Revisiting Difficulty Notions For Client Puzzles and DoS Resilience

Bogdan Groza[1] and Bogdan Warinschi[2]

[1] Faculty of Automatics and Computers, Politehnica University of Timisoara,
bogdan.groza@aut.upt.ro
[2] Computer Science Department, University of Bristol,
bogdan@cs.bris.ac.uk

Abstract. Cryptographic 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 towards 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 defying 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 optimal and ideal) 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. In addition, we briefly investigate existing definitions for the related notion of DoS security. We demonstrate that the only rigorous security notions proposed to date is not sufficiently demanding (as it allows to prove secure protocols that are clearly not DoS resilient) and suggest an alternative definition. Our results are not only of theoretical interest. We show that our better characterization of hardness for puzzles and DoS resilience allows establishing formal bounds on the effectiveness of client puzzles which confirm previous empirical observations.

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 examples, many papers addressed their use against resource depletion in SSL/TLS [7], TCP/IP [14], general authentication protocols [3,10], spam combat [9], [8], [11]. 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 [17]. 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 [17], the constructions proposed by Tritilanunt et al. in [21] and later by Jeckmans [12], Ghassan and Čapkun [15], Tang and Jeckmans [20], Jerschow and Mauve [13]. 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.

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. [6] 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. [18] 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. [18] propose a notion of puzzle difficulty that accounts for multiple puzzles being solved at once and prove that two existing constructions Hash-Inversion (initially used by Juels and Brainard [14]) 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 two 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*. We call a puzzle *ideal* if on the average the puzzle is as hard to solve as in the worst case. These notions have already appeared in the literature but 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 [20] require that solving the puzzle be done via deterministic computation – this seems

to be what we call an *ideal* puzzle. The formulations for both of these notions are in the multi-puzzle setting which, as correctly observed in [18], 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*. 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 [18] 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.

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 [5] and also [16] and [19]) and all seem to have inherited the weaknesses in the definition of [18].

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 [6]. Recently, Stebila et al. [18], motivated by the observation that the security notion of [6] 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.

Shortcomings of existing definitions. There are several weak points in the difficulty definition outlined above. Perhaps the most problematic one is that the property of a puzzle of being "strongly" difficult is in fact a property of the function ϵ that upperbounds 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 [6] 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.

To show that the tightness of the bound matters, take for example the case of the time-lock puzzles. 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.

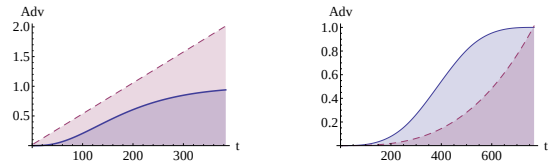
We can prove, and we specify this in a remark that follows, that if the bound is tight then the condition from [18] is sufficient to make a puzzle difficulty preserving. But, one may further ask if this condition is really necessary. The answer is negative. In fact, quite surprisingly, the HashTrail 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 [14] 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], [14].

Moreover, and this is another weakness for the definition of [18], 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.

Flaws in existing proofs. In light of the above comments, it is natural to ask how tight are the bounds obtained in [18]. By inspecting the security proofs it turns out that beside 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 [18] are wrong). Figure 1 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. Informally, the HashInversion puzzle requires that given $H(x'||x''), x''$ find random $x' \in_R [0..2^d)$. The difficulty bound claimed for this puzzle is $\epsilon_{d,k,n} = (\frac{q+n}{n \cdot 2^d})^n$ 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 [18] 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.



(i) (ii)

Fig. 1. 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)

3 Puzzle Difficulty

To formalize puzzle difficulty and related notions we proceed as follows. First we define the usual game of solving multiple puzzles and bound the adversary advantage by $\epsilon_{k,d,n}(t)$. Then, we 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, 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 ideal puzzles as puzzles that can not be solved faster than the average number of steps, this again up to some negligible factor.

3.1 Syntax for Cryptographic Puzzles

Our definition of a puzzle system follows in spirit the definition from [6], 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 algorithms,* $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Find}, \mathsf{Ver})$*, with the following descriptions:*

- $\mathsf{Setup}(1^k)$ *is the setup algorithm that takes as input a security parameter* 1^k *and outputs* dSpace*,* pSpace*,* sSpace *and* aSpace*,*
- $\mathsf{Gen}(d, \mathsf{atr})$ *is the puzzle generation algorithm, it takes as input the difficulty of the puzzle to be created* $d \in \mathsf{dSpace}$ *and a list of attributes* $\mathsf{atr} \in \mathsf{aSpace}$ *then outputs a puzzle instance* $\mathsf{puz} \in \mathsf{pSpace}$*,*
- $\mathsf{Find}(\mathsf{puz}, t)$ *is the solving algorithm that takes as input a puzzle* $\mathsf{puz} \in \mathsf{pSpace}$ *and the maximum number of steps* t *that is allowed to perform, then outputs a solution* $\mathsf{sol} \in \mathsf{sSpace} \cup \{\perp\}$ *(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* $\mathsf{sol}' \in \mathsf{sSpace}$ *and a puzzle* $\mathsf{puz} \in \mathsf{pSpace}$ *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 run the Find algorithm and that for any sol that is output of Find the verification holds. 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. In [18] Ver also takes the secret master key s and is also responsible for verifying if the solution is authentic, also embedding the functionality of VerAuth from [6]. Here we choose to keep the puzzle description close to that from [6], thus Ver is responsible just for verifying the correctness of the solution, and not the authenticity of the puzzle.
Remark 2. 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 [6], etc.
Remark 3. 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.

3.2 Optimal, ideal, and difficulty preserving puzzles

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.

PUZZLE SOLVING GAME. We define the puzzle game $\mathsf{Exec}^{\mathsf{CPuz},k}_{\mathcal{A},d,n}(t)$ 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 $\mathsf{dSpace}, \mathsf{pSpace}, \mathsf{sSpace}, \mathsf{aSpace}$ and sets $d \in \mathsf{dSpace}$ as the difficulty level of the game,
2. adversary $\mathcal{A}$ is allowed to make q_{Gen} queries to GenSolvePuz which returns each time a puzzle and its corresponding solution, i.e., $\{\mathsf{puz}, \mathsf{sol}\}$, and n queries to a Test oracle which on each invocation generates and returns a target puzzle $\mathsf{puz}^{\diamondsuit}$,
3. after t steps adversary $\mathcal{A}$ outputs the solutions $\{\mathsf{sol}^{\diamondsuit}_1, \mathsf{sol}^{\diamondsuit}_2, ..., \mathsf{sol}^{\diamondsuit}_n\}$ for puzzles $\{\mathsf{puz}^{\diamondsuit}_1, \mathsf{puz}^{\diamondsuit}_2, ..., \mathsf{puz}^{\diamondsuit}_n\}$ that were returned by Test,
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.

The *winning* event, denoted by $\mathsf{Win}^{\mathsf{CPuz},k}_{\mathcal{A},d,n}(t)$, is the event in which the adversary outputs a correct solution for the puzzles and the game returns 1, i.e.,

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

Remark 4. We did not stress whether the adversary $\mathcal{A}$ runs GenSolvePuz on its own or these are simulated by the challenger $\mathcal{C}$ as we did not make distinction 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.
Remark 5. In addition to previous hardness definitions we allow collision in the generation algorithm, that is, we did 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 previous section showed.
Remark 6. In $\mathsf{Exec}^{\mathsf{CPuz},k}_{\mathcal{A},d,n}(t)$ we assumed puzzles of the same difficulty level. But it is easy to extend this definition to puzzles of various difficulty levels as well. Such an extension to puzzles of multiple difficulty levels does not appear to be possible with the definition from related work [18] since multiple puzzle difficulty is linked inextricably to single puzzle difficulty, but for precisely the same difficulty parameter d.

Definition 2 (Difficulty bound). *For $\epsilon_{k,d} : \mathrm{N} \to [0,1]$ a family of functions indexed by parameters k, d and n, we say that $\epsilon_{k,d}(\cdot)$ is a difficulty bound for puzzle system* CPuz *if:* $\mathsf{Win}^{\mathsf{CPuz},k}_{\mathcal{A},d,n}(q_{\mathsf{Gen}}, t) \leq \epsilon_{k,d,n}(q_{\mathsf{Gen}}, t)$.

Before formally defining the different properties for puzzles, we need to introduce the average and the maximum solving time that one should expect from an honest client, that is a client who simply executes the Find algorithm that defines the puzzle. Below, we write $\mathsf{Exec}^{\mathsf{CPuz},k}_{\mathsf{Find},d,n}(t)$ 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. The following definitions captures the average probability of solving n puzzles of difficulty d in time t.

Definition 3 (Find bound). *For a given* CPuz *we denote by $\zeta_{k,d,n}(t)$ the probability that* Find *correctly finishes in at most t steps, i.e., $\zeta_{k,d,n}(t) = \Pr\left[\mathsf{Exec}^{\mathsf{CPuz},k}_{\mathsf{Find},d,n}(t) = 1\right]$.*

For a puzzle system, the next definition identifies the maximum number of steps needed by the Find algorithm to solve n puzzles with probability 1.

Definition 4 (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 $\zeta_{k,d,n}(t)$ is 1, i.e., $\zeta_{k,d,n}(\mathrm{t}_{\max}) = 1, \zeta_{k,d,n}(\mathrm{t}'_{\max}) < 1, \forall \mathrm{t}'_{\max} < \mathrm{t}_{\max}$.*

Definition 5 (Average solving time). *For a given* CPuz *we define the average solving time as the average number of steps required by* Find*, i.e., $\mathrm{t}_{\mathrm{avr}}(k,d,n) = \sum_{i=1,\mathrm{t}_{\max}} i \cdot [\zeta_{k,d,n}(i) - \zeta_{k,d,n}(i-1)]$.*

Remark 7. In Definition 1 we assumed that Find can solve at most one puzzle at a time, thus whenever Find is used to solve more than one puzzle we consider the usual way in which one repeatedly uses Find for each of the puzzles. In the case when Find behaves differently on more than one puzzle, one can extend the input and output of Find to a vector of puzzle instances and solutions.
Remark 8. There are puzzles for which $\mathrm{t}_{\max}$ is infinite while $\mathrm{t}_{\mathrm{avr}}$ is finite. Consider for example the trivial HashTrail puzzle, which we analyze in the next section, that consists in finding an input for a hash function such that the output ends with d consecutive zeros. Obviously, if one considers the hash function simulated by a random oracle, we have $\mathrm{t}_{\max} = \infty$ and $\mathrm{t}_{\mathrm{avr}} = 2^d$.

Definition 6 (Optimal puzzle). *We say that* CPuz *is optimal if at any number of steps and any number of puzzles the success probability of the 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., $\forall t, n, \epsilon_{k,d,n}(t) \leq \zeta_{k,d,n}(t) + \nu_n(k,d)$.*

Definition 7 (Difficulty preserving puzzle). *We say that an optimal* CPuz *is difficulty preserving if the average solving time for n puzzles of difficulty d equals n times the average solving time of a puzzle of difficulty 1 up to some negligible factor, i.e.,* $\forall n, d, |\mathrm{t}_{\mathrm{avr}}(k, d, n) - n \cdot \mathrm{t}_{\mathrm{avr}}(k, d, 1)| \leq \nu_n(k, d)$.

Remark 9. The optimality condition $\epsilon_{k,d,n}(t) \leq \zeta_{k,d,n}(t) + \nu_n(k, d)$ ensures that the bound from the puzzle solving game, i.e., $\epsilon_{k,d,n}(t)$, is $\nu_n(k, d)$ tight.
Remark 10. The condition $\epsilon_{k,d,n}(t) \leq \epsilon_{k,d,1}(t/n)$ is enough to assure that an optimal puzzle, i.e., a puzzle for which $\forall n, d, |\epsilon_{k,d,n}(t) - \zeta_{k,d,n}(t)| \leq \nu_n(k, d)$, 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).
Remark 11. To assure that a puzzle is difficulty preserving for puzzles of various difficulty levels, one must enforce that $\mathrm{t}_{\mathrm{avr}}(k, \overline{d}, n)$ is the sum of the difficulty levels, i.e., $\mathrm{t}_{\mathrm{avr}}(k, \overline{d}, n) = \mathrm{t}_{\mathrm{avr}}(k, d_1, 1) + \mathrm{t}_{\mathrm{avr}}(k, d_2, 1) + ... + \mathrm{t}_{\mathrm{avr}}(k, d_n, 1)$. Here $\overline{d}$ denotes an array of the difficulty levels.

Definition 8 (Ideal puzzle). *We say that an optimal puzzle* CPuz *is ideal if the average solving time equals the maximum solving time up to some negligible value in the the security parameter k and difficulty level d, i.e.,* $\forall n, d, |\mathrm{t}_{\mathrm{avr}}(k, d, n) - \mathrm{t}_{\max}(k, d, n)| \leq \nu_n(k, d)$. *Alternatively, having an optimal puzzle, i.e.,* $\epsilon_{k,d,n}(t) \leq \zeta_{k,d,n}(t) + \nu_n(k, d)$, $\forall n, d$, *the puzzle is ideal if* $\epsilon_{k,d,n}(t) = \nu_n(k, d), \forall n, d, \forall t < \mathrm{t}_{\max}$.

4 New difficulty bounds for HashTrail and HashInversion

We now examine the HashInversion and HashTrail puzzles and establish tight security bounds for each of them.

HASHTRAIL PUZZLE. Let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^k$ be a publicly known hash function. The HashTrail puzzle is a quadruple of algorithms:

- $\mathsf{Setup}(1^k)$ is the setup algorithm that on input 1^k outputs $\mathsf{dSpace} = [1, k]$, $\mathsf{pSpace} = \{0,1\}^* \times \{0,1\}^k$, $\mathsf{sSpace} = \{0,1\}^*$,
- $\mathsf{Gen}(d)$ 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 puz, sol as input and returns 1 if $\mathcal{H}(r||\mathsf{sol})_{1..d} = 0$ and 0 otherwise.

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

Remark 12. For HashTrail, in [18] 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 1 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.

HASHINVERSION PUZZLE. Let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^k$ be a publicly known hash function. The HashInv puzzle is the quadruple of algorithms:

- $\mathsf{Setup}(1^k)$ is the setup algorithm that on input 1^k outputs $\mathsf{dSpace} = [1,k]$, $\mathsf{pSpace} = \{0,1\}^* \times \{0,1\}^k$, $\mathsf{sSpace} = \{0,1\}^*$,
- $\mathsf{Gen}(d)$ 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 puz, sol as input and returns 1 if $\mathcal{H}(\mathsf{sol}||x'') = \mathcal{H}(x)$ and 0 otherwise.

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 $\mathrm{t}_{\mathrm{avr}}(k,d,1) = 2^{d-1}$, $\mathrm{t}_{\mathrm{max}}(k,d,1) = 2^d$ and solving and difficulty bounds: $\zeta^{HI}_{k,d,n}(t) = \sum_{i=n,t}[z^i]\left(z \cdot \frac{1-z^{2^d}}{1-z}\right)^n \cdot \frac{1}{2^{nd}}$, $\epsilon^{HI}_{k,d,n}(t) \leq \zeta^{HI}_{k,d,n}(t) + \frac{n}{2^d} + \frac{q^2_{\mathsf{Gen}}}{2^{k-d+1}}$.*

The proof of Theorem 1 can be found in Appendix A.1, due to space limitations we defer a proof of Theorem 2 for the extended version of this paper.

Remark 13. In [18] the advantage of HashInversion is upper bounded by $\left(\frac{q+n}{n \cdot 2^d}\right)^n$. As Figure 1 shows for a small numerical example, the advantage of the solving algorithm from the previous theorem is much bigger, thus the bound in [18] is wrong.

5 DoS Resilience

Defining resilience against resource exhaustion DoS is a non-trivial task that requires subtle analysis of the costs incurred by the 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.

AN ATTACK ON THE APPROACH OF STEBILA ET AL. The security definition for DoS resilience of Stebila et al. [18] builds directly on the difficulty of puzzle systems of [18] and, essentially, requires that an adversary cannot claim more resources than the number of puzzles he is able to solve in the running time of the adversary. The problem with

this definition 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 it uses these puzzles with solutions to claim the corresponding resources in a much shorter interval. The definition of [18] 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), but does not account for the possibility that the puzzles sent to claim resources could have been solved earlier.

OUR APPROACH. To prevent such attacks we take 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 $\Pi_{t_{puz}}(\mathsf{CPuz})$ 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} after the generation of the puzzle. We stress that we do not consider the detailed cost of running the server program, etc., and we consider as a premise that puzzles of difficulty d from CPuz are enough to protect the server.

PROTOCOL ATTACK GAME. We define the attack game $\mathsf{Exec}_{\mathbf{Adv}}^{\Pi_{t_{puz}}(\mathsf{CPuz}_{d,n}^k)}$ based on a two stage adversary. First adversary $\mathbf{Adv}_1$ 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 $\mathbf{Adv}_1$ outputs state information $\mathsf{state}_{\mathbf{Adv}_1}$ to $\mathbf{Adv}_2$ which is allowed to do the same actions subject only to one restriction: t_1 marks the time at which $\mathbf{Adv}_1$ has send its state information and at $t_2 + t_{puz}$ he 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.,

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

Definition 9 (DoS Resilience). *Let* $\mathsf{CPuz}_{d,n}^k$ *be an unforgeable, difficulty preserving puzzle. Protocol* $\Pi_{d,t_{puz}}(\mathsf{CPuz}_{d,n}^k)$ *is* $\epsilon_{d,n}^k$-DoS resilient *if for any* $t_1, t_2 \in [0, t_\Pi]$ *with* $t_1 < t_2$*, having an adversary* $\mathbf{Adv}$ *that can perform at most* $t_{\mathbf{Adv}} : t_2 - t_1 + t_{puz}$ *computations in time* $t_2 - t_1 + t_{puz}$ *it holds:*

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

PRACTICAL APPLICABILITY. We sketch the practical applicability of our security notions. The next theorem links the efficacy of a puzzle-based DoS defense system with the parameters of the underlying puzzle. Informally, the theorem states that a puzzle scheme can protect a protocol only when the ratio between the computational power of the adversary and that of the client does not exceed 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 on client puzzles). 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). While previous definition is of theoretical interest, it can be easily translated to practical systems where resource exhaustion occurs as soon as the requests of an adversary exceed the inverse of the service time. In the proof of the following theorem, 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 and maximum acceptable load of the server $\theta^{-1}_{service}$.

Theorem 3. *Consider protocol* $\Pi_{d,t_{puz}}(\mathsf{CPuz}^k_{d,n})$ *runs on a server side with service time* $\theta_{service}$ *for each connection and the computational resources of the adversary and clients are* $\pi_{\mathcal{A}}$ *and* $\pi_{\mathcal{C}}$ *respectively. Protocol* $\Pi_{d,t_{puz}}(\mathsf{CPuz}^k_{d,n})$ *can provide DoS protection only if* $\pi_{\mathcal{A}} < \pi_{\mathcal{C}} \cdot \theta^{-1}_{service}$ *and the maximum level of protection is reached at* $d = \pi_{\mathcal{A}}$.

This bound seems to justify existing empirical results. Dean and Stubblefield [7] provided the first positive results for protecting SSL/TLS by using client puzzles. In the performance related section, the authors of [7] 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 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.

6 Conclusion

We refined difficulty notions for puzzles, making a clear distinction between optimal, difficulty preserving, and ideal puzzles. Also we provided new difficulty bounds for two hash based puzzles. We showed that these bounds are tight enough to ensure optimality and that the puzzles are difficulty preserving. 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 including an explicit puzzle management mechanism in the puzzle protocol). 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, 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.

Acknowledgement. First author was partially supported by national research grant CNCSIS UEFISCDI, project number PNII IDEI 940/2008-2011 and by the strategic grant POSDRU/21/1.5/G/13798, inside POSDRU Romania 2007-2013, co-financed by the European Social Fund - Investing in People.

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. 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.
6. 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.
7. 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.
8. 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.
9. C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, pages 139–147, London, UK, 1993. Springer-Verlag.
10. Y. Gao, W. Susilo, Y. Mu, and J. Seberry. Efficient trapdoor-based client puzzle against DoS attacks. *Network Security*, pages 229–249, 2010.
11. A. Jeckmans. Computational puzzles for spam reduction in SIP. draft, July 2007.
12. A. Jeckmans. Practical client puzzle from repeated squaring. Technical report, August 2009.
13. 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.
14. 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.
15. 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.
16. 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.

17. R. Rivest, A. Shamir, and D. Wagner. Time-lock puzzles and timed-release crypto. Technical report, Cambridge, MA, USA, 1996.
18. 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.
19. 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.
20. Q. Tang and A. Jeckmans. On non-parallelizable deterministic client puzzle scheme with batch verification modes. 2010.
21. 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 Proofs

A.1 Proof of Theorem 1

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

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 GenSolvePuz, Test, ComputeHash which C answers as follows: (2.1) on GenSolvePuz, 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 $\{r, \mathsf{sol}\}$, (2.2) on Test, challenger $\mathcal{C}$ queries itself GenSolvePuz 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 $\{r_1^\diamondsuit, r_2^\diamondsuit, ..., r_n^\diamondsuit\}$, (2.3) on ComputeHash, challenger $\mathcal{C}$ simulates $\mathcal{H}$ to the adversary A, that is, he receives r, sol from adversary $\mathcal{A}$, check if r, sol was not already queried and if not he flips coins to get y and stores stores the triple $\{r, \mathsf{sol}, y\}$ 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 14. For correct simulation of GenSolvePuz 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 GenSolvePuz, 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 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:

$$\Pr\big[\mathcal{A} \text{ wins } \mathbf{G}_1\big] = \Pr\big[\mathcal{A} \text{ wins } E_0\big] + \frac{1}{2^d} \cdot \Pr\big[\mathcal{A} \text{ wins } E_1\big] + \frac{1}{2^{2d}} \cdot \Pr\big[\mathcal{A} \text{ wins } E_2\big] +$$

$$... + \frac{1}{2^{nd}} \cdot \Pr\big[\mathcal{A} \text{ wins } E_n\big] = \zeta_{k,d,n}(t) + \frac{1}{2^d} \cdot \Pr\big[\mathcal{A} \text{ wins } E_1\big] +$$

$$+ \frac{1}{2^{2d}} \cdot \Pr\big[\mathcal{A} \text{ wins } E_2\big] + ... + \frac{1}{2^{nd}} \cdot \Pr\big[\mathcal{A} \text{ wins } E_n\big] <$$

$$< \zeta_{k,d,n}^{HT}(t) + \frac{1}{2^d} + \frac{1}{2^{2d}} + ... + \frac{1}{2^{nd}} < \zeta_{k,d,n}^{HT}(t) + \frac{1}{2^d - 1}$$

By elementary calculations it follows that: $\mathsf{Win}_{\mathcal{A},d,n}^{\mathsf{HashTrail},k}(q_{\mathsf{Gen}}, t) \leq \Big|\Pr\big[\mathcal{A} \text{ wins } \mathbf{G}_0\big] - \Pr\big[\mathcal{A} \text{ wins } \mathbf{G}_1\big]\Big| + \Pr\big[\mathcal{A} \text{ wins } \mathbf{G}_1\big] = \zeta_{k,d,n}^{HT}(t) + \frac{1}{2^d-1} + \frac{q_{\mathsf{Gen}}^2}{2^{k+1}}$.

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 $d=1$ it is easy to prove that $\mathrm{t}_{\mathrm{avr}}(k,1,d) = 2^d$. This is straight forward since:

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

$$= \frac{1}{2^d} \cdot \lim_{i \to \infty} \frac{i \cdot \left(1 - \frac{1}{2^d}\right)^{i-1} \cdot \left(-\frac{1}{2^d}\right) - \left(1 - \frac{1}{2^d}\right)^i + 1}{\frac{1}{2^{2d}}} = 2^d$$

We now want to show that $n \cdot \mathrm{t}_{\mathrm{avr}}(k,d,1) = \mathrm{t}_{\mathrm{avr}}(k,d,n)$. 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:

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

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

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

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

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

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

A.2 Proof of Theorem 3

Let $\mathcal{R}$ denote the number of resources takeover by the adversary and λ the number of requests to the server. We have $\lambda \in [0, \lambda_{\mathcal{A}}]$ where $\lambda_{\mathcal{A}}$ is the maximum rate at which an adversary can request connections (limited by network parameters only). Obviously a DoS takes place if $\lambda > \theta^{-1}_{service}$ since the server can handle at most $\theta^{-1}_{service}$ connections each second. But by using client puzzle the number of requests is also bounded by the computational power of the adversary. A misleading bound on the adversary request rate is $\lambda_{\max} = \dfrac{\pi_{\mathcal{A}}}{d}$. By careful inspection of Definition 9 the difficulty bound includes the puzzle lifetime t_{puz} and the correct bound is $\lambda_{\max} = \dfrac{\pi_{\mathcal{A}} + t_{puz}\pi_{\mathcal{A}}}{d}$ (since all puzzle computed during t_{puz} can be used as well to gain resources). But puzzle lifetime t_{puz} must be bigger than the time a client needs to solve the puzzle, i.e., $t_{puz} > d\pi_{\mathcal{C}}^{-1}$, since otherwise clients are unable to solve the puzzles and cannot get resources anyway. Thus $\lambda_{\max} > \dfrac{\pi_{\mathcal{A}}}{d} + \dfrac{\pi_{\mathcal{A}}}{\pi_{\mathcal{C}}}$. It follows: $\mathcal{R}(\lambda) = \lambda$, if $\lambda \in \left[0, \dfrac{\pi_{\mathcal{A}}}{d} + \dfrac{\pi_{\mathcal{A}}}{\pi_{\mathcal{C}}}\right]$. Which means that the number of resources drops with the increase in the difficulty of the puzzle but it never drops below $\dfrac{\pi_{\mathcal{A}}}{\pi_{\mathcal{C}}}$ since: $\lim_{d \to +\infty} \mathcal{R}(\lambda) = \dfrac{\pi_{\mathcal{A}}}{\pi_{\mathcal{C}}}$. Accordingly, the adversary can always get at least $\pi_{\mathcal{A}} \cdot \pi_{\mathcal{C}}^{-1}$ resources, 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 resources gained on the side of the adversary and this met as soon as $d > \pi_{\mathcal{A}}$.