client has to find b given $r, f(r,b)$ then to reply with a valid solution $r', f(r',b')$ such that $b' \neq b$. As f is public, just one computation of f is enough to solve this. Obviously solving one puzzle without any computation of f has probability 1/2. But surprisingly, solving two puzzles without computing f has the same probability and not 1/4 as one expects. That is, if the client receives two puzzles $r_1, f(r_1,b_1)$, $r_2, f(r_2,b_2)$ he can simply reply by switching the puzzles with $r_2, f(r_2,b_2)$, $r_1, f(r_1,b_1)$ which is a correct solution with probability 1/2.

3 Difficulty notions

To formalize puzzle difficulty and related notions we proceed as follows. First we define the usual concurrent game of solving multiple puzzles and bound the adversary advantage. Second we define a sequential solving game and then based on relations between difficulty bounds we define difficulty preserving puzzles. We further define puzzle optimality which means that, up to some negligible factor, there is no adversary that can solve one or more puzzles with better advantage than the solving algorithm that comes with the puzzle. This property was generally ignored, we consider it to be the most relevant, since if an adversary can solve puzzles in less steps than the puzzle solving algorithm, then such a construction may have no use at all. Further, in a forthcoming proposition we establish that if the puzzle is optimal (assuming the usual way of solving more puzzles by running the solving algorithm on each of the puzzles) the puzzle is difficulty preserving and solving n puzzles is n times as hard as solving one. For completeness, we also define fairness with respect to both the probability to guess a solution as well as to solving time.

3.1 Syntax for cryptographic puzzles

Our definition of a puzzle system follows in spirit the definition from [9], with several differences. One is that we do not consider arbitrary strings as inputs together with keys to the puzzle generation, but instead we group these in what we call the attribute space. This ensures a more general setting, since strings and long term secrets are part of puzzles that assure additional properties, such as unforgeability, etc. Thus in the simpler case where one does not want to ensure any additional property, the attributes can be set to null. We use the symbol $\perp$ to indicate the null attribute. The attributes can also be used to simulate secret keys, if these are used in the construction of the protocol.

Definition 1 (Cryptographic puzzle). *Let dSpace denote the space of difficulty levels, pSpace the puzzle space, sSpace the solution space and aSpace the attribute space. A cryptographic puzzle, or alternatively client puzzle,* CPuz *is a quadruple of publicly known algorithms* $\{\mathsf{Setup}, \mathsf{Gen}, \mathsf{Find}, \mathsf{Ver}\}$ *including fixed spaces* $\{pSpace, sSpace, aSpace\}$ *with the following descriptions:*

- $\mathsf{Setup}(1^k)$ *is the setup algorithm that takes as input a security parameter* 1^k *and fixes puzzle attributes* $\mathsf{atr} \in aSpace$*;*
- $\mathsf{Gen}(d, \mathsf{atr})$ *is the puzzle generation algorithm, it takes as input the difficulty of the puzzle to be created* $d \in dSpace$ *and a list of attributes* $\mathsf{atr} \in aSpace$ *then outputs a puzzle instance* $\mathsf{puz} \in pSpace$*,*
- $\mathsf{Find}(\mathsf{puz}, t)$ *is the solving algorithm that takes as input a puzzle* $\mathsf{puz} \in pSpace$ *and the maximum number of steps* t *that is allowed to perform, then outputs a solution* $\mathsf{sol} \in sSpace \cup \{\perp\}$ *(where* $\perp$ *is for the case when a solution could not be found in* t *steps),*
- $\mathsf{Ver}(\mathsf{puz}, \mathsf{sol}')$ *is the verification algorithm that takes as input a potential solution* $\mathsf{sol}' \in sSpace$ *and a puzzle* $\mathsf{puz} \in pSpace$ *and outputs* 1 *if and only if* sol' *is a correct solution for puzzle* puz *and* 0 *otherwise.*

For soundness, we require that puz is the input necessary and sufficient to successfully run the Find algorithm and that for any sol that is output of Find the verification holds, i.e., $\forall \mathsf{puz} \leftarrow \mathsf{Gen}(d, \mathsf{atr}), \exists t$ such that $1 \leftarrow \mathsf{Ver}(\mathsf{puz}, \mathsf{Find}(\mathsf{puz}, t))$.

By this, we force that one cannot produce a puzzle construction that is impossible to solve either because the information is not sufficient or the puzzle has no solution.

Remark 1. To simplify our definition we assumed that all spaces are fixed since this covers all puzzle constructions that we study here. If required by a more sophisticated construction, then it is possible to let Setup output the description of these spaces along with the description of the algorithms.

Remark 2. In [28] Ver also takes as input the secret master key s that can be used as trapdoor in case of puzzles that are not publicly verifiable (indeed, in some cases this trapdoor is needed for efficient verification, e.g., the time-lock puzzle). Our definition does not account for this possibility since we deal with puzzles that do not have such trapdoors. Extending the definition is straight forward by adding the master secret to the input of the verification algorithm Ver and embedding it in the attributes string atr that is output by the setup algorithm Setup (the difficulty notions that we study are not affected by these changes).

Remark 3. The puzzle is generic and can be further augmented with other algorithms to ensure additional properties. For example one can add the Auth algorithm to verify authenticity for the case of unforgeable puzzles as in [9], etc.

REMARK 4. On purpose, we did not specify any detail on the runtime of Gen, Find and Ver algorithms. This is because we wanted to keep the definition as generic as possible as it addresses puzzle in general. For practical purposes, one can request that all four algorithms work in probabilistic polynomial time.

We now exemplify the definition above on the HashTrail and HashInversion puzzles.

HASHTRAIL PUZZLE. Let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^k$ be a publicly known hash function and fix $dSpace = [1,k], pSpace = \{0,1\}^* \times \{0,1\}^k$ and $sSpace = \{0,1\}^*$. The $\mathsf{HashTrail}$ puzzle is a quadruple of algorithms:

- $\mathsf{Setup}(1^k)$ is the setup algorithm that on input 1^k outputs $\mathsf{atr} = \perp$,
- $\mathsf{Gen}(d)$ is the generation algorithm which on input d randomly chooses $r \in \{0,1\}^k$ and outputs puzzle instance $\mathsf{puz} = \{d, r\}$,
- $\mathsf{Find}(\mathsf{puz}, t)$ is the solving algorithm that on input puz and the number of steps t iteratively samples $\mathsf{sol} \in [0, t)$ until $\mathcal{H}(r||\mathsf{sol})_{1..d} = 0$,
- $\mathsf{Ver}(\mathsf{puz}, \mathsf{sol})$ is the algorithm that takes $\mathsf{puz}, \mathsf{sol}$ as input and returns 1 if $\mathcal{H}(r||\mathsf{sol})_{1..d} = 0$ and 0 otherwise.

HASHINVERSION PUZZLE. Let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^k$ be a publicly known hash function and fix $dSpace = [1,k], pSpace = \{0,1\}^* \times \{0,1\}^k$ and $sSpace = \{0,1\}^*$. The $\mathsf{HashInv}$ puzzle is the quadruple of algorithms:

- $\mathsf{Setup}(1^k)$ is the setup algorithm that on input 1^k outputs $\mathsf{atr} = \perp$,
- $\mathsf{Gen}(d)$ is the puzzle generation algorithm which on input d randomly chooses $x \in \{0,1\}^k$, computes $\mathcal{H}(x)$, sets x' as the first d bits of x and x'' as the remaining bits and outputs puzzle instance $\mathsf{puz} = \{d, x'', \mathcal{H}(x)\}$,
- $\mathsf{Find}(\mathsf{puz})$ is the solving algorithm that on input puz and the number of steps t iteratively samples at most t values $\mathsf{sol} \in \{0,1\}^d$ until $\mathcal{H}(\mathsf{sol}||x'') = \mathcal{H}(x)$,
- $\mathsf{Ver}(\mathsf{puz}, \mathsf{sol})$ be the algorithm that takes $\mathsf{puz}, \mathsf{sol}$ as input and returns 1 if $\mathcal{H}(\mathsf{sol}||x'') = \mathcal{H}(x)$ and 0 otherwise.

REMARK 5. We set $\mathsf{atr} = \perp$ since, while we do study difficulty notions for these puzzles, we are not interested in their unforgeability and thus including a secret key in the puzzle construction will lead only to unnecessary complications.

3.2 Difficulty preservation, optimality and fairness

We formalize puzzle difficulty using a game in which the adversary $\mathcal{A}$ is allowed to get as many puzzles and their solutions from the challenger $\mathcal{C}$ and later needs to find solutions for one or more puzzles generated by the challenger.

The following games make use of the following two oracles: $\mathsf{OGenSolve}$ and OTest, the former is used to generate puzzles and solutions (by running Gen and subsequently if needed Find for the required number of steps) while the later is used to output target puzzles (by running Gen). We do not stress whether the adversary $\mathcal{A}$ runs $\mathsf{OGenSolve}$ on its own or these are simulated by the challenger $\mathcal{C}$ as we do not distinguish here between interactive and non-interactive puzzles (puzzles that are generated by the solver or the challenger). We defer such specific details for the security proof of each particular puzzle that we analyze.

CONCURRENT SOLVING GAME (CS). For any fixed parameters k, d, n we define the concurrent puzzle solving game $\mathsf{Exec}^{\mathsf{CS/CPuz}}_{\mathcal{A},k,d,n}(q_{\mathsf{Gen}}, t)$ as the following four stage game between challenger $\mathcal{C}$ and adversary $\mathcal{A}$:

1. challenger $\mathcal{C}$ runs Setup on input 1^k to get $\mathsf{atr} \in aSpace$,
2. adversary $\mathcal{A}$ is allowed to make q_{Gen} queries to oracle $\mathsf{OGenSolve}$ which returns each time a puzzle and its corresponding solution, i.e., $\{\mathsf{puz}, \mathsf{sol}\}$, and n queries to oracle OTest which on each invocation generates and returns a target puzzle $\mathsf{puz}^\diamondsuit$,

3. after t steps adversary $\mathcal{A}$ outputs a set of solutions $\{\mathsf{sol}_1^\diamondsuit, \mathsf{sol}_2^\diamondsuit, ..., \mathsf{sol}_n^\diamondsuit\}$ for puzzles $\{\mathsf{puz}_1^\diamondsuit, \mathsf{puz}_2^\diamondsuit, ..., \mathsf{puz}_n^\diamondsuit\}$ that were returned by oracle OTest,
4. challenger $\mathcal{C}$ queries Ver on all puzzles and solutions output from adversary $\mathcal{A}$ and returns 1 if all solutions are correct else returns 0.

Remark 6. The exact definition for each computational step done by the adversary in stage 3 of the game strictly relies on the computational model behind the puzzle (for example one step can be the computation of one hash function or one modular squaring, etc.). Therefore, the precise definition of each computational step from stage 3 of the game is deferred for the proof related to each of the puzzles that we analyze.

Remark 7. By using the factor q_{Gen}, in addition to previous hardness definitions, we allow collisions in the generation algorithm, that is, we do not exclude that the same puzzle can be outputted more than once. Generally, collisions appear as a negligible factor in the hardness bound, but this factor is relevant as the examples from the introductory section showed.

Sequential solving game (SS). For any fixed parameters k, d, n we define the sequential solving game $\mathsf{Exec}^{\mathsf{SS}/\mathsf{CPuz}}_{\mathcal{A},k,d,n}(q_{\mathsf{Gen}}, t)$ as the following four stage game between challenger $\mathcal{C}$ and adversary $\mathcal{A}$ comprised of n isolated adversaries $\mathcal{A}_i, i = 1..n$ each performing t_i steps and $q_{\mathsf{Gen},i}$ queries:

1. similarly to step 1 of the *concurrent solving game* challenger $\mathcal{C}$ runs Setup on input 1^k to get $\mathsf{atr} \in \mathit{aSpace}$,
2. for $i = 1..n$
   (a) adversary $\mathcal{A}_i$ is allowed to make $q_{\mathsf{Gen},i}$ queries to oracle $\mathsf{OGenSolve}$ which returns each time a puzzle and its corresponding solution, i.e., $\{\mathsf{puz}, \mathsf{sol}\}$, and 1 query to oracle OTest which generates and returns a target puzzle $\mathsf{puz}_i^\diamondsuit$,
   (b) after t_i steps adversary adversary $\mathcal{A}_i$ outputs the solution $\mathsf{sol}_i^\diamondsuit$ for the puzzle $\mathsf{puz}_i^\diamondsuit$ that was returned by oracle OTest,
3. challenger $\mathcal{C}$ verifies that $\mathcal{A}$ is a valid adversary by checking that $t = t_1 + ... + t_n$ and $q_{\mathsf{Gen}} = q_{\mathsf{Gen},1} + ... + q_{\mathsf{Gen},n}$ then queries Ver on all puzzles and solutions returned by adversaries $\mathcal{A}_i, i = 1..n$ and returns 1 if and only if all these solutions are correct else it returns 0.

Having $\Gamma \in \{\mathsf{CS}, \mathsf{SS}\}$ by $\mathsf{Win}^{\Gamma/\mathsf{CPuz}}_{\mathcal{A},k,d,n}(q_{\mathsf{Gen}}, t)$ we denote the probability of the *winning* event which is the event in which the adversary outputs a correct solution for the puzzles and the game $\mathsf{Exec}^{\Gamma/\mathsf{CPuz}}_{\mathcal{A},k,d,n}(q_{\mathsf{Gen}}, t)$ returns 1, i.e.,

$$\mathsf{Win}^{\Gamma/\mathsf{CPuz}}_{\mathcal{A},k,d,n}(q_{\mathsf{Gen}}, t) = \Pr\left[\mathsf{Exec}^{\mathsf{SS}/\mathsf{CPuz}}_{\mathcal{A},k,d,n}(q_{\mathsf{Gen}}, t) = 1\right]$$

Definition 2 (Difficulty bound). *For $\epsilon_{k,d,n} : \mathrm{N} \to [0, 1]$ a family of functions indexed by parameters k, d and n, we say that $\epsilon_{k,d,n}(\cdot)$ is a difficulty bound for a puzzle played in a puzzle solving game Γ/CPuz with $\Gamma \in \{\mathsf{CS}, \mathsf{SS}\}$, if for any adversary $\mathcal{A}$ it holds*

$$\mathsf{Win}^{\Gamma/\mathsf{CPuz}}_{\mathcal{A},k,d,n}(q_{\mathsf{Gen}}, t) \leq \epsilon_{k,d,n}(q_{\mathsf{Gen}}, t)$$

The difficulty bound allows us to define difficulty relations between puzzles. To best of our knowledge, this aspect was also overlooked by previous work on puzzle difficulty. In what follows, let us denote by $\nu(k, d)$ a negligible constant in k and d that can be set as small as needed by proper choice of the security parameter k and difficulty level of the puzzle d (subsequently, $\overline{\nu}(k, d)$ denotes a non-negligible value).

Definition 3 (Difficulty relations). *Given two puzzles $\mathsf{CPuz}^\diamondsuit$ and $\mathsf{CPuz}^\blacklozenge$ played independently in games $\Gamma^\diamondsuit$ and $\Gamma^\blacklozenge$, we say that:*

1. $\mathsf{CPuz}^{\diamondsuit}$ *in game* $\Gamma^{\diamondsuit}$ *is at least as difficult as* $\mathsf{CPuz}^{\blacklozenge}$ *in game* $\Gamma^{\blacklozenge}$ *and denote it as* $\Gamma^{\diamondsuit}/\mathsf{CPuz}^{\diamondsuit} \leqq_{\mathsf{Dif}} \Gamma^{\blacklozenge}/\mathsf{CPuz}^{\blacklozenge}$ *if there exists a difficulty bound* $\epsilon^{\diamondsuit}_{k,d,n}(q_{\mathsf{Gen}}, t)$ *of* $\Gamma^{\diamondsuit}/\mathsf{CPuz}^{\diamondsuit}$ *such that for any difficulty bound* $\epsilon^{\blacklozenge}_{k,d,n}(q_{\mathsf{Gen}}, t)$ *of* $\Gamma^{\blacklozenge}/\mathsf{CPuz}^{\blacklozenge}$ *it holds* $\epsilon^{\diamondsuit}_{k,d,n}(q_{\mathsf{Gen}}, t) \leq \epsilon^{\blacklozenge}_{k,d,n}(q_{\mathsf{Gen}}, t) + \nu(k, d)$,
2. $\mathsf{CPuz}^{\diamondsuit}$ *in game* $\Gamma^{\diamondsuit}$ *is more difficult than* $\mathsf{CPuz}^{\blacklozenge}$ *in game* $\Gamma^{\blacklozenge}$ *and denote it as* $\Gamma^{\diamondsuit}/\mathsf{CPuz}^{\diamondsuit} \ll_{\mathsf{Dif}} \Gamma^{\blacklozenge}/\mathsf{CPuz}^{\blacklozenge}$ *if there exists a difficulty bound* $\epsilon^{\diamondsuit}_{k,d,n}(q_{\mathsf{Gen}}, t)$ *of* $\Gamma^{\diamondsuit}/\mathsf{CPuz}^{\diamondsuit}$ *such that for any difficulty bound* $\epsilon^{\blacklozenge}_{k,d,n}(q_{\mathsf{Gen}}, t)$ *of* $\Gamma^{\blacklozenge}/\mathsf{CPuz}^{\blacklozenge}$ *it holds* $\epsilon^{\diamondsuit}_{k,d,n}(q_{\mathsf{Gen}}, t) + \overline{\nu}(k, d) \leq \epsilon^{\blacklozenge}_{k,d,n}(q_{\mathsf{Gen}}, t)$,
3. $\mathsf{CPuz}^{\diamondsuit}$ *in game* $\Gamma^{\diamondsuit}$ *and* $\mathsf{CPuz}^{\blacklozenge}$ *in game* $\Gamma^{\blacklozenge}$ *are equally difficult and denote it as* $\Gamma^{\diamondsuit}/\mathsf{CPuz}^{\diamondsuit} \cong_{\mathsf{Dif}} \Gamma^{\blacklozenge}/\mathsf{CPuz}^{\blacklozenge}$ *if* $\Gamma^{\diamondsuit}/\mathsf{CPuz}^{\diamondsuit} \leqq_{\mathsf{Dif}} \Gamma^{\blacklozenge}/\mathsf{CPuz}^{\blacklozenge}$ *and* $\Gamma^{\blacklozenge}/\mathsf{CPuz}^{\blacklozenge} \leqq_{\mathsf{Dif}} \Gamma^{\diamondsuit}/\mathsf{CPuz}^{\diamondsuit}$.

The difficulty relations allow us now to define difficulty preserving puzzles which would require the existence of a difficulty bound of the concurrent solving game which is upper bounded by any difficulty bound of the sequential solving game plus some negligible factor (this invariantly makes the concurrent solving game and the sequential solving game equally difficult).

Definition 4 (Difficulty preserving puzzle). *We say that a puzzle* CPuz *is difficulty preserving if* CPuz *in the sequential solving game is equally difficult to* CPuz *in the concurrent solving game, i.e,* $\mathsf{SS}/\mathsf{CPuz} \cong_{\mathsf{Dif}} \mathsf{CS}/\mathsf{CPuz}$.

Remark 8. For any adversary in the sequential solving game there exists an adversary in the concurrent solving game that wins with probability at least equal to it. This is obvious since at worst the adversary in the concurrent solving game can work as a challenger for the adversary in the sequential solving game. Thus, any puzzle in the sequential solving game is at least as hard as in the concurrent solving game.

Remark 9. In $\mathsf{Exec}^{\Gamma/\mathsf{CPuz}}_{\mathcal{A},k,d,n}(q_{\mathsf{Gen}}, t)$ we assumed puzzles of the same difficulty level. It is easy however to extend this definition to puzzles of various difficulty levels as well. This can be done by replacing in both the sequential and concurrent games the value of d with a vector containing various difficulty levels, e.g., $d =< d_1, d_2, ..., d_n >$. The difficulty preserving condition will now simply enforce that for the same set of difficulty levels the concurrent solving game is as hard as the sequential solving game. Such an extension to puzzles of multiple difficulty levels does not appear to be possible with the definition from [28] since multiple puzzle difficulty is linked inextricably to single puzzle difficulty, but for precisely the same difficulty parameter d.

In what follows, we define the security properties of puzzles by comparing the success of an adversary in solving them with that of an honest party that simply runs the Find algorithm. Below we clarify what is the average and the worst case solving time by such an honest party; we start with the former. We write $\mathsf{Exec}^{\mathsf{CPuz}}_{\mathsf{Find},k,d,n}(t)$ for the random variable obtained by executing the experiment defined above with a "benign" adversary who for each puzzle that it obtains as challenge it solves it using the Find algorithm (note that q_{Gen} is missing since this parameter is irrelevant for the solving algorithm). The following definition captures the average probability of solving n puzzles of difficulty d in time t.

Definition 5 (Find bound). *For a given* CPuz *we denote by* $\zeta^{\mathsf{CPuz}}_{k,d,n}(t)$ *the probability that* Find *correctly finishes in at most t steps, i.e.,* $\zeta^{\mathsf{CPuz}}_{k,d,n}(t) = \Pr\left[\mathsf{Exec}^{\mathsf{CPuz}}_{\mathsf{Find},k,d,n}(t) = 1\right]$.

The next definition identifies the maximum number of steps needed by the Find algorithm to solve n puzzles with probability 1. Note that since the find bound $\zeta^{\mathsf{CPuz}}_{k,d,n}(t)$ comes from game $\mathsf{Exec}^{\mathsf{CPuz}}_{\mathsf{Find},k,d,n}(t)$ the same probability distribution as in the case of the solving games is assumed for the puzzles instances.

Definition 6 (Maximum solving time). *For a given* CPuz *the maximum solving time of* CPuz *is* $\mathrm{t}_{\max}$ *if* $\mathrm{t}_{\max}$ *is the minimum number of steps at which* $\zeta^{\mathsf{CPuz}}_{k,d,n}(t)$ *is 1, i.e.,* $\zeta^{\mathsf{CPuz}}_{k,d,n}(\mathrm{t}_{\max}) = 1, \zeta^{\mathsf{CPuz}}_{k,d,n}(\mathrm{t}'_{\max}) < 1, \forall \mathrm{t}'_{\max} < \mathrm{t}_{\max}$.

Definition 7 (Average solving time). *For a given* CPuz *we define the average solving time as the average number of steps required by* Find*, i.e.,*

$$\mathrm{t}_{\mathrm{avr}}(k,n,d) = \sum_{i=1,\mathrm{t}_{\max}} i \cdot \left[\zeta^{\mathsf{CPuz}}_{k,d,n}(i) - \zeta^{\mathsf{CPuz}}_{k,d,n}(i-1)\right].$$

Example 1. Consider for example the HashTrail puzzle. If one considers the hash function to be simulated by a random oracle, we have $\mathrm{t}_{\max} = \infty$ and $\mathrm{t}_{\mathrm{avr}} = 2^d$. Thus, there are puzzles for which $\mathrm{t}_{\max}$ is infinite while $\mathrm{t}_{\mathrm{avr}}$ is finite. On the contrary for the HashInversion puzzle both the maximum solving time and average solving time are finite since we have $\mathrm{t}_{\max} = 2^d$ and $\mathrm{t}_{\mathrm{avr}} = 2^{d-1}$.

Definition 8 (Optimal puzzle). *We say that* CPuz *is optimal if at any number of steps and any number of puzzles the success probability of any adversary is upper bounded by the success probability of the solving algorithm of the puzzle plus some negligible factor in the difficulty level and security parameter, i.e.,* $\forall t, n, \epsilon_{k,d,n}(q_{\mathsf{Gen}}, t) \leq \zeta^{\mathsf{CPuz}}_{k,d,n}(t) + \nu(k,d)$.

The next proposition establishes the link between optimal puzzles and difficulty preserving puzzles showing that optimality is the desired property.

Proposition 1. *Assume multiple puzzles are solved through* Find *in the usual way by independently running* Find *on each of the puzzles, which inextricably makes the average solving time for* n *puzzles of difficulty* d *to be* n *times the average solving time for a puzzle of difficulty 1, i.e.,* $\forall n, d, \mathrm{t}_{\mathrm{avr}}(k,d,n) = n \cdot \mathrm{t}_{\mathrm{avr}}(k,d,1)$. *If* CPuz *is optimal then* CPuz *is difficulty preserving.*

To verify the statement in Proposition 1 we need to show that $\mathsf{SS}/\mathsf{CPuz} \cong_{\mathsf{Dif}} \mathsf{CS}/\mathsf{CPuz}$. The sequential solving game is obviously at least as hard as the concurrent solving game $\mathsf{SS}/\mathsf{CPuz} \leqq_{\mathsf{Dif}} \mathsf{CS}/\mathsf{CPuz}$. Since the puzzle is optimal we have $\epsilon_{k,d,n}(q_{\mathsf{Gen}}, t) \leq \zeta^{\mathsf{CPuz}}_{k,d,n}(t) + \nu(k,d)$ and thus there exists a difficulty bound of the concurrent game which is smaller than the bound of the solving algorithm $\zeta^{\mathsf{CPuz}}_{k,d,n}(t)$ plus negligible. We show that any bound of the sequential solving game is even bigger than $\zeta^{\mathsf{CPuz}}_{k,d,n}(t)$ with some negligible an thus $\mathsf{CS}/\mathsf{CPuz} \leqq_{\mathsf{Dif}} \mathsf{SS}/\mathsf{CPuz}$. For this, assume an adversary in the sequential that simply runs Find at each iteration and outputs some random solution for each puzzle that Find failed to solve in the predefined number of steps. This adversary upper bounds Find since it has the same probability to win as Find plus some negligible factor that it guessed the solution (by returning some random value) which completes our argument.

Remark 10. The optimality condition $\epsilon_{k,d,n}(q_{\mathsf{Gen}}, t) \leq \zeta^{\mathsf{CPuz}}_{k,d,n}(t) + \nu(k,d)$ ensures that the bound from the concurrent solving game, i.e., $\epsilon_{k,d,n}(q_{\mathsf{Gen}}, t)$, is $\nu(k,d)$ tight.

Remark 11. The condition $\epsilon_{k,d,n}(t) \leq \epsilon_{k,d,1}(t/n)$ is enough to ensure that an optimal puzzle, i.e., a puzzle for which $\forall n, d, |\epsilon_{k,d,n}(q_{\mathsf{Gen}}, t) - \zeta^{\mathsf{CPuz}}_{k,d,n}(t)| \leq \nu(k,d)$, is difficulty preserving. This is trivial to prove, but it seems that the condition $\epsilon_{k,d,n}(t) \leq \epsilon_{k,d,1}(t/n)$ is not so trivial since none of the puzzles that we analyze next satisfies it (one could easily plot the difficulty bounds to verify this).

Fairness of the first kind or *fairness in answering* means that an adversary that failed to solve the puzzle cannot lie about this while *fairness of the second kind* or *fairness in solving* means that the adversary cannot win against the puzzle in less steps than $\mathrm{t}_{\max}$ (all these up to some negligible probability in the difficulty and security parameters).

Definition 9 (Fairness of first and second kind). *Let* $\widetilde{\mathcal{A}}$ *denote a semi-honest adversary that in game* $\Gamma \in \{\mathsf{CS}, \mathsf{SS}\}$ *plays with the following restriction: in step 3 of the concurrent solving game (or 2.b of the sequential solving game) he is allowed just to run* Find *for* t *steps. Let* $\neg\mathsf{Solved}^{\Gamma/\mathsf{CPuz}^{\diamondsuit}}_{\widetilde{\mathcal{A}},k,d,n}(q_{\mathsf{Gen}}, t)$ *denote the event that* Find *didn't return a solution for the puzzle instance* $\mathsf{CPuz}^{\diamondsuit}$ *returned by the* OTest *oracle in step 2. We say that puzzle* CPuz *played in game* $\Gamma \in \{\mathsf{CS}, \mathsf{SS}\}$ *has:*

1. *fairness of the first kind (or fairness in answering) if for any puzzle instance* $\mathsf{CPuz}^{\diamondsuit}$ *the probability that the adversary wins given that he failed to solve the puzzle in t steps is bounded by a negligible constant in the difficulty parameter and security level, i.e.,*

$$\Pr\left[\mathsf{Win}^{\Gamma/\mathsf{CPuz}^{\diamondsuit}}_{\tilde{\mathcal{A}},k,d,n}(q_{\mathsf{Gen}},t)\middle|\neg\mathsf{Solved}^{\Gamma/\mathsf{CPuz}^{\diamondsuit}}_{\tilde{\mathcal{A}},k,d,n}(q_{\mathsf{Gen}},t)\right] \le \nu(k,d), \forall t < \mathrm{t}_{\max}$$

2. *fairness of the second kind (or fairness in solving) if the probability that the adversary wins in less than* $\mathrm{t}_{\max}$ *steps is bounded by a negligible constant in the difficulty parameter and security level, i.e.,*

$$\Pr\left[\mathsf{Win}^{\Gamma/\mathsf{CPuz}}_{\mathcal{A},k,d,n}(q_{\mathsf{Gen}},t)\right] \le \nu(k,d), \forall t < \mathrm{t}_{\max}$$

Example 2. To clarify these notions, consider again the HashTrail and HashInversion puzzles. HashTrail has *fairness of the first kind* since if the adversary failed to ask the hashing oracle an input for which the output has d trailing zeroes then the probability to guess such an input, after any number of failed steps, is still 2^d. In the case of the HashInversion puzzle this doesn't hold since after t calls to the hashing oracle the probability to guess the solution is $(2^d - t)^{-1}$ which is non-constant and non-negligible in the difficulty level. Thus the HashInversion puzzle doesn't have fairness in answering. None of these two puzzles has fairness in solving since the probability to solve them increases at each step toward a non-negligible value. The time lock puzzle on the other hand has *fairness in solving* and implicitly *fairness in answering.* By definition, fairness of the second kind implies fairness of the first kind.

Remark 12. If CPuz has fairness in solving then the average solving time equals the maximum solving time up to some negligible value in the security parameter k and difficulty level d, i.e., $\forall n, d$, $|\mathrm{t}_{\mathrm{avr}}(k,d,n) - \mathrm{t}_{\max}(k,d,n)| \le \nu(k,d)$.

3.3 New difficulty bounds for HashTrail and HashInversion

We now establish tight security bounds for the HashInversion and HashTrail puzzles. The proofs of the following theorems can be found in Appendix B.

Theorem 1. *In the random oracle model, the HashTrail puzzle is optimal and difficulty preserving with* $\mathrm{t}_{\mathrm{avr}}(k,d,1) = 2^d$, $\mathrm{t}_{\max}(k,d,1) = \infty$ *and solving and difficulty bounds:*

$$\zeta^{HT}_{k,d,n}(t) = \sum_{i=n,t} \binom{i-1}{n-1} \cdot \frac{1}{2^{nd}} \cdot \left(1 - \frac{1}{2^d}\right)^{i-n}, \qquad \epsilon^{HT}_{k,d,n}(t) \le \zeta^{HT}_{k,d,n}(t) + \frac{1}{2^d - 1} + \frac{q^2_{\mathsf{Gen}}}{2^{k+1}}.$$

Remark 13. For HashTrail, in [28] the advantage is upper bounded by $\frac{q+n}{n \cdot 2^d}$ using Markov inequality - obviously, $\frac{q}{2^d}$ is a bound for the probability to solve 1 puzzle in q queries and dividing it with n gives a bound of the probability for n instances. While such a bound is easy to prove, Figure 2 shows how loose this is compared to the advantage from the previous theorem for a small numerical example. In section 2 we showed that loose bounds cannot say much about the difficulty of solving multiple puzzles.

Theorem 2. *Let $[z^i]P(z)$ denote the coefficient of z^i in the expansion of polynomial $P(z)$. In the random oracle model, the HashInversion puzzle is optimal and difficulty preserving with* $\mathrm{t}_{\mathrm{avr}}(k,d,1) = 2^{d-1}$, $\mathrm{t}_{\max}(k,d,1) = 2^d$ *and solving and difficulty bounds:*

$$\zeta^{HI}_{k,d,n}(t) = \sum_{i=n,t} [z^i] \left(z \cdot \frac{1 - z^{2^d}}{1 - z}\right)^n \cdot \frac{1}{2^{nd}}, \qquad \epsilon^{HI}_{k,d,n}(t) \le \zeta^{HI}_{k,d,n}(t) + \frac{n}{2^d} + \frac{q^2_{\mathsf{Gen}}}{2^{k-d+1}}.$$

Remark 14. In [28] the advantage of HashInversion is upper bounded by $\left(\frac{q+n}{n\cdot 2^d}\right)^n$. As Figure 2 shows for a small numerical example, the advantage of the solving algorithm from the previous theorem is much bigger, thus the bound in [28] is wrong.

Remark 15. The bounds from Theorem 1 and Theorem 2 are not suitable for computation at large values of the difficulty parameters d or n. Nonetheless, we can manipulate them as abstract functions to prove (see Appendix B) that difficulty is preserved for multiple puzzle instances while relying on approximations that are not tight enough is not reliable within our framework. Nevertheless, to make computations feasible we claim the following approximations of the bounds:

$$\epsilon^{HT}_{k,d,n}(t) \leq \left[1 - \left(1 - \frac{1}{2^d}\right)^{t+1}\right]^n + \frac{1}{2^d - 1} + \frac{q^2_{\mathsf{Gen}}}{2^{k+1}}$$

,

$$\epsilon^{HI}_{k,d,n}(t) \leq \frac{1}{n}\left(\frac{t-n+1}{2^d}\right)^n + \frac{n}{2^d} + \frac{q^2_{\mathsf{Gen}}}{2^{k-d+1}}.$$

.

Figure 3 shows a graphical depiction of the approximate bounds compared to the tight bounds of the previous two theorems and the bounds in [28] (also see Fig. 2 for the previous two bounds). Obtaining tighter approximations may be subject of future work for us.

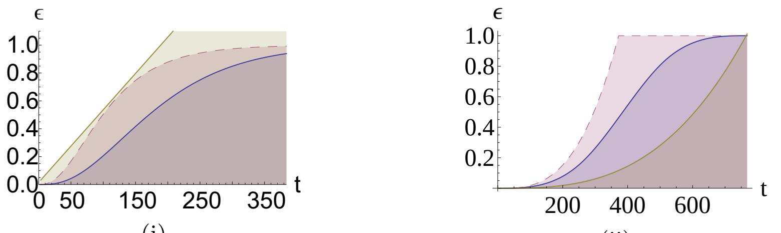

Fig. 3: Graphical depiction of the approximate bounds (dotted line) vs. the tighter bounds and the bounds from Stebila et al. at $n = 3$, $d = 8$ for HashTrail (i) and HashInversion (ii) puzzles

4 DoS resilience

Defining resilience against resource exhaustion DoS is a non-trivial task that requires subtle analysis of the costs incurred by the computational steps done on the server side. In practice, choosing the right amount of work that needs to be done in order to gain access to a particular resource on the server side is a matter of protocol engineering, rather than cryptography. Notably, as the server resources are always limited, when the number of honest clients exceeds the total amount of resources, resource exhaustion is unavoidable. Thus from the protocol design, the best one could do is to hinder an adversary from claiming resources in the name of potentially many honest participants - this is were proof-of-work comes into action.

We assume the generic construction in which a protocol is prefixed by three rounds in which the client requests and receives the puzzle, then sends the solution to the server, and denote this client puzzle protocol as $\Pi(\mathsf{CPuz})$. Two main conditions must be met by the puzzle in order to make the protocol secure: first, the puzzles must be unforgeable (Chen et al. [9]), otherwise an adversary may reply with solution for easier puzzles (generated by himself), second, the puzzles must be difficulty preserving (Stebila et al. [28]), otherwise an adversary may solve more puzzles easier. Unforgeability is essential in assuring DoS resilience. In [9] it is observed for the first time that the client puzzles of Jules and Brainard [20] are not unforgeable and this can cause a DDoS attack. Later, in [28] it is outlined that unforgeability is not an intrinsic property of a puzzle since non-interactive puzzles cannot be produced as unforgeable (as they are generated by the client himself). But in the same work, for the DoS countermeasure protocol the puzzle itself is augmented with a MAC computed with the server secret key which achieves essentially the same objective as unforgeability. A third condition which must be met is that for any solution provided by some client it must not be possible by the adversary to claim the work as his own (this condition is already present in [28] and several other lines of work). Although this is usually not defined as a specific property for a puzzle, puzzles protocols are usually designed in such way that the solution is bound with the solver's identity so that it cannot be stolen. Indeed, there are scenarios in which this is not possible. For example consider that the client identity is an IP address, as long as IPs can be spoofed by adversaries it will not be possible to protect the clients work (given that a secure channel between clients and server does not exist).

Figure 4 depicts a generic client puzzle protocol. For technical reasons we assume the existence of a secure timer on the server side.

4.1 DoS resilience and puzzle difficulty

A WEAKNESS OF AN EXISTING APPROACH. The only existing formal definition of computational security for DoS resilience is due to Stebila et al. and builds directly on the difficulty of puzzle systems [28]. Essentially, it requires that an adversary cannot claim more resources than the number of puzzles he is able to solve in its permitted running time. As specified in [28], the definition of DoS resilience means in fact ϵ hardness, that is, an adversary can finish the protocol with probability ϵ given that he spent a predefined number of computational steps. From this perspective, the definition from [28] is correct, but is only part of the story. The reasons is that DoS resistance means something more, namely, that an adversary will not be able to consume all the responder resources (e.g., all available connections at some instant in time or during a particular interval). The problem with the definition in [28] is that it disregards an important aspect of puzzle defense against DoS, namely puzzle management. Puzzles used for DoS resilience come with an expiration time to avoid what we call a *next day attack* where an adversary first spends large amounts of resources to solve a large amount of puzzles and later uses their solutions to claim the corresponding resources in a much shorter interval. Although management can be built on top of some puzzle defense schemes, strictly speaking the definition of [28] allows for next day attacks as the execution that is considered looks directly at how many puzzles an adversary can solve in time t (and this amount is bounded by puzzle difficulty). The definition itself does not account for the possibility that the puzzles sent to claim resources could have been solved earlier. Introducing puzzle expiry, as a parameter in the adversary running time, seems to be the only reasonable way to tackle the question of DoS resilience and more, to determine precise bounds on the effectiveness of puzzles. Without it, the DoS-resilience definition will at least fail to send the correct application message while at worst it will not allow bridging with the efficiency limitations that we prove in what follows.

OUR APPROACH. To prevent such attacks we propose two measures: first we introduce a fixed lifetime for the puzzles, then we define resilience as a condition that must hold in any time interval $[t_2, t_1]$ and not just for an adversary having runtime t. By $\Pi(\mathsf{CPuz}, t_{puz})$ we denote a protocol Π that is protected by puzzles generated by CPuz and with lifetime t_{puz}, i.e., the protocol deems as invalid any solution received later than t_{puz} cycles after the generation of the corresponding puzzle. We stress that we do not consider the detailed cost of running the server program and we take as a premise that puzzles of difficulty d from CPuz are enough to protect the server.

PROTOCOL ATTACK GAME. We define the attack game $\mathsf{Exec}^{\Pi}_{\mathcal{A}}(\mathsf{CPuz}, k, d, n, t_{puz})$ based on a two stage adversary. First adversary $\mathcal{A}_1$ is allowed to interact with the server and honest clients via: (1) $\mathsf{RequestPuz}(\mathsf{str})$

on which the server answers with a new fresh instance puz, (2) $\mathsf{SolvePuz}(\mathsf{puz})$ on which any client answers with a solution sol. Then $\mathcal{A}_1$ outputs state information $\mathsf{state}_{\mathcal{A}_1}$ to $\mathcal{A}_2$ which is allowed to do the same actions subject only to one restriction: t_1 marks the time at which $\mathcal{A}_1$ has send its state information and at time $t_2 + t_{puz}$ adversary $\mathcal{A}_2$ must output the solutions to n puzzles created no sooner than t_1. The game returns 1 if the adversary has returned correct solutions for all n puzzles, i.e.,

$$\mathsf{Win}_{\mathcal{A}}^{\Pi(\mathsf{CPuz},t_{puz},k,d,n)}(t_2 - t_1 + t_{puz}, n) = \Pr\left[\mathsf{Exec}_{\mathcal{A}}^{\Pi(\mathsf{CPuz},t_{puz},k,d,n)} = 1\right]$$

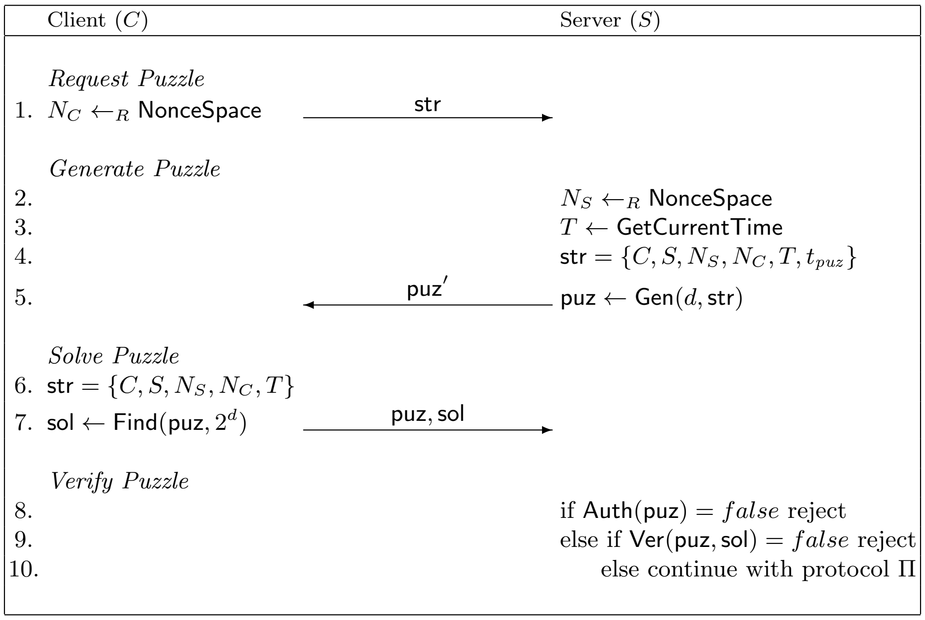
Client (C)
Server (S)
Request Puzzle
1. $N_C \leftarrow_R$ NonceSpace
str
Generate Puzzle
2. $N_S \leftarrow_R$ NonceSpace
3. $T \leftarrow$ GetCurrentTime
4. $\mathsf{str} = \{C, S, N_S, N_C, T, t_{puz}\}$
5. puz'
$\mathsf{puz} \leftarrow \mathsf{Gen}(d, \mathsf{str})$
Solve Puzzle
6. $\mathsf{str} = \{C, S, N_S, N_C, T\}$
7. $\mathsf{sol} \leftarrow \mathsf{Find}(\mathsf{puz}, 2^d)$
$\mathsf{puz}, \mathsf{sol}$
Verify Puzzle
8. if $\mathsf{Auth}(\mathsf{puz}) = false$ reject
9. else if $\mathsf{Ver}(\mathsf{puz}, \mathsf{sol}) = false$ reject
10. else continue with protocol Π

Fig. 4: Generic Client Puzzle Protocol $\Pi(\mathsf{CPuz})$

Definition 10 (DoS Resilience). *Let* CPuz *be an unforgeable, difficulty preserving puzzle. Protocol* $\Pi(\mathsf{CPuz}, t_{puz}, k, d, n)$ *is* $\epsilon_{k,d,n}$-DoS resilient *if for any* $t_1, t_2 \in [0, t_\Pi]$ *with* $t_1 < t_2$*, having an adversary* $\mathcal{A}$ *that can perform at most* $t_{\mathcal{A}}$ *computations in time* $t_2 - t_1 + t_{puz}$ *it holds:*

$$\Pr\left[\mathsf{Win}_{\mathcal{A}}^{\Pi(\mathsf{CPuz},t_{puz},k,d,n)}(t_2 - t_1 + t_{puz}, n)\right] \leq \epsilon_{k,d,n}(t_{\mathcal{A}}) + \nu(k)$$

Remark 16. Some comments on the parameters follow. Choosing d and t_{puz} obviously depends on the practical setting, here we assume these values are fixed. In practice, puzzle life-time should compensate both for the solving time, as well as for network delays. Thus, puzzle life-time gives bounds for both the maximum network delay as well as for the minimum computational power for a client to gain the resource.

Remark 17. The generic protocol must be $\epsilon_{k,d,n}$-*DoS resilient* if CPuz is unforgeable, difficulty preserving and if puzzle solutions cannot be spoofed. This can be easily proved since if the adversary manages to win higher probability then one could use the adversary to win against the puzzle solving game just by simulating the protocol game.

4.2 Limitations of practical schemes

While the previous definition is of theoretical interest, it can be translated to practical systems as well. The motivation behind the limitations that we point out here is in the controversy that still stays behind the usefulness of puzzles. While puzzles were proposed to combat spam by Dwork and Naor [12] more than two decades ago, spam is still an increasing problem and there is no practical large scale implementation with PoW against spam. It was shown in fact by the work of Laurie and Clayton [22], published more than a decade later, that puzzles may not work. A detailed economical analysis in done in [22] and the main argument appears to be that the computational cost at which puzzles are effective against spam becomes prohibitive for large lists operators. Although reactions occurred and several research papers showed that when used with other mechanisms (such as reputation systems) proof-of-work can actually work [23], up to this day there is no practical wide spread system that uses PoW for combating spam.

On the side DoS resilience, things are apparently different with a huge amount of proposals for augmenting protocols with client puzzles and practical experiments that show them to work, e.g., [3], TCP/IP [20], SSL/TLS [10], etc. Still there is no result in the spirit of Laurie and Clayton [22] to disapprove this. But despite a large number of practical proposals, there is little attention paid to the actual practical effectiveness of such mechanisms and a rigorous treatment on the parameters of such protocols is usually absent. Most of the proposals are based on common sense, e.g., when a server is under attack the hardness of the PoW is increased, and on empirical observations, e.g., the best protection is achieved when the difficulty is set to a certain threshold. But there is no rigorous treatment for some fundamental questions such as: when are PoWs effective and when they are not? Notable exceptions are more recent works which analyze the effectiveness of client puzzles in the rigorous framework provided by game theory [13], [24] and well before that [7]. Game theory is the right tool when one wants to address the revenue of the adversary but it is not always helpful since not all adversaries are motivated by the revenue. Malicious adversaries will simply want to lock the network regardless of the revenue. In what follows we a give simple answer to these questions using rules of thumb that can be straight forwardly applied to find limitations.

A MORE PRACTICAL VIEWPOINT. In practice, DoS is usually analyzed by means of queuing theory and the main parameter is service time $\theta_{service}$ which gives the maximum input rate that can be handled by the system. For example, if service time is $\theta_{service} = 10ms$ then the server can handle a maximum input rate $\lambda = 100$ connections each second and beyond this the systems gets saturated (leading to a waiting queue than can grow without bound). Thus we we will use a simple model which accounts for the arrival rate and computational power of the adversary. Then we analyze three commonly used PoW schemes: the basic PoW scheme (Figure 7), the filtering based PoW scheme which selects different difficulties for clients and adversaries (Figure 8) and the cascade PoW scheme which adds an initial PoW to the filtering (Figure 9). The filtering works by classifying the principals into honest and dishonest. While dishonest connections can be instantly dropped (which is supported by the computational model by increasing puzzle difficulty to ∞ in case of adversaries), there are still dishonest connections that come from the false negative rate of the filter β. Consequently, a β fraction of the connections are still of adversarial nature and they cannot be instantly dropped since they are taken as coming from honest clients. We denote these schemes as $\Pi^{\mathsf{Basic}}_{t_{puz},d_{\mathsf{init}}}(\mathsf{CPuz})$, $\Pi^{\mathsf{Filter}}_{t_{puz},d_{\mathcal{A}},d_{\mathcal{C}}}(\mathsf{CPuz})$ and $\Pi^{\mathsf{Filter}}_{t_{puz},d_{\mathcal{A}},d_{\mathcal{C}}}(\mathsf{CPuz})$ where each d denotes the difficulty level of the corresponding PoW (for simplicity we do skip the security parameter k from our notations since this analysis is done at a network level, assuming cryptography to be perfect). For each of these we outline practical bounds on effectiveness. Figure 5 outlines the parameters of our scenario and Figure 6 depicts honest clients and the adversary arriving at average rates of $\lambda_{\mathcal{C}}$ and $\lambda_{\mathcal{A}}$. We assume that the service is locked when the rate of arrivals exceeds the inverse of the serving time, i.e., $\theta^{-1}_{service}$ (which is the total number of resources that can be provided by the server). This means that the DoS condition is $\lambda_{\mathcal{A}} > \theta^{-1}_{service}$ (whenever the DoS is caused by the adversary alone).

In the proof of the following theorem (found in Appendix C) note that the lifetime of the puzzle t_{puz} from the $\epsilon_{k,d,n}$-*DoS resilience* is used to derive a practical bound that depends strictly on the computational resources of the participants and on the maximum acceptable load of the server $\theta^{-1}_{service}$. The theorem links the efficacy of a puzzle-based DoS defense system with the parameters of the underlying protocol. Informally, the theorem states that a puzzle scheme cannot protect a protocol when the ratio between the computational power of the adversary and that of the client exceeds the inverse of the service time (note that paradoxically this is independent on the hardness of the puzzle, an aspect that to best of our knowledge is overlooked in previous work). Further, filtering based schemes can compensate on this but only up to the false negative rate of the filter. As correctly noted in [8], hidden difficulty puzzles, i.e., puzzles for which the difficulty level remains hidden unless some computational power is commited to solve them, can also increase effectiveness but we underline a practical bound for these as well.

$\theta_{service}$	service time
$\lambda_{\mathcal{C}}$	arrival rate of clients
$\lambda_{\mathcal{A}}$	arrival rate of the adversary
$\pi_{\mathcal{C}}$	computational power of clients
$\pi_{\mathcal{A}}$	computational power of adversary
β	false negative rate of the filter
d_{init}	difficulty level of PoW-Init
$d_{\mathcal{C}}$	difficulty level of PoW-Client
$d_{\mathcal{A}}$	difficulty level of PoW-Adv
t_{puz}	puzzle lifetime

Fig. 5: Scenario parameters

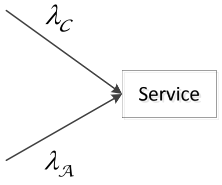

Fig. 6: Generic service model

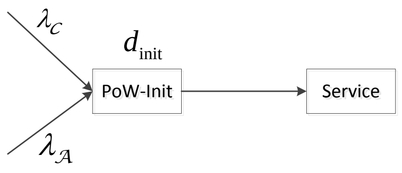

Fig. 7: The basic PoW scheme

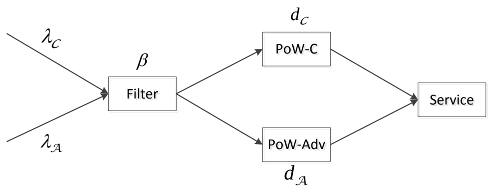

Fig. 8: The filtering based PoW scheme

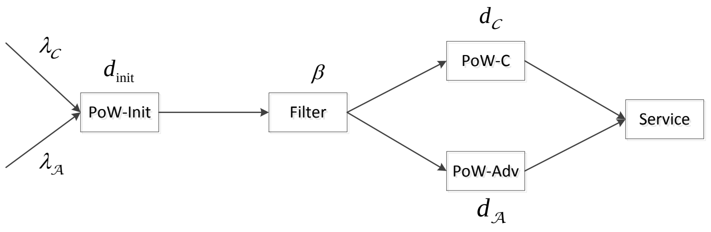

Fig. 9: The cascade PoW scheme

Theorem 3. *Consider server side has service time $\theta_{service}$ and the computational resources of the adversary and clients are $\pi_{\mathcal{A}}$ and $\pi_{\mathcal{C}}$ respectively and all these are set to a common reference with the difficulties, e.g., an adversary can compute $\pi_{\mathcal{A}}/d_{\mathsf{init}}$ PoWs of difficulty d_{init}. Then the following negative results hold regardless of puzzle difficulty in any of the PoWs: i) protocol $\Pi^{\mathsf{Basic}}_{t_{puz},d_{\mathsf{init}}}(\mathsf{CPuz})$ cannot provide DoS protection if $\pi_{\mathcal{A}} > \pi_{\mathcal{C}} \cdot \theta^{-1}_{service}$ and there are no benefits in increasing puzzle difficulty to more than $d = \pi_{\mathcal{A}}$, ii) protocol $\Pi^{\mathsf{Filter}}_{t_{puz},d_{\mathcal{A}},d_{\mathcal{C}}}(\mathsf{CPuz})$ cannot provide DoS protection if $\beta\lambda_{\mathcal{A}} > \theta^{-1}_{service}$ and* *iii) if puzzle difficulties are hidden in* $\Pi^{\mathsf{Filter}}_{t_{puz},d_{\mathcal{A}},d_{\mathcal{C}}}(\mathsf{CPuz})$*, having* $\beta\lambda_{\mathcal{A}} > \theta^{-1}_{service}$ *then* $\Pi^{\mathsf{Filter}}_{t_{puz},d_{\mathcal{A}},d_{\mathcal{C}}}(\mathsf{CPuz})$ *cannot provide protection if* $\beta\pi_{\mathcal{A}} > \pi_{\mathcal{C}} \cdot \theta^{-1}_{service}$ *and the same holds for the cascade protection scheme* $\Pi^{\mathsf{Cascade}}_{t_{puz},d_{\mathcal{A}},d_{\mathcal{C}},d_{\mathsf{init}}}(\mathsf{CPuz})$*.*

Remark 18. The bound outlined in i) seems to justify existing empirical results. Dean and Stubblefield [10] provided the first positive results for protecting SSL/TLS by using client puzzles. In the performance related section, the authors of [10] note that *20-bit puzzles* seem to offer the optimal level of protection. While this observation is only empirical, it is supported by the result of Theorem 3 which shows $d = \pi_{\mathcal{A}}$ as the maximum difficulty level and indeed in practice the computational power of an adversary is in the order of 2^{20} hashes per second. For distributed DoS attacks these values must be scaled up with the size of the bot-net that the adversary controls.

5 Conclusion

We refined difficulty notions for puzzles, bringing light on puzzles that are optimal, difficulty preserving and possess fairness in answering or solving. New difficulty bounds for two hash based puzzles are also provided. We showed that these bounds are tight enough to ensure optimality and that the puzzles are difficulty preserving while we also give reasonable approximations for these bounds to ease computation. Finally, we introduced a stronger definition for DoS resilience motivated by the observation that previous definitions may still allow an adversary to mount a successful attack. As this is the third paper proposing rigorous difficulty notions for client puzzles and showing that previous definitions fail, it is clear that formalizing puzzles properties is not as easy as it may appear on first sight.

Our definition opens the avenue of studying puzzles and their use in DoS defense in more detail than was possible in the past (e.g., by introducing new security notions and bounds on their effectiveness against DoS attacks). Previously, choosing puzzle difficulty in practice was only based on empirical observations, here we provided a clear upper bound for this as well as a bound on the usefulness of client puzzles against DoS. Namely, for the basic puzzle protection scheme which is most commonly employed, puzzles will work only if $\pi_{\mathcal{A}} < \pi_{\mathcal{C}} \cdot \theta^{-1}_{service}$ which places the computational power of the adversary and clients in a clear, crisp relation with network service time. But nevertheless this limits the practical use of puzzles due to disparities between the computational power of honest clients and malicious botnets controlled by adversaries. Current estimates place the average size of botnets in the order of tens of thousands of computers and this is just an average size (largest botnets reportedly reaching millions of computers). In such case, the basic PoW approach will almost certainly fail to protect while filters can help but only if they are good enough to compensate for such disparities. Our conclusion needs not be interpreted in a pessimistic sense. Formalizing puzzle properties still holds interesting open questions and relations to main-stream areas in cryptography, e.g., some of the security bounds that we determined are closely related to the recent work of Bellare et al. on multi-instance security [6]. However, for practical effectiveness of puzzles, basic approaches as commonly discussed in the literature appear to be insufficient and there is stringent need for more research in designing filters and calibrating puzzle difficulties for precise practical needs.

Acknowledgement. We thank Douglas Stebila and to the anonymous referees for their comments and feedback on our work. First author was partially supported by national research grants CNCSIS UEFISCDI, project number PNII IDEI 940/2008-2011 and POSDRU/21/1.5/G/13798, inside POSDRU Romania 2007-2013.

References

1. M. Abadi, M. Burrows, M. Manasse, and T. Wobber. Moderately hard, memory-bound functions. *ACM Transactions on Internet Technology*, 5:299–327, May 2005.
2. M. Abliz and T. Znati. A guided tour puzzle for denial of service prevention. In *Proceedings of the 2009 Annual Computer Security Applications Conference*, ACSAC '09, pages 279–288. IEEE Computer Society, 2009.
3. T. Aura, P. Nikander, and J. Leiwo. DoS-resistant authentication with client puzzles. In *Revised Papers from the 8th International Workshop on Security Protocols*, pages 170–177, London, UK, 2001. Springer-Verlag.
4. A. Back. Hashcash - a denial of service counter-measure. Technical report, 2002.
5. M. Bellare, R. Impagliazzo, and M. Naor. Does parallel repetition lower the error in computationally sound protocols. In *In Proceedings of 38th Annual Symposium on Foundations of Computer Science, IEEE*, pages 374–383. IEEE, 1997.
6. M. Bellare, T. Ristenpart, and S. Tessaro. Multi-Instance Security and its Application to Password-Based Cryptography. In *In Advances in CryptologyCRYPTO 2012*, pages 312-329. Springer, 2012.
7. B. Bencsáth, I. Vajda, and L. Buttyán. A game based analysis of the client puzzle approach to defend against dos attacks. In *Proceedings of SoftCOM*, volume 11, pages 763–767, 2003.
8. C. Boyd, J. Gonzalez-Nieto, L. Kuppusamy, H. Narasimhan, C. Rangan, J. Rangasamy, J. Smith, D. Stebila, and V. Varadarajan. An investigation into the detection and mitigation of denial of service (Dos) attacks: Critical information infrastructure protection. *Cryptographic Approaches to Denial-of-Service Resistance*, page 183, 2011.
9. L. Chen, P. Morrissey, N. P. Smart, and B. Warinschi. Security notions and generic constructions for client puzzles. In *Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '09, pages 505–523. Springer-Verlag, 2009.
10. D. Dean and A. Stubblefield. Using client puzzles to protect TLS. In *Proceedings of the 10th conference on USENIX Security Symposium - Volume 10*, SSYM'01, pages 1–1, Berkeley, CA, USA, 2001. USENIX Association.
11. C. Dwork, A. Goldberg, and M. Naor. On memory-bound functions for fighting spam. In *Proceedings of the 23rd Annual International Cryptology Conference*, pages 426–444. Springer-Verlag, 2003.
12. C. Dwork and M. Naor. Pricing via processing or combating junk mail. In *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, pages 139–147, London, UK, 1993. Springer-Verlag.
13. M. Fallah. A puzzle-based defense strategy against flooding attacks using game theory. *Dependable and Secure Computing, IEEE Transactions on*, 7(1):5–19, 2010.
14. Y. Gao. Efficient trapdoor-based client puzzle system against DoS attacks. Technical report, 2005.
15. Y. Gao, W. Susilo, Y. Mu, and J. Seberry. Efficient trapdoor-based client puzzle against DoS attacks. *Network Security*, pages 229–249, 2010.
16. B. Groza and B. Warinschi. Revisiting difficulty notions for client puzzles and DoS resilience. In *Information Security Conference (ISC)*. Springer-Verlag, 2012.
17. A. Jeckmans. Computational puzzles for spam reduction in SIP. draft, July 2007.
18. A. Jeckmans. Practical client puzzle from repeated squaring. Technical report, August 2009.
19. Y. I. Jerschow and M. Mauve. Non-parallelizable and non-interactive client puzzles from modular square roots. In *Sixth International Conference on Availability, Reliability and Security, ARES 2011*, pages 135–142, 2011.
20. A. Juels and J. Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *Proceedings of NDSS '99 (Networks and Distributed Security Systems)*, pages 151–165, 1999.
21. G. Karame and S. Čapkun. Low-cost client puzzles based on modular exponentiation. In *Proceedings of the 15th European conference on Research in computer security*, ESORICS'10, pages 679–697. Springer-Verlag, 2010.
22. B. Laurie and R. Clayton. Proof-of-work proves not to work; version 0.2, 2004.
23. D. Liu and L. Camp. Proof of work can work. In *Fifth Workshop on the Economics of Information Security*, 2006.
24. H. Narasimhan, V. Varadarajan, and C. Rangan. Game theoretic resistance to denial of service attacks using hidden difficulty puzzles. *Information Security, Practice and Experience*, pages 359–376, 2010.
25. J. Rangasamy, D. Stebila, C. Boyd, and J. Nieto. An integrated approach to cryptographic mitigation of denial-of-service attacks. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, pages 114–123. ACM, 2011.
26. R. P. Grimaldi Chapter 3.2. Generating Functions, in K. H. Rosen, Handbook of discrete and combinatorial mathematics, CRC, 1999.
27. R. Rivest, A. Shamir, and D. Wagner. Time-lock puzzles and timed-release crypto. Technical report, Cambridge, MA, USA, 1996.
28. D. Stebila, L. Kuppusamy, J. Rangasamy, C. Boyd, and J. G. Nieto. Stronger difficulty notions for client puzzles and denial-of-service-resistant protocols. In *Proceedings of the 11th international conference on Topics in cryptology: CT-RSA 2011*, CT-RSA'11, pages 284–301. Springer-Verlag, 2011.
29. S. Suriadi, D. Stebila, A. Clark, and H. Liu. Defending web services against denial of service attacks using client puzzles. In *Web Services (ICWS), 2011 IEEE International Conference on*, pages 25–32. IEEE, 2011.
30. Q. Tang and A. Jeckmans. On non-parallelizable deterministic client puzzle scheme with batch verification modes. 2010.
31. S. Tritilanunt, C. Boyd, E. Foo, and J. M. G. Nieto. Toward non-parallelizable client puzzles. In *Proceedings of the 6th international conference on Cryptology and network security*, CANS'07, pages 247–264. Springer-Verlag, 2007.

A Puzzle properties

Some flavours of the notions defined by us appeared in the literature but they have never been formalized and previous work does not seem to make a clear distinction between them. For example, [2] introduces informally the notion of *computation guarantee* which requires that a malicious party cannot solve the puzzle significantly faster than honest clients. This is what we call optimality. Other papers [30] require that solving the puzzle be done via deterministic computation – this seems to be what we call an *fairness in solving.* For completeness we now enumerate various puzzle properties that can be found in the literature. Most of them are orthogonal, although we are not aware whether constructions for each combination of them exists or if they are useful in some protocols.

1. *Non-parallelizability.* Prohibits and adversary to use distributed computation in solving the puzzle. The first construction was provided by Rivest et al. in [27] in the context of time released crypto. Later non-parallelizable constructions were proposed by Tritilanunt et al. in [31] and constructions based on repeated squarings were studied by Jeckmans [18], Ghassan and Čapkun [21]. A construction based on computing modular square roots is presented in [19].
2. *Batching.* A popular technique in the case of the RSA cryptosystem, the ability to verify more puzzles in parallel can certainly save time on the side of the verifier. This property is discussed in [30].
3. *Granularity* as called by Tritilanunt et al. [31] or *adjustability of difficulty* by Abliz and Znati[2]. Refers to the pattern under which the difficulty level can be scaled: linearly, exponential, etc. One may want full control on how this difficulty level is adjusted, but some constructions by default allow only an exponential growth of difficulty (this can be easily fixed in most situations).
4. *Non-interactiveness.* Allows puzzles to be constructed in the absence of the verifier. This property was initially used by Back in [4] to combat spam, as one can not expect that the recipient of an e-mail will be present at the time when the e-mail is sent in order to produce a puzzle for the sender.
5. *Unforgeability.* Initially proposed by Chen et al. [9] this property prevents an adversary from forging puzzles. This property was questioned by Stebila et al. [28] in the context of non-interactive puzzles where the property does not always make sense (as the solver is the one that builds them). Still, this property is vital for practical scenarios when interaction between principals exists.
6. *State.* Some constructions require the server side to store information. Stateless puzzles may be desirable in order to avoid server depletion of memory resources (this is more prevalent in constrained environments).
7. *Freshness.* Replaying puzzles to clients can cause resource exhaustion on the client side, freshness prohibits this. This property is also called *tamper-resistance* by Abliz and Znati [2].
8. *Cost.* This can be refined along several lines, namely the cost on the server side (to generate) or the client side (to solve). Further it can be analyzed along with the unit of cost (CPU time, memory, bandwidth, etc.) and with the quantity required by one step (one hash function, one modular squaring, etc.). This is in close relation to the *difficulty* of the puzzle discussed bellow, but difficulty is usually defined formally as an upper bound on the adversary advantage in answering a puzzle, rather than the cost itself for solving the puzzle.
9. *Strong puzzle difficulty* was introduced by Stebila et al. in [28]. This requires that an adversary is unable to solve n puzzles easier than n times solving one puzzle. The same property is actually encountered by Abliz and Znati [2] as *correlation-free*, with the informal requirement that previous answers must not help the adversary to solve a new puzzle easier.
10. *Resilience to pre-computed attacks.* In some scenarios it is relevant if an adversary can perform off-line computations before obtaining the puzzle itself. Lack of resilience to pre-computed attacks may allow the adversary to mount a directed attack against a particular principal, despite the existence of a PoW protocol (performing off-line computations, before a connection is actually requested, is also immediately achievable in the case of non-interactive puzzles).

11. *Trapdoor.* Refers to whether there exists some private information that makes the puzzle easier to solve. This property along with constructions and practical settings is discussed by Gao in [14].
12. *Fairness* is defined in [2] as the property that a puzzle should take the same amount of time whatever the resource of the solver are. This is required in the context of DoS resistance in order to reduce the capability of a powerful attacker to a regular user. An example in this sense are memory bound functions from Abadi et al. [1] which rely on memory speed that in contrast to CPU speed is more uniform between devices.
13. *Minimum interference* is defined in [2] as a property that requires a puzzle not to interfere with the user's regular operations. If the puzzle takes too long then the user may avoid the use of the puzzle. Possibly, this is more likely related to good engineering of protocols, rather than an intrinsic property of a puzzle.
14. *Uniqueness.* Refers to whether a puzzle has or not a unique solution. This property is trivial, but it is not underlined explicitly in related work. In some contexts this property is not relevant, for example in most PoW protocols, while it is extremely relevant in others, for example in time-release crypto. In this later context it is essential that a puzzle has a unique solution since this solution has to be used as a key to decrypt some particular ciphertext.

B Proofs for the difficulty bounds (Theorems 1 and 2)

As a general procedure, in both proofs for the difficulty bounds we proceed in a similar way by first establishing the bound of the solving algorithm and then bound the adversary advantage which proves that the puzzles are optimal. As can be easily noted, the game played by the adversary is the concurrent solving game CS and subsequently, as also established by Proposition 1, the puzzles are difficulty preserving given that the solving algorithm works in a sequential manner (for completeness we prove that the solving time of n puzzles is n times the solving time for 1 puzzle given any fixed difficulty level d).

B.1 Proof of Theorem 1

Proof of the solving bound. Suppose that Find finishes at exactly the t-th query and let $t = t_1+t_2+...+t_n$ where t_i denotes the number of queries made to $\mathcal{H}$ to solve the i^{th} puzzle. The probability to solve the i^{th} puzzle at exactly the t_i query is obviously $(1-\frac{1}{2^d})^{t_i-1}\cdot\frac{1}{2^d}$. Since solving each puzzle is an independent event, the probability to solve the puzzles at exactly $t_1, t_2, ..., t_n$ steps for each puzzle is $\prod_{i=1,n}(1-\frac{1}{2^d})^{t_i-1}\cdot\frac{1}{2^d} = (1-\frac{1}{2^d})^{t-n}\cdot\frac{1}{2^{nd}}$. But there are exactly $\binom{t-1}{n-1}$ ways of writing t as a sum of exactly n integers from which the probability to solve the puzzle follows as: $\zeta^{HT}_{k,d,n}(t) = \sum_{i=n,t}\binom{i-1}{n-1}\cdot\frac{1}{2^{nd}}\cdot\left(1-\frac{1}{2^d}\right)^{i-n}$.

Proof of the adversary advantage. We prove the adversary advantage in the random oracle model. For this, challenger $\mathcal{C}$ simulates $\mathcal{H}$ by flipping coins and playing the following game $\mathbf{G}_0$ with adversary $\mathcal{A}$:

(1) The challenger $\mathcal{C}$ runs Setup on input 1^k then it will flip coins to answer to the adversary $\mathcal{A}$,

(2) The adversary $\mathcal{A}$ is allowed to ask $\mathsf{OGenSolve}$, OTest, $\mathsf{ComputeHash}$ which C answers as follows:

- on $\mathsf{OGenSolve}$, challenger $\mathcal{C}$ picks $r \in \{0,1\}^k$ checks if r is present on its tape and stores it if not then randomly chooses a solution sol and returns the pair $\{r, \mathsf{sol}\}$,
- on OTest, challenger $\mathcal{C}$ queries itself $\mathsf{OGenSolve}$ but marks its answers and solutions as $\{(r_1^\diamondsuit, \mathsf{sol}_1^\diamondsuit), (r_2^\diamondsuit, \mathsf{sol}_2^\diamondsuit), ..., (r_n^\diamondsuit, \mathsf{sol}_n^\diamondsuit)\}$ and returns just $\{r_1^\diamondsuit, r_2^\diamondsuit, ..., r_n^\diamondsuit\}$,
- on $\mathsf{ComputeHash}$, challenger $\mathcal{C}$ simulates $\mathcal{H}$ to the adversary A, that is, he receives (r, sol) from adversary $\mathcal{A}$, checks if (r, sol) was not already queried and if not he flips coins to get y and stores stores the triple (r, sol, y) on its tape then returns y to $\mathcal{A}$,

(3) At any point the adversary $\mathcal{A}$ can stop the game by sending $\mathcal{C}$ a set of pairs $\{(r_1^\diamondsuit, \mathsf{sol}_1^\diamondsuit), (r_2^\diamondsuit, \mathsf{sol}_2^\diamondsuit), ..., (r_n^\diamondsuit, \mathsf{sol}_n^\diamondsuit)\}$,

(4) When challenger $\mathcal{C}$ receives $\{(r_1^\diamondsuit, \mathsf{sol}_1^\diamondsuit), (r_2^\diamondsuit, \mathsf{sol}_2^\diamondsuit), ..., (r_n^\diamondsuit, \mathsf{sol}_n^\diamondsuit)\}$ he checks that each $\{r_1^\diamondsuit, r_2^\diamondsuit, ..., r_n^\diamondsuit\}$ are stored on its tape and for each solution it checks that the last d bits of y in $\{r, \mathsf{sol}, y\}$ are zero. If a triple $\{r, \mathsf{sol}, y\}$ such that the last d bits of y are zero is not present on the tape, then challenger $\mathcal{C}$ flips coins one more time to get a new y and accepts the solution if y ends with d zeros (note that these values are not stored on the tape). If all these hold then challenger $\mathcal{C}$ outputs 1, otherwise it outputs 0.

Remark 19. For correct simulation of $\mathsf{OGenSolve}$ the length l of the correct answer should be chosen according to the probability distribution of the lengths for a particular difficulty level, i.e., $\Pr[l] = (1-(1-2^{-d})^{2^l})(1-2^{-d})^{2^{l-1}}$.

Let $\mathbf{G}_1$ be the same as $\mathbf{G}_0$ with the following difference: on $\mathsf{OGenSolve}$, challenger $\mathcal{C}$ picks $r \in \{0,1\}^k$ checks if r is present on its tape and aborts if so, otherwise it continues as in $\mathbf{G}_0$ by storing the values then sending them to $\mathcal{A}$. We have: $\Big|\Pr\big[\mathcal{A} \text{ wins } \mathbf{G}_0\big] - \Pr\big[\mathcal{A} \text{ wins } \mathbf{G}_1\big]\Big| \leq \frac{q_{\mathsf{Gen}}^2}{2^{k+1}}$.

We now bound the adversary advantage in $\mathbf{G}_1$. At the end of the game, challenger $\mathcal{C}$ inspects his tape and sets t as the number of queries made to $\mathsf{ComputeHash}$ that have an $r_i^\diamondsuit, \forall i \in \{1,n\}$ as input. Let E_i denote the event that for i of the puzzles a pair $\{r^\diamondsuit, \mathsf{sol}^\diamondsuit, y\}$ where y ends with d zeros is not present on the tape. Obviously, there $n+1$ possible outcomes of $\mathbf{G}_1$: $E_0, E_1, ..., E_n$. In each E_i let $\Pr\big[\mathcal{A} \text{ wins } E_i\big]$ be the probability that the adversary has the correct answers for $n-i$ of the puzzles and he guessed the output of i of them which happens with probability 2^{-id} since the adversary never queried $\mathcal{H}$ to get a correct output. We have:

$$\begin{aligned}\Pr\big[\mathcal{A} \text{ wins } \mathbf{G}_1\big] &= \Pr\big[\mathcal{A} \text{ wins } E_0\big] + \frac{1}{2^d}\cdot\Pr\big[\mathcal{A} \text{ wins } E_1\big] + \frac{1}{2^{2d}}\cdot\Pr\big[\mathcal{A} \text{ wins } E_2\big] + \\ &\quad ... + \frac{1}{2^{nd}}\cdot\Pr\big[\mathcal{A} \text{ wins } E_n\big] = \zeta_{k,d,n}(t) + \frac{1}{2^d}\cdot\Pr\big[\mathcal{A} \text{ wins } E_1\big] + \\ &\quad + \frac{1}{2^{2d}}\cdot\Pr\big[\mathcal{A} \text{ wins } E_2\big] + ... + \frac{1}{2^{nd}}\cdot\Pr\big[\mathcal{A} \text{ wins } E_n\big] < \\ &\quad < \zeta_{k,d,n}^{HT}(t) + \frac{1}{2^d} + \frac{1}{2^{2d}} + ... + \frac{1}{2^{nd}} < \zeta_{k,d,n}^{HT}(t) + \frac{1}{2^d-1}\end{aligned}$$

By elementary calculations it follows that: $\mathsf{Win}_{\mathcal{A},k,d,n}^{\mathsf{HashTrail}}(q_{\mathsf{Gen}}, t) \leq \Big|\Pr\big[\mathcal{A} \text{ wins } \mathbf{G}_0\big] - \Pr\big[\mathcal{A} \text{ wins } \mathbf{G}_1\big]\Big| + \Pr\big[\mathcal{A} \text{ wins } \mathbf{G}_1\big] = \zeta_{k,d,n}^{HT}(t) + \frac{1}{2^d-1} + \frac{q_{\mathsf{Gen}}^2}{2^{k+1}}$. The puzzle follows as optimal since $\epsilon_{k,d,n}^{HT}(t) \leq \zeta_{k,d,n}^{HT}(t) + \frac{1}{2^d-1} + \frac{q_{\mathsf{Gen}}^2}{2^{k+1}}$ and $\frac{1}{2^d-1} + \frac{q_{\mathsf{Gen}}^2}{2^{k+1}}$ is negligible in d and k respectively. Now we prove that the puzzle is difficulty preserving which is trivial to do. For $n=1$ it is easy to prove that $\mathrm{t}_{\mathrm{avr}}(k,1,d) = 2^d$. This is straight forward since:

$$\begin{aligned}\mathrm{t}_{\mathrm{avr}}(k,1,d) &= \sum_{i=1,\infty} i\cdot\frac{1}{2^d}\cdot\left(1-\frac{1}{2^d}\right)^{i-1} = \frac{1}{2^d}\cdot\sum_{i=1,\infty} i\cdot\left(1-\frac{1}{2^d}\right)^{i-1} = \\ &= \frac{1}{2^d}\cdot\lim_{i\to\infty}\frac{i\cdot\left(1-\frac{1}{2^d}\right)^{i-1}\cdot\left(-\frac{1}{2^d}\right) - \left(1-\frac{1}{2^d}\right)^{i} + 1}{\frac{1}{2^{2d}}} = 2^d\end{aligned}$$

We now want to show that $n\cdot\mathrm{t}_{\mathrm{avr}}(k,1,d) = \mathrm{t}_{\mathrm{avr}}(k,n,d)$. By definition we have $\zeta_{k,d,n}^{HT}(t) = \sum_{i=n,t}\binom{i-1}{n-1}\cdot\frac{1}{2^{nd}}\cdot\left(1-\frac{1}{2^d}\right)^{i-n}$. Thus it follows:

$$\mathrm{t}_{\mathrm{avr}}(k,n,d) = \sum_{i=n,\infty} i \cdot (\zeta^{HT}_{k,d,n}(t) - \zeta^{HT}_{k,d,n}(t-1)) = \sum_{i=n,\infty} i \cdot \binom{i-1}{n-1} \cdot \frac{1}{2^{nd}} \cdot \left(1 - \frac{1}{2^d}\right)^{i-n}$$

Recall that $\binom{i}{j} = \binom{i-1}{j-1} + \binom{i-1}{j}$ and write

$$\mathrm{t}_{\mathrm{avr}}(k,n,d) = \sum_{i=n,\infty} i \cdot \left[\binom{i-2}{n-2} + \binom{i-2}{n-1}\right] \cdot \frac{1}{2^{nd}} \cdot \left(1 - \frac{1}{2^d}\right)^{i-n} =$$

$$= \frac{1}{2^d} \cdot \underbrace{\sum_{i=n,\infty} i \cdot \binom{i-2}{n-2} \cdot \frac{1}{2^{(n-1)d}} \cdot \left(1 - \frac{1}{2^d}\right)^{i-n}}_{\mathrm{t}_{\mathrm{avr}}(k,n-1,d) + \underbrace{\sum_{i=n,\infty} \binom{i-2}{n-2} \cdot \frac{1}{2^{(n-1)d}} \cdot \left(1 - \frac{1}{2^d}\right)^{i-n}}_{\epsilon_{k,d,n-1}(\infty)=1}}$$

$$+ \left(1 - \frac{1}{2^d}\right) \cdot \underbrace{\sum_{i=n,\infty} i \cdot \binom{i-2}{n-1} \cdot \frac{1}{2^{nd}} \cdot \left(1 - \frac{1}{2^d}\right)^{i-n-1}}_{\mathrm{t}_{\mathrm{avr}}(k,n,d) + \underbrace{\sum_{i=n,\infty} \binom{i-2}{n-2} \cdot \frac{1}{2^{(n-1)d}} \cdot \left(1 - \frac{1}{2^d}\right)^{i-n}}_{\epsilon_{k,d,n}(\infty)=1}}$$

Multiply with 2^d to get $\mathrm{t}_{\mathrm{avr}}(k,n,d) = \mathrm{t}_{\mathrm{avr}}(k,n-1,d) + 2^d$ from which by recurrence we have $\mathrm{t}_{\mathrm{avr}}(k,n,d) = n \cdot \mathrm{t}_{\mathrm{avr}}(k,1,d)$ which completes the proof.

B.2 Proof of Theorem 2

Proof of the solving bound. Algorithm Find solves n instances in exactly t steps for any set $\{t_1, t_2, ..., t_n\}$ such that $t = t_1 + t_2 + ... + t_n$ and $1 \leq t_i \leq 2^d$ where each t_i denotes the exact number of queries made to solve the i-th puzzle. The number of such sets is given by the restricted compositions of integer n which is $[z^i](z \cdot \frac{1-z^{2^d}}{1-z})^n$, i.e., the ways of writing i as sum of n terms each at most 2^d. Each such compositions has probability $\frac{1}{2^{nd}}$, thus the bound of Find follows:

$$\zeta^{HI}_{k,d,n}(t) = \sum_{i=n,t} [z^i] \left(z \cdot \frac{1 - z^{2^d}}{1-z}\right)^n \cdot \frac{1}{2^{nd}}$$

Proof of the adversary advantage. We prove the adversary advantage in the random oracle model. For this, challenger $\mathcal{C}$ simulates $\mathcal{H}$ by flipping coins and playing the following game $\mathbf{G}_0$ with adversary $\mathcal{A}$:

(1) The challenger $\mathcal{C}$ runs Setup on input 1^k then it will flip coins to answer to the adversary $\mathcal{A}$,

(2) The adversary $\mathcal{A}$ then starts to ask OGenSolve, OTest, ComputeHash which C answers as follows:

- on OGenSolve, challenger $\mathcal{C}$ picks $x \in \{0,1\}^k$ flips coins to get $y = \mathcal{H}(x)$, sets x' as the first d bits of x and x'' as the remaining bits, stores $\{\mathsf{puz} = (x'', y), \mathsf{sol} = x'\}$ on its tape and returns this value,

- on OTest, challenger $\mathcal{C}$ queries itself $\mathsf{OGenSolve}$ but marks its answers and solutions as $\{(\mathsf{puz}_1^{\diamondsuit}, \mathsf{sol}_1^{\diamondsuit}), (\mathsf{puz}_2^{\diamondsuit}, \mathsf{sol}_2^{\diamondsuit}), ..., (\mathsf{puz}_n^{\diamondsuit}, \mathsf{sol}_n^{\diamondsuit})\}$ and returns just $\{\mathsf{puz}_1^{\diamondsuit}, \mathsf{puz}_2^{\diamondsuit}, ..., \mathsf{puz}_n^{\diamondsuit}\}$,
- on $\mathsf{ComputeHash}$, challenger $\mathcal{C}$ simulates $\mathcal{H}$ to the adversary A, that is, he receives sol, x'' from adversary $\mathcal{A}$, it inspects its tape to see if a triple sol, x'', y is present on its tape and returns y if so, otherwise it flips coins to get an y and stores the triple $\{\mathsf{sol}, x'', y\}$ on its tape then returns y to $\mathcal{A}$.

(3) At any point the adversary $\mathcal{A}$ can stop the game by sending $\mathcal{C}$ a set of pairs $\{(\mathsf{puz}_1^{\diamondsuit}, \mathsf{sol}_1^{\diamondsuit}), (\mathsf{puz}_2^{\diamondsuit}, \mathsf{sol}_2^{\diamondsuit}), ..., (\mathsf{puz}_n^{\diamondsuit}, \mathsf{sol}_n^{\diamondsuit})\}$,

(4) When challenger $\mathcal{C}$ receives $\{(\mathsf{puz}_1^{\diamondsuit}, \mathsf{sol}_1^{\diamondsuit}), (\mathsf{puz}_2^{\diamondsuit}, \mathsf{sol}_2^{\diamondsuit}), ..., (\mathsf{puz}_n^{\diamondsuit}, \mathsf{sol}_n^{\diamondsuit})\}$ he checks that each $\{\mathsf{puz}_1^{\diamondsuit}, r_2^{\diamondsuit}, ..., \mathsf{puz}_n^{\diamondsuit}\}$ is on its tape and for each puzzle and solution it checks that a triple sol, x'', y is present on its tape. If these hold then challenger $\mathcal{C}$ outputs 1, otherwise it outputs 0.

Let $\mathbf{G}_1$ be the same as $\mathbf{G}_0$ with the following difference: on $\mathsf{OGenSolve}$, challenger $\mathcal{C}$ picks $x \in \{0,1\}^k$ checks if x' is already present on its tape and aborts if so, otherwise it continues as in $\mathbf{G}_0$ and stores it then sends it to $\mathcal{A}$. We have:

$$\Big|\Pr\big[\mathcal{A} \text{ wins } \mathbf{G}_0\big] - \Pr\big[\mathcal{A} \text{ wins } \mathbf{G}_1\big]\Big| \leq \frac{q_{\mathsf{Gen}}^2}{2^{k-d+1}}.$$

We now bound the adversary advantage in $\mathbf{G}_1$. At the end of $\mathbf{G}_1$ challenger $\mathcal{C}$ inspects his tape and sets t as the number of queries made to $\mathsf{ComputeHash}$ that have a target puzzle as input.

Let E_i denote the event that for i of the puzzles the solution is not present on the tape records from $\mathsf{ComputeHash}$. Obviously, there are $n+1$ possible outcomes of $\mathbf{G}_1$: $E_0, E_1, ..., E_n$. In each E_i it must be that the solution was guessed and passed the verification of the challenger. Note that in each E_i the challenger has performed exactly i more queries to $\mathcal{H}$ and the probability to get a correct solution in exactly $t+i$ queries is $[z^{t+1}](z \cdot \frac{1-z^{2^d}}{1-z})^n \cdot \frac{1}{2^{nd}}$.

We make the following relevant observation:

$$[z^i]\left(z \cdot \frac{1-z^{2^d}}{1-z}\right)^n \cdot \frac{1}{2^{nd}} \leq \frac{1}{2^d}, \forall i \in [n..n \cdot 2^d]$$

This is easy to prove. Note that for $n = 1$ it holds since each coefficient is 2^{-d}. Now proceed by induction. Assume this holds for $n-1$ and prove that it holds for n. We have:

$$[z^i]\left(z \cdot \frac{1-z^{2^d}}{1-z}\right)^n \cdot \frac{1}{2^{nd}} = [z^i]\left(\left(z \cdot \frac{1-z^{2^d}}{1-z}\right)^{n-1} \cdot \frac{1}{2^{(n-1)d}} \cdot \left(z + z^2 + ... + z^{2^d}\right) \cdot \frac{1}{2^d}\right)$$

Note that this last product has on the left side the coefficients for $n-1$ which are all smaller than 2^{-d} and due to the multiplication to the right hand side all these coefficients are added the divided again by 2^{-d}, which again gives coefficients at most 2^{-d}.

Thus we have:

$$\Pr\big[\mathcal{A} \text{ wins } \mathbf{G}_{\mathcal{R}}\big] \leq \zeta_{k,d,n}^{HI}(t) + [z^{t+1}]\left(z \cdot \frac{1-z^{2^d}}{1-z}\right)^n \cdot \frac{1}{2^{nd}} + [z^{t+2}]\left(z \cdot \frac{1-z^{2^d}}{1-z}\right)^n \cdot \frac{1}{2^{nd}} +$$
$$... + [z^{t+n}]\left(z \cdot \frac{1-z^{2^d}}{1-z}\right)^n \cdot \frac{1}{2^{nd}}$$
$$\leq \zeta_{k,d,n}^{HI}(t) + \frac{1}{2^d} + \frac{1}{2^d} + \frac{1}{2^d} + ... + \frac{1}{2^d} \leq \zeta_{k,d,n}^{HI}(t) + \frac{n}{2^d}$$

It follows that:

$$\begin{aligned}
\mathsf{Win}^{\mathsf{HashInv}}_{\mathcal{A},k,d,n}(q_{\mathsf{Gen}},t) &= \Pr\big[\mathcal{A} \text{ wins } \mathbf{G}_0\big] \\
&= \Pr\big[\mathcal{A} \text{ wins } \mathbf{G}_0\big] + \Pr\big[\mathcal{A} \text{ wins } \mathbf{G}_1\big] - \Pr\big[\mathcal{A} \text{ wins } \mathbf{G}_1\big] \\
&\le \Big|\Pr\big[\mathcal{A} \text{ wins } \mathbf{G}_0\big] - \Pr\big[\mathcal{A} \text{ wins } \mathbf{G}_1\big]\Big| + \Pr\big[\mathcal{A} \text{ wins } \mathbf{G}_1\big] \\
&= \zeta^{HI}_{k,d,n}(t) + \frac{n}{2^d} + \frac{q^2_{\mathsf{Gen}}}{2^{k-d+1}}.
\end{aligned}$$

Same as previously, the puzzle is optimal as $\epsilon^{HI}_{k,d,n}(t) \le \zeta^{HI}_{k,d,n}(t) + \frac{n}{2^d} + \frac{q^2_{\mathsf{Gen}}}{2^{k-d+1}}$ and $\frac{n}{2^d} + \frac{q^2_{\mathsf{Gen}}}{2^{k-d+1}}$ is negligible in d and k respectively.

To prove that the puzzle is difficulty preserving, for one instance of the puzzle the average solving time is $2^{d-1} + 1/2$. For n instances the average solving time is given by:

$$\mathrm{t}_{\mathrm{avr}}(k,n,d) = \sum_{t=n,n\cdot 2^d} t \cdot [\zeta^{HI}_{k,d,n}(t) - \zeta^{HI}_{k,d,n}(t-1)]$$

Take polynomial $P(z) = z^n \cdot (1 + z + z^2 + ... + z^{2^d-1})^n$, derive it and replace z with 1. Notice that this gives exactly $2^{nd} \cdot \mathrm{t}_{\mathrm{avr}}(k,n,d)$ and subsequently we have $\mathrm{t}_{\mathrm{avr}}(k,n,d) = n \cdot (2^{d-1} + 1/2)$.

B.3 Proofs on the approximations of the bounds

The first bound requires us to compute:

$$\sum_{i=n,t} \binom{i-1}{n-1} \cdot \frac{1}{2^{nd}} \cdot \left(1 - \frac{1}{2^d}\right)^{i-n} = \sum_{i=0,t-n} \binom{i+n-1}{n-1} \cdot \frac{1}{2^{nd}} \cdot \left(1 - \frac{1}{2^d}\right)^{i}$$

.

By using the binomial identity $\binom{n}{k} = \binom{n}{n-k}$ the term $\binom{i+n-1}{n-1}$ can be rewritten as $\binom{i+n-1}{i}$ and this is precisely the coefficient of x^i in the expansion of $(1 + x + x^2 + x^3 + ...)^n$ (see [26], page 208). While the sum $(1 + x^2 + x^3 + ...)^n$ ranges to infinity, the coefficient of the i-th term is obviously determined only by the first i terms and thus contained in $(1 + x^2 + x^3 + ... + x^i)^n$. By replacing x with $\left(1 - \frac{1}{2^d}\right)$ the i-th term is exactly the term we are looking for and this is obviously smaller than the sum of the terms, i.e.,

$$\begin{aligned}
\binom{i+n-1}{n-1} \cdot \left(1 - \frac{1}{2^d}\right)^i &< \left[1 + \left(1 - \frac{1}{2^d}\right)^1 + \left(1 - \frac{1}{2^d}\right)^2 + ... + \left(1 - \frac{1}{2^d}\right)^i\right]^n \\
&= \left[\frac{\left(1 - \frac{1}{2^d}\right)^{i+1} - 1}{-\frac{1}{2^d}}\right]^n = \left[2^d - \frac{(2^d-1)^{i+1}}{2^{di}}\right]^n
\end{aligned}$$

Now multiply the right part with 2^{-nd} to get:

$$\zeta^{HT}_{k,d,n}(t) = \sum_{i=n,t} \binom{i-1}{n-1} \cdot \frac{1}{2^{nd}} \cdot \left(1 - \frac{1}{2^d}\right)^{i-n} < \left[1 - \left(1 - \frac{1}{2^d}\right)^{i+1}\right]^n$$

To approximate the second difficulty bound we work with the coefficients of the polynomial $\left(z \cdot \frac{1-z^{2^d}}{1-z}\right)^n = z^n \left(z^0 + z^1 + ... + z^{2^d-1}\right)^n$. Indeed, these are the same coefficients as previously but we will not use their binomial expression. To compute the bound $\zeta^{HI}_{k,d,n}(t)$ we are simply interested in the sum of the coefficients from $\left(z^0 + z^1 + ... + z^{2^d-1}\right)^n$ up to the term having z^{t-n} (note that multiplication with z^n will shift all these coefficients to the right with n positions, so we sum up to $t-n$ rather than up to t). The sum of these coefficients is upper bounded by $(z^0 + z^1 + z^2 + ... + z^{t-n})^n$ if we set $z = 1$ (indeed, no term higher than z^{t-n} will contribute to this sum). But this sums all the $(t-n)n+1$ coefficients while we are interested only in the first $t-n$ of them. Since the first $t-n$ coefficients are the smallest (up to the term in the middle which is the larger due to the binomial expansion), we can safely divide the sum with n. Consequently, we can write:

$$\zeta^{HI}_{k,d,n}(t) = \sum_{i=n,t} [z^i] \left(z \cdot \frac{1-z^{2^d}}{1-z}\right)^n \cdot \frac{1}{2^{nd}} < \frac{(t-n+1)^n}{n2^{nd}} = \frac{1}{n}\left(\frac{t-n+1}{2^d}\right)^n$$

C Proofs for limitations of practical schemes (Theorem 3)

Basic scheme. Let $\mathcal{R}(\lambda)$ denote the revenue function for the adversary, that is, the number of connections earned by the adversary given that he made λ requests to the server. In case of PoW protocols, $\mathcal{R}(\lambda)$ is bounded by the amount of puzzles that the adversary correctly solved (not necessarily equal to λ). The maximum number of requests from the adversary is upper bounded by $\lambda_{\mathcal{A}}$ which is the maximum arrival rate of the adversary (limited by network parameters only), i.e., we have $\lambda \in [0, \lambda_{\mathcal{A}}]$. Obviously, a DoS takes place if $\mathcal{R}(\lambda) > \theta^{-1}_{service}$ since the server can handle at most $\theta^{-1}_{service}$ connections each second. It also holds that $\mathcal{R}(\lambda) \leq \lambda_{\mathcal{A}}$ since the adversary cannot get more connections than he requested for. Clearly, while $\lambda_{\mathcal{A}}$ is limited by network parameters only, $\mathcal{R}(\lambda)$ is also limited by the number of puzzles he was able to solve.

Now let $\mathcal{R}_{\max}$ denote the maximum number of connections that the adversary can get, given the limits of its computational resources. A misleading intuition is that the number of connections granted to the adversary is upper bounded by $\mathcal{R}_{\max} = \dfrac{\pi_{\mathcal{A}}}{d_{\mathsf{init}}}$ (this represents the maximum number of puzzles that he is able to solve at each instant of time). But by careful inspection of Definition 10 the difficulty bound includes the puzzle lifetime t_{puz} and the correct bound is $\mathcal{R}_{\max} = \dfrac{\pi_{\mathcal{A}} + t_{puz}\pi_{\mathcal{A}}}{d_{\mathsf{init}}}$ (since all puzzles computed during t_{puz} can be used as well to gain connections). But puzzle lifetime t_{puz} must be bigger than the time a client needs to solve the puzzle, i.e., $t_{puz} > d_{\mathsf{init}}\pi_{\mathcal{C}}^{-1}$, since otherwise clients are unable to solve the puzzles and cannot get connections anyway. Thus $\mathcal{R}_{\max} > \dfrac{\pi_{\mathcal{A}}}{d_{\mathsf{init}}} + \dfrac{\pi_{\mathcal{A}}}{\pi_{\mathcal{C}}}$. It follows that for any number of requests from the adversary (up to the maximum number of connections that he can get due to limitations on its computational resources) the revenue function is defined as $\mathcal{R}(\lambda) = \lambda$, if $\lambda \in \left[0, \dfrac{\pi_{\mathcal{A}}}{d_{\mathsf{init}}} + \dfrac{\pi_{\mathcal{A}}}{\pi_{\mathcal{C}}}\right]$. Which means that the number of connections (granted to the adversary) drops with the increase in the difficulty of the puzzle but it never drops below $\dfrac{\pi_{\mathcal{A}}}{\pi_{\mathcal{C}}}$ since: $\lim_{d_{\mathsf{init}} \to +\infty} \mathcal{R}(\lambda) = \dfrac{\pi_{\mathcal{A}}}{\pi_{\mathcal{C}}}$. Accordingly, the adversary can always get at least $\pi_{\mathcal{A}} \cdot \pi_{\mathcal{C}}^{-1}$ connections, regardless of the puzzle difficulty level, and the DoS condition is met when $\pi_{\mathcal{A}} \cdot \pi_{\mathcal{C}}^{-1} \geq \theta^{-1}_{service}$. Obviously $\pi_{\mathcal{A}} \cdot \pi_{\mathcal{C}}^{-1}$ is the minimum amount of connections gained on the side of the adversary and this is met as soon as $d_{\mathsf{init}} > \pi_{\mathcal{A}}$.

Filtering scheme. Having $\lambda \in [0, \lambda_{\mathcal{A}}]$ we build an adversary which gains connections faster than the inverse of the service time. By previous observation that the number of puzzles solved by the adversary includes the puzzles solved during the lifetime of the puzzles let us use the notation $\widetilde{\pi}_{\mathcal{A}} = \pi_{\mathcal{A}} + \pi_{\mathcal{A}}\frac{d_{\mathcal{C}}}{\pi_{\mathcal{C}}}$ to depict a more accurate bound on the computational resources of the adversary. If $\lambda(\beta d_{client} + (1-\beta) d_{\mathcal{A}}) < \widetilde{\pi}_{\mathcal{A}}$ the total number of PoWs received by the adversary does not exceed its computational resources, consequently he solves all PoWs that he receives. Further, if this is not the case then he solves all the $\beta\lambda$ easier PoWs (received due to the false negative rate of the filter) and uses its remaining resources to solve $(\widetilde{\pi}_{\mathcal{A}} - \beta\lambda d_{\mathcal{C}})/d_{\mathcal{A}}$ harder PoWs. All this as long as $\beta\lambda < \widetilde{\pi}_{\mathcal{A}}/d_{\mathcal{C}}$ a situation in which he solves only the easier PoWs. The revenue function of our adversary is the following:

$$\mathcal{R}(\lambda) = \begin{cases} \lambda, & \text{if } \lambda < \dfrac{\widetilde{\pi}_{\mathcal{A}}}{\beta d_{client} + (1-\beta) d_{\mathcal{A}}} \\ \dfrac{\widetilde{\pi}_{\mathcal{A}} - \beta\lambda d_{\mathcal{C}}}{d_{\mathcal{A}}} + \beta\lambda, & \text{if } \dfrac{\widetilde{\pi}_{\mathcal{A}}}{\beta d_{client} + (1-\beta) d_{\mathcal{A}}} \leq \lambda < \dfrac{\widetilde{\pi}_{\mathcal{A}}}{\beta d_{\mathcal{C}}} \\ \dfrac{\widetilde{\pi}_{\mathcal{A}}}{d_{\mathcal{C}}}, & \text{if } \dfrac{\widetilde{\pi}_{\mathcal{A}}}{\beta d_{\mathcal{C}}} \leq \lambda \end{cases}$$

But λ is limited by $\lambda_{\mathcal{A}}$. Since $\lambda_{\mathcal{A}} > \theta_{service}^{-1}$ (indeed, for a DoS attack to make sense, the adversary arrival rate must exceed the number of connections that can be handled by the server) and $\pi_{\mathcal{A}} \cdot \pi_{\mathcal{C}}^{-1} \geq \theta_{service}^{-1}$ clearly in the first and third case of $\mathcal{R}(\lambda)$ the filter cannot help. In the second case by increasing the difficulty $d_{\mathcal{A}}$ he solves at least $\beta\lambda$ PoWs and this does not help either if $\beta\lambda > \theta_{service}^{-1}$.

Filtering scheme with hidden difficulty puzzles. We use a similar adversary as previously, except that he cannot choose to solve only the easier PoW (of the client) since the difficulty is hidden. However, he can choose to invest only up to $d_{\mathcal{C}}$ in each puzzle and renounce to solve it if a solution is not found. This gives:

$$\mathcal{R}(\lambda) = \begin{cases} \lambda, & \text{if } \lambda < \dfrac{\widetilde{\pi}_{\mathcal{A}}}{\beta d_{client} + (1-\beta) d_{\mathcal{A}}} \\ \dfrac{\widetilde{\pi}_{\mathcal{A}} - \lambda d_{\mathcal{C}}}{d_{\mathcal{A}} - d_{\mathcal{C}}} + \beta\lambda, & \text{if } \dfrac{\widetilde{\pi}_{\mathcal{A}}}{\beta d_{client} + (1-\beta) d_{\mathcal{A}}} \leq \lambda < \dfrac{\widetilde{\pi}_{\mathcal{A}}}{d_{\mathcal{C}}} \\ \beta\dfrac{\widetilde{\pi}_{\mathcal{A}}}{d_{\mathcal{C}}}, & \text{if } \dfrac{\widetilde{\pi}_{\mathcal{A}}}{\beta d_{\mathcal{C}}} \leq \lambda \end{cases}$$

Same as previously, the first and second branches cannot help. In the third branch, the adversary again gets more connections than the inverse of the service time if $\beta\frac{\widetilde{\pi}_{\mathcal{A}}}{d_{\mathcal{C}}} > \theta_{service}^{-1}$.

Cascade scheme. In this case, the difficulty of each client PoW is added to the difficulty of the initial PoW. This also modifies $\widetilde{\pi}_{\mathcal{A}}$ to $\widetilde{\pi}_{\mathcal{A}} = \pi_{\mathcal{A}} + \pi_{\mathcal{A}}\frac{d_{\mathcal{C}} + d_{\mathsf{init}}}{\pi_{\mathcal{C}}}$. The revenue of the adversary follows as:

$$\mathcal{R}(\lambda) = \begin{cases} \lambda, & \text{if } \lambda < \dfrac{\widetilde{\pi}_{\mathcal{A}}}{\beta d_{client} + (1-\beta) d_{\mathcal{A}} + d_{\mathsf{init}}} \\ \dfrac{\widetilde{\pi}_{\mathcal{A}} - \lambda\beta(d_{\mathcal{C}} + d_{\mathsf{init}})}{d_{\mathcal{A}}} + \beta\lambda, & \text{if } \dfrac{\widetilde{\pi}_{\mathcal{A}}}{\beta d_{client} + (1-\beta) d_{\mathcal{A}} + d_{\mathsf{init}}} \leq \lambda < \dfrac{\widetilde{\pi}_{\mathcal{A}}}{\beta d_{\mathcal{C}} + d_{\mathsf{init}}} \\ \beta\dfrac{\widetilde{\pi}_{\mathcal{A}}}{\beta d_{\mathcal{C}} + d_{\mathsf{init}}}, & \text{if } \lambda > \dfrac{\widetilde{\pi}_{\mathcal{A}}}{\beta d_{\mathcal{C}} + d_{\mathsf{init}}} \end{cases}$$

On the first and second branches, the Cascade scheme fails to protect from the same reasons as previously. On the third branch we have:

$$\beta\frac{\widetilde{\pi}_{\mathcal{A}}}{\beta d_{\mathcal{C}} + d_{\mathsf{init}}} > \beta\frac{\widetilde{\pi}_{\mathcal{A}}}{d_{\mathcal{C}} + d_{\mathsf{init}}} > \beta\frac{\pi_{\mathcal{A}} + \pi_{\mathcal{A}}\dfrac{d_{\mathcal{C}} + d_{\mathsf{init}}}{\pi_{\mathcal{C}}}}{d_{\mathcal{C}} + d_{\mathsf{init}}} > \beta\frac{\pi_{\mathcal{A}}}{\pi_{\mathcal{C}}}$$

This again shows that if $\beta\frac{\pi_{\mathcal{A}}}{\pi_{\mathcal{C}}} > \theta_{service}^{-1}$ the scheme fails to protect.