International Journal of Computer Mathematics
Vol. 88, No. 11, July 2011, 2246–2257

Taylor & Francis
Taylor & Francis Group

Towards a security model for computational puzzle schemes

Qiang Tang* and Arjan Jeckmans

DIES, Faculty of EEMCS, University of Twente, The Netherlands

(*Received 15 April 2010; revised version received 24 August 2010; accepted 25 November 2010*)

In the literature, computational puzzle schemes have been considered as a useful tool for a number of applications, such as constructing timed cryptography, fighting junk emails, and protecting critical infrastructure from denial-of-service attacks. However, there is a lack of a general security model for studying these schemes. In this paper, we propose such a security model and formally define two properties, namely the determinable difficulty property and the parallel computation resistance property. Furthermore, we prove that a variant of the RSW scheme, proposed by Rivest, Shamir, and Wagner, achieves both properties.

Keywords: computational puzzle; determinable difficulty; parallelization

2000 AMS Subject Classification: 68Q01

1. Introduction

A computational puzzle scheme [13,19,20,30] enables a prover to prove to a verifier that a certain amount of resources has been dedicated to solve a puzzle. To motivate our discussions, consider the following two toy examples.

(1) Let H be a one-way hash function. The verifier selects a random number x and set the puzzle to be $\mathsf{H}(x)$, while the prover searches for a number x', where $\mathsf{H}(x) = \mathsf{H}(x')$, as the puzzle solution. The verifier verifies a submitted puzzle solution x' through checking the equality $\mathsf{H}(x) = \mathsf{H}(x')$.
(2) Let H be a one-way hash function. The verifier selects d random numbers r_i $(1 \leq i \leq d)$ as the puzzle, while the prover computes $\mathsf{H}(r_i)(1 \leq i \leq d)$ as the puzzle solution. The verifier verifies a submitted puzzle solution $h_i(1 \leq i \leq d)$ through checking the equalities $\mathsf{H}(r_i) = h_i(1 \leq i \leq d)$.

In the first scheme, the verifier only needs to compute one hash in order to verify the solution, regardless of the amount of computation that the prover needs to perform. In contrast, in the second scheme, the verifier needs to perform the same amount of computation as that of the prover in

*Corresponding author. Email: q.tang@utwente.nl

ISSN 0020-7160 print/ISSN 1029-0265 online

DOI: 10.1080/00207160.2010.543951
http://www.informaworld.com

order to verify the solution. Intuitively, the first scheme seems to be the preferred one, but it still has some drawbacks.

- First, the amount of computation the prover needs to perform is probabilistic. Suppose x is chosen from the domain $[1, N]$. In the best case, the prover only needs to compute one hash, while in the worst case, it needs to compute N hashes until we find the right answer. As a consequence, the verifier never knows what is the exact amount of computation required to solve a puzzle, although the average is $(N/2)$ hashes.
- Second, if a prover has access to more than one computer, then it can speed up the puzzle solving process by having all of them work in parallel. For example, client A, which has access to x computers, could be $x - 1$ times faster in finding a solution than client B, which has access to one computer. In practice, it is very difficult for a verifier to determine the amount of computing resources a prover can access, especially in the presence of malicious provers which control a large number of Zombie computers. In some cases, such as timed cryptography [5,17,30], this will create an unfair situation for different provers. Arguably, in many other cases, such as fighting against denial-of-service (DoS) attacks [20], the parallelism property of a computational puzzle scheme may also be undesirable.

1.1 *Contribution*

Most of the existing computational puzzle schemes, as surveyed in Section 2, possess similar drawbacks. In fact, there is a lack of a general security model for formally studying relevant properties. In this paper, we propose a security model for computational puzzle schemes in general. In particular, in correspondence to the above drawbacks, we formally define two properties, namely *determinable difficulty* and *parallel computation resistance*.

- The determinable difficulty property implies that the verifier can precisely determine the required resource required from the prover in solving a puzzle.
- The parallel computation resistance property implies that the prover cannot accelerate the puzzle-solving process by letting more than one computer work in parallel.

These two properties describe the fundamental characteristics of computational puzzles. This is the first model of this kind, though there are security models for some specific categories of computational puzzle schemes [9]. In addition, we also provide discussions on some other properties such as computation disparity between the verifier and a prover, puzzle hardness granularity, and puzzle statefulness.

We prove that a variant of the RSW computational puzzle scheme, proposed by Rivest, Shamir, and Wagner, achieves the determinable difficulty and parallel computation resistance properties. This not only shows that the defined security properties are achievable but also solves the open issue about analysing the security of the well-known RSW scheme in a rigorously defined security model. To our knowledge, most existing computational puzzles do not achieve the parallel computation resistance property.

1.2 *Organization*

The rest of the paper is organized as follows. In Section 2, we present a brief literature review on computational puzzle schemes and present a formal definition. In Section 3, we propose a general security model and define the properties. In Section 4, we present a variant of the RSW computational puzzle scheme and prove its security. In Section 5, we conclude the paper.

2. Preliminary of computational puzzles

In this section, we briefly review the literature of computational puzzle schemes, and then present a general and formal definition of such schemes to facilitate our following discussions.

2.1 *Literature review*

Merkle [27] was the first to introduce the notion of *puzzle* which led to the invention of public key cryptography. In this context, the puzzle is required to be unsolvable by any polynomial-time entity. Dwork and Naor [13] proposed the concept of *pricing function* to combat junk emails. Rivest *et al.* [30] proposed the concept of *timed-lock puzzle*, which serves as a tool to realize the concept of *timed-release crypto*. Juels and Brainard [20] proposed the concept *client puzzle* and suggested to use it to prevent DoS attacks. Regardless of the different notations, *pricing function*, *timed-lock puzzle*, *proof of work*, and *client puzzle* share the same characteristic: they can be regarded as another type of *puzzle* (different from that of Merkle [27]) which is moderately hard in the sense that a polynomial-time entity can successfully find a solution by spending a certain amount of resources. Without loss of generality, we use the term *computational puzzle* to refer to the second type of puzzle.

Roughly speaking, there are two types of computational puzzle schemes. One type is CPU-bound, where the computation is measured by the amount of CPU cycles needed to solve a puzzle. Some examples are those in [4,9,13,19,20,30,32,36], which form the majority of the existing computational puzzle schemes. Abadi *et al.* [1] first noticed the fact that CPU power varies a lot for different computers (such as PC, PDA, and Workstation), and introduced memory-bound computational puzzle schemes, where the computation is measured by the amount of memory look-ups needed to solve a puzzle. The schemes in [11,12] fall into this category. Regardless of the nature of different forms of computations involved in solving puzzles, the essence is the same, namely the prover needs to spend a certain amount of resources (either CPU cycles or memory look-ups) in finding a solution.

Computational puzzle schemes have been explored to realize a number of secure functionalities, such as timed encryption [30], timed bit commitment [5], timed release of digital signatures [17], uncheatable benchmarks [8], trustworthy website usage metering [15], and generating delays in lottery applications [18]. The rationale is that the process of spending resources in solving a puzzle automatically results in a time delay, which enables the verifier to control when the prover is able to access the functionalities. In other words, the functionalities are masked by puzzles, which need to be solved first before the prover can access the functionalities. Besides realizing various security functionalities, researchers have also applied computational puzzle schemes to mitigate a wide range of DoS attacks, such as fighting junk emails [12,13], protecting authentication protocols [2,23,29], protecting IP networks [10,14,16,20,33–35], protecting wireless networks [24,26], and preventing Sybil attacks [6]. With a computational puzzle scheme implemented, the server (playing the role of verifier) can mitigate an attack by asking every client (playing the role of prover) to solve a puzzle before allocating any required resource. The rationale is that the number of 'valid' requests from a malicious client will drop to some extent, because the client has only a limited resource to find puzzle solutions. In the literature, there has been some debate on whether computational puzzle schemes are really helpful to detect DoS attacks. Based on the collected results from ISPs in UK, Laurie and Clayton [21] claimed that computational puzzle schemes are hardly effective in combating junk emails in practice, while Liu and Jean Camp [25] argued that computational puzzle schemes could be helpful if such schemes are used in combination with reputation systems.

Chen *et al.* [9] proposed a security model for computational puzzle schemes tailored for defeating DoS attacks. In particular, their definition has focused on the unforgeability and difficulty

of puzzles. However, their definition of puzzle difficulty implies neither the determinable difficulty property nor the parallel computation resistance property.

2.2 *Definition of computational puzzle schemes*

In the design and analysis of computational puzzle schemes, we typically do not directly talk about how many CPU cycles or memory lookups are required for puzzle generation, puzzle solving, and puzzle solution verification. Instead, we often use the number of some generic operations (such as a hash or a multiplication in some group) as a puzzle complexity metric. For the purpose of notation, we assume the generic operation to be denoted as Func.

A computational puzzle scheme consists of four (probabilistic) polynomial-time algorithms (Setup, PuzzleGen, PuzzleSol, PuzzleVer).

- Setup(ℓ, D): Run by the verifier, this algorithm takes a security parameter ℓ and an upper bound D on the puzzle hardness (see the definition below) as input, and outputs the public system parameter *params* and a private key *mk*. The public system parameter *params* should also include a specification about the metric function Func. This system parameter is implicitly part of the input to other algorithms, and we omit it in the descriptions for simplicity reasons.
- PuzzleGen(mk, d, *req*): Run by the verifier, this algorithm takes the private key mk, a hardness parameter d, and some additional information req as input, and outputs a puzzle puz and some relevant information *info*. The hardness parameter d is an integer which indicates the total number of Func operations required. The verifier sends puz to the prover, and keeps *info* for verifying the solution.
- PuzzleSol(*puz*): Run by a prover, this algorithm takes a puzzle puz as input and outputs a puzzle solution sol.
- PuzzleVer(mk, *info*, *sol*): Run by the verifier, this algorithm takes the private key *mk*, the puzzle information *info*, and the solution *sol* as input, and outputs 1 if *sol* is correct or 0 otherwise.

It is worth noting that a prover may not follow the PuzzleSol algorithm to find a solution, and in fact it can use any means to find the solution. But the properties *determinable difficulty* and *parallel computation resistance*, which will be defined in the next subsection, imply that a prover need to perform d operations of Func in order to find a solution whether or not it follows the PuzzleSol algorithm. In addition, in some schemes, a long-term private key mk may be unnecessary or the additional information *req* may not be required as part of the input. In some application scenarios, such as timed cryptography [5,17,30], a puzzle verification algorithm PuzzleVer is not explicitly required. Nonetheless, the above definition is meant to be general enough to cover the most existing computational puzzle schemes.

3. Property formulations for computational puzzles

Similar to other types of cryptographic schemes, soundness is a basic requirement for a computational puzzle scheme. In our case, soundness is defined to be that, given a puzzle with the hardness parameter d, the prover can find a correct solution by performing a total number of d operations of Func.

In the following, we formalize the properties *determinable difficulty* and *parallel computation resistance*. It is worth noting that, as in the case of most cryptographic formulations, we only consider a polynomial time adversary in our formulation and define that a property is achieved if

an adversary has only a negligible advantage in the attack. The notion of negligibility is defined as follows.

DEFINITION 1 *The function $P(k) : \mathbb{Z} \to \mathbb{R}$ is said to be negligible with respect to k if, for every polynomial $f(k)$, there exists an integer N_f such that $P(k) < (1/f(k))$ for all $k \geq N_f$.*

3.1 *The determinable difficulty property*

Informally, the determinable difficulty property implies that the verifier can precisely determine the required resource from the prover in solving a puzzle. Basically, this has two implications.

- One is that puzzles, generated by the verifier, are independent from each other in the sense that solving one puzzle does not help solve another puzzle. In other words, any puzzle generated by the verifier will require the prover to perform d Func operations.
- The other is that puzzles, generated by the verifier, are unpredictable in the sense that any puzzle generated by the verifier looks fresh so that the prover is unable to pre-compute the solution.

An attack succeeds if, after receiving a puzzle of hardness d, the adversary finds a solution by performing less than d Func operations. As a standard practice, attacks against the determinable difficulty property is simulated through the following three-phrase game between an adversary and a challenger, as depicted in Figure 1. In the attack game, the adversary plays the role of a malicious prover and the challenger plays the role of the verifier.

In practice, the only information to a prover is the puzzles generated by the verifier and the public system parameter. It is straightforward to check that the adversary has all the same privileges as a malicious prover in practice. Formally, the determinable difficulty property is defined as follows.

DEFINITION 2 *Suppose that the adversary has performed $d^\dagger$ Func operations in the Response phase of the attack game (defined in Figure 1). A computational puzzle scheme achieves the determinable difficulty property if the probability $d^\dagger < d^*$ is negligible with respect to the security parameter ℓ. The probability is computed based on the (random) coin tosses of the challenger and the adversary.*

1. Setup phase: The challenger takes the security parameter and an upper bound D on the puzzle hardness as input, and runs the Setup algorithm to generate the master key mk and the public system parameter $params$.

2. Challenge phase: The adversary issues queries to the PuzzleGen oracle with its chosen d and req, and obtains the reply $puz = \text{PuzzleGen}(mk, d, req)$ from the challenger. At some point, the adversary sends d^* and req^* to the challenger, which sends $puz^* = \text{PuzzleGen}(mk, d^*, req^*)$ back as a challenge.

3. Response phase: The adversary can issue queries to the PuzzleGen oracle as in the challenge phase. At some point, the adversary terminates by outputting a correct solution sol^* to the puzzle puz^*.

Figure 1. The attack game.

3.2 *The parallel computation resistance property*

As illustrated by the example schemes in Section 1, if a prover has access to a number of computers, it is able to solve a puzzle much faster than others by dividing the workload and letting the computers work in parallel. Informally, the parallel computation resistance property implies that a prover cannot accelerate the puzzle-solving process by exploiting the parallelism. More concretely, the parallel computation resistance property requires that, after receiving a puzzle of hardness d from the verifier, a prover needs to *sequentially* perform d Func operations to find a solution. It means that the prover is unable to speed up the process by letting more than one computer work in parallel. This also implies that the best strategy for the prover is to use its fastest computer to solve the puzzle. We first introduce the notion of *sequential computing time* as follows.

DEFINITION 3 *Suppose that a prover has access to a number of computers which can work in parallel, and the* Func *function is implemented as a program in every computer. Let the each evaluation of the* Func *function be denoted as an event, marked by the starting time and ending time of the evaluation, and the set of events be denoted by* g_t $(1 \leq t \leq N)$ *in a computation task. Consider all sequences of the events of the following form:*

$$\text{Seq}_x = \{q_1, q_2, \ldots, q_x\},$$

where, $1 \leq x \leq N$ *and for all* $1 \leq i \leq x - 1$, *the* q_{i+1}*th* Func *evaluation is started by a computer after the* q_i*th* Func *evaluation has ended (by any computer). The sequential computing time of the computation task is the maximum value of the sequence indexes, namely* $\max_x$.

Formally, the parallel computation resistance property is defined as follows.

DEFINITION 4 *Suppose that the sequential computing time of the computation task of finding a correct solution* sol^* *is* $d^\dagger$ *in the Response phase of the attack game* (*defined in Figure* 1). *A computational puzzle scheme achieves the parallel computation resistance property if the probability* $d^\dagger < d^*$ *is negligible with respect to the security parameter* ℓ. *The probability is computed based on the* (*random*) *coin tosses of the challenger and the adversary.*

According to our definition, the parallel computation resistance property implies the determinable difficulty property. However, the reverse is clearly not true.

Parallel computation resistance is a very desirable property when we want to limit the disparity of computing powers of potential provers of a computational puzzle scheme. Moreover, in some cases, this property is crucial to mitigate attacks from malicious provers which control a cluster of Zombie computers. However, the determinable difficulty property is of independent interest in evaluating the security of computational puzzle schemes when the property parallel computation resistance is unnecessary. There are straightforward ways to convert a scheme, which achieves the parallel computation resistance property, into a scheme which achieves determinable difficulty and yet supports parallelism. For example, to generate a puzzle of hardness d, the verifier generates x independent sub-puzzles $\{puz_1, \ldots, puz_x\}$ such that each of the sub-puzzle has the hardness (d/x). By doing so, the total hardness is still $d = (d/x) \cdot x$, but a prover can solve the sub-puzzles in parallel.

3.3 *Further remarks*

When designing computational puzzle schemes, a naturally desirable property is the *computation disparity* between the verifier and a prover: it should take very little resource for the verifier

to generate a puzzle and verify a solution, while it should take the prover a certain amount of resources (determined by the verifier) to find a solution. Formally, this property can be evaluated by a ratio parameter (v/d), where v is the total number of Func operations required in generating a puzzle and verifying a solution, and d is the puzzle hardness.

With respect to puzzle hardness d, it is desirable that a computational puzzle scheme provides finer granularity support. Take the hash-based client puzzle scheme as an example[1]: a puzzle is in the form of (h, x_2), where $h = \mathsf{H}(x_1||x_2)$, H is a collision-resistant hash function, and x_1 has bit-length k; a solution is a k bit x_1' such that $h = \mathsf{H}(x_1'||x_2)$. Clearly, the puzzle hardness can only be set exponentially with respect to k in this case. However, in the RSW computational puzzle scheme [30] and the variant in Section 4, the puzzle hardness can be set linearly in the number of multiplications. Clearly, with the latter, the verifier can more flexibly set d according to the requirements of the underlying applications.

According to the definition of a computational puzzle scheme given in Section 2, the verifier may store some state information *info* for each puzzle. In this case, the scheme is said to be stateful. In practice, a stateful computational puzzle scheme may be considered to be inefficient because the verifier needs to store a piece of information for every unverified puzzle. Nevertheless, it is straightforward to turn a stateful scheme into a stateless one by sending both *puz* and *info* to the client, which should send *sol* and *info* back to the server for verification. In the case that *info* compromises the determinable difficulty property or the parallel computation resistance property, the verifier can protect *info* with a symmetric key encryption algorithm and a message authentication code algorithm. It is worth noting that a completely stateless computational puzzle scheme may be prone to replay attacks, in which an adversary replays a puzzle and its solution to the verifier. A more detailed discussion of this issue is beyond the scope of this paper, and should be addressed in the deployment of a computational puzzle scheme.

4. Analysis of a variant of the RSW scheme

In this section, we analyse a variant of the RSW computational puzzle scheme and show that it achieves the parallel computation resistance property. In the literature, Tritilanunt *et al.* [32] proposed a scheme towards achieving the parallel computation resistance property, but they did not give a rigorous security proof. It is unclear whether their scheme can be proven in our security model.

4.1 *Description of the scheme*

The algorithms of the new scheme are defined as follows.

- Setup(ℓ, D): This algorithm selects two random large primes p, q and a cryptographic hash function $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_{pq}^*$. The public parameter is (ℓ, D, pq), and the master key is $mk = (p, q)$. In addition, the metric function Func is multiplication in $\mathbb{Z}_{pq}^*$. Given ℓ, the prime numbers p, q should be chosen in such a way that the required assumptions hold, as stated in the next subsection.
- PuzzleGen(mk, d, req): This algorithm chooses $r \in_R \mathbb{Z}_{pq}$ and computes $g = \mathsf{H}(r||req)$, and outputs the puzzle $puz = (g, d)$. The related puzzle information is $info = (r, d, req)$.
- PuzzleSol(puz): This algorithm outputs $sol = g^{2^d} \mod pq$.
- PuzzleVer$(mk, info, sol)$: This algorithm returns 1 if $sol \equiv (\mathsf{H}(r||req))^{2^d \bmod \phi(pq)} \pmod{pq}$, and returns 0 otherwise.

In this scheme, g is computed as $g = \mathsf{H}(r||req)$, which can be regarded as a randomly chosen element of $\mathbb{Z}_{pq}^*$ if H is modelled as a random oracle (as shown in our security proof). By doing so,

if needed, this element can be bound to situational information (such as the identity information of the prover) contained in *req*. Furthermore, the modification also facilitates the proof in our security model. Note that, in [30], g is directly assumed to be a randomly chosen element of $\mathbb{Z}^*_{pq}$. We further note that the hah function H can be replaced with any function that maps $r||req$ to a random element in $\mathbb{Z}^*_{pq}$.

The puzzle hardness d can be set linearly in the number of multiplications in the group $\mathbb{Z}^*_{pq}$. It is clear that the workload of the verifier is constant regardless of the puzzle hardness. With respect to the verification complexity of the verifier, we omit the computation $2^d \mod \phi(pq)$ for two reasons. One is that it could be pre-computed and stored by the verifier. The other is that, in many cases, multiple puzzles might share the same hardness so that the computation only needs to be done once. As a consequence, it is straightforward to calculate that the average verification complexity for the verifier is approximately $(3L/2)$ multiplications in $\mathbb{Z}^*_{pq}$, where L is the bit-length of $\phi(pq)$. The computation disparity ratio parameter is $(3L/2d)$.

4.2 *Security analysis*

Leander and Rupp [22] model $\mathbb{Z}_{pq}$ as a generic ring structure in order to prove the equivalence of RSA and factoring. In this paper, we assume the same structure. For our purpose, we only describe a generic algorithm according to the multiplication operation in $\mathbb{Z}^*_{pq}$ in the following. It is worth noting that there are different ways of giving the definition [28,31], and we follow that of Shoup [31].

Let $g \in_R \mathbb{Z}^*_{pq}$ and σ be an encoding function of $\mathbb{G} = \{g^i | i \in \mathbb{N}\}$ on $\{0,1\}^{|pq|}$, where $\mathbb{N}$ is the set of integers and $|pq|$ means the bit-length of pq. Suppose $\mathcal{O}$ is a multiplication oracle, which, for any $r \in \mathbb{G}$, computes $\sigma(r)$ as follows: if r has been calculated before, then the same value of $\sigma(r)$ is returned; otherwise, it sets $\sigma(r)$ to be a random value from $\{0,1\}^{|pq|} \setminus \mathcal{S}$, where $\mathcal{S}$ is a set initialized to be $\{\sigma(g)\}$. A generic algorithm $\mathcal{A}$ is a probabilistic algorithm, which takes $\mathcal{S}$ and pq as input, and behaves as follows. At any time, $\mathcal{A}$ can send a query with the input (x, y, b), where $x, y \in \mathcal{S}$ and $b \in \{1, -1\}$, to $\mathcal{O}$, and will receive $\sigma(x \cdot y^b)$ as the reply. After every query, the result is added to the set $\mathcal{S}$.

Remark 1 The concept of generic algorithms, which traces back to Babai and Szemeredi [3], has been extensively used to model algebraic objects, such as groups and rings. It has made it possible to prove the security of many cryptographic protocols. Note that this methodology does possess an inherent limitation, namely a lot of attacks against cryptographic protocols have actually been subject to the successful exploit of the structures of the underlying algebraic objects. Nevertheless, a security proof can still provide a certain level of confidence in the security of the studied protocols.

To complete our proof, we further need the following computational assumption associated with the multiplication group $\mathbb{Z}^*_{pq}$. This assumption is similar to the representation assumption in groups of prime order made by Brands [7].

DEFINITION 5 *Let p, q be two prime numbers. The extended discrete logarithm assumption holds for $\mathbb{Z}^*_{pq}$ if the following event only occurs with a negligible probability with respect to a security parameter ℓ: Given $(pq, g, g_i\ (1 \le i \le V))$, where $g, g_i\ (1 \le i \le V)$ are chosen uniformly at random from $\mathbb{Z}^*_{pq}$ and V is any polynomial in the security parameter, a polynomial-time adversary finds $x \ne 0, x_i\ (1 \le i \le V) \in \mathbb{N}$ such that*

$$g^x \equiv \prod_{x_i \in \mathbb{N}}^{1 \le i \le V} g_i^{x_i} \pmod{pq}.$$

In the following theorem, we prove that the variant scheme achieves the parallel computation resistance property (and certainly also the determinable difficulty property).

THEOREM 1 *If the adversary is modelled as a generic algorithm as defined above, then the variant scheme achieves the parallel computation resistance property based on the extended discrete logarithm assumption (given in Definition 5) in the random oracle model.*

Proof sketch In order to prove the theorem, we need to show that the adversary's advantage in the attack game (defined in Figure 1) is negligible according to Definition 4. Since the adversary is modelled as a generic algorithm, there are two types of oracle queries it may issue. One is the PuzzleGen query which can be issued to the challenger, and the other is multiplication oracle query which can be issued to the generic group oracle $\mathcal{O}$.

We first consider a simple situation where the adversary does not issue any PuzzleGen query in the game. In this situation, the best strategy for the adversary to issue oracle queries to $\mathcal{O}$ in the Response phase is the following.

(1) At the beginning, the adversary issues two queries with the inputs $(\sigma(g^*), \sigma(g^*), 1)$ and $(\sigma(g^*), \sigma(g^*), -1)$. Clearly, until it receives the replies, namely $\sigma((g^*)^2)$ and $\sigma(1)$, from the oracle $\mathcal{O}$, it does not make sense for the adversary to send any other query.
(2) After obtaining the replies, the set $\mathcal{S}$ becomes $\{\sigma(1), \sigma(g^*), \sigma((g^*)^2)\}$. Then the adversary issues queries with the following inputs:

$$(\sigma(g^*), \sigma((g^*)^2), 1), (\sigma((g^*)^2), \sigma((g^*)^2), 1),$$
$$(\sigma(1), \sigma(g^*), -1), (\sigma(1), \sigma((g^*)^2), -1).$$

Clearly, until it receives the replies, namely $\sigma((g^*)^3)$, $\sigma((g^*)^4)$, $\sigma((g^*)^{-1})$, and $\sigma((g^*)^{-2})$ from the oracle $\mathcal{O}$, it does not make sense for the adversary to send any other query.
(3) After obtaining the replies, the set $\mathcal{S}$ becomes

$$\{\sigma(1), \sigma(g^*), \sigma((g^*)^2), \sigma((g^*)^3), \sigma((g^*)^4), \sigma((g^*)^{-1}), \sigma((g^*)^{-2})\}.$$

Then the adversary issues queries with (x, y, b), where $x, y \in \mathcal{S}, b \in \{1, -1\}$, and $\sigma(x \cdot y^b) \notin \mathcal{S}$. Clearly, until it receives the replies, it does not make sense for the adversary to send any other query.
(4) The adversary continues the above process until it sends the response to the challenger.

Note that, we have enlarged the adversary's ability here, by allowing it to issue potentially an exponential number of queries to the oracle $\mathcal{O}$. In fact, a polynomial-time adversary can only issue a polynomial number of queries. Nonetheless, if such an extensively empowered adversary cannot win the game, then a polynomial-time adversary can neither.

Suppose, at the end of the game, the adversary performs $d^* - 1$ steps as above (which also means the *sequential computing time* is $d^* - 1$), then the set $\mathcal{S}$ is a subset of

$$\{\sigma((g^*)^{-2^{d^*-1}}), \sigma((g^*)^{-2^{d^*-1}+1}), \ldots, \sigma(1), \ldots, \sigma((g^*)^{2^{d^*-1}-1}), \sigma((g^*)^{2^{d^*-1}})\}.$$

In the game, the adversary could choose to submit a value $\sigma((g^*)^z)$ from $\mathcal{S}$ or a value r from $\{0,1\}^{|pq|} \setminus \mathcal{S}$ as the response to the challenger.

- In the first case, if $\sigma((g^*)^z) = \sigma((g^*)^{2^{d^*}})$, then the adversary has actually found $2^{d^*} - z$ such that $(g^*)^{2^{d^*}-z} \equiv 1 \pmod{pq}$. Based on the extended discrete logarithm assumption, the probability is negligible.

- In the second case, if $(g^*)^{2^{d^*}}$ has not been queried to the oracle, the probability $r = \sigma((g^*)^{2^{d^*}})$ is (Q/pq), where Q is the total number of queries issued to the oracle $\mathcal{O}$. Clearly, this probability is negligible. Otherwise, if $(g^*)^{2^{d^*}}$ has been queried to the oracle $\mathcal{O}$, the probability $r = \sigma((g^*)^{2^{d^*}})$ is 0.

To summarize, in this simple situation, the adversary's advantage is negligible.

Next, we evaluate a more complex situation where the adversary is free to issue PuzzleGen oracle queries. This reflects the practical situation that a prover can ask the verifier for puzzles in a computational puzzle scheme. Our analysis will show that such a privilege does not help the adversary gain any benefit. Let $\sigma(g_1), \sigma(g_2), \ldots, \sigma(g_V)$ be the elements resulted from the replies of PuzzleGen oracle queries, where $g_1, g_2, \ldots, g_V$ are random elements from $\mathbb{Z}^*_{pq}$. Certainly, V is a polynomial in the security parameter. Suppose, at the end of the game, the adversary performs $d^* - 1$ steps in the Response phase, then the set $\mathcal{S}$ is a subset of $\mathcal{S}_1 \bigcup \mathcal{S}_2 \bigcup \mathcal{S}_3$. The set $\mathcal{S}_1$ is the following:

$$\mathcal{S}_1 = \{\sigma((g^*)^{-2^{d^*-1}}), \sigma((g^*)^{-2^{d^*-1}+1}), \ldots, \sigma(1), \ldots, \sigma((g^*)^{2^{d^*-1}-1}), \sigma((g^*)^{2^{d^*-1}})\}.$$

The set $\mathcal{S}_2$ contains a polynomial number of encodings of the form $\sigma(\prod_{x_i \in \mathbb{N}}^{1 \le i \le V} g_i^{x_i})$, and the set $\mathcal{S}_3$ contains a polynomial number of encodings of the form $\sigma(A \cdot B)$, where $\sigma(A) \in \mathcal{S}_1$ and $\sigma(B) \in \mathcal{S}_2$.

In the game, the adversary could choose to submit a value $\sigma((g^*)^z)$ from $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$, or a value r from $\{0,1\}^{|pq|} \setminus \mathcal{S}$ as the response to the challenger.

- In the first case, from the analysis in the simple situation, the adversary's advantage is negligible.
- In the second and the third cases, if $\sigma((g^*)^z \cdot \prod_{x_i \in \mathbb{N}}^{1 \le i \le V} g_i^{x_i}) = \sigma((g^*)^{2^{d^*}})$, then the adversary has actually found $2^{d^*} - z$ such that $(g^*)^{2^{d^*}-z} \equiv \prod_{x_i \in \mathbb{N}}^{1 \le i \le V} g_i^{x_i} \pmod{pq}$. Based on the extended discrete logarithm assumption, the probability is negligible.
- In the forth case, from the analysis in the simple situation, the adversary's advantage is negligible.

As a result, in this situation, the adversary's advantage is negligible. The theorem follows. ■

5. Conclusion

In this paper, we have revisited the concept of computational puzzle schemes, proposed a security model and presented formal definitions for two important properties, namely the determinable difficulty property and the parallel computation resistance property. We have proved that a variant of the RSW client puzzle scheme achieves both properties. To our knowledge, this is the first scheme which has been proved possessing the parallel computation resistance property. Compared with many other schemes such as the hash-based ones [4,20], the RSW scheme is computationally more expensive for the verifier, which needs to perform one exponentiation in verifying a puzzle solution. Therefore, an interesting direction is to investigate how to improve the verifier's efficiency in pracice. Another interesting direction is to analyse the memory-bound computational puzzle schemes in our security model.

Note

1. This is a simplied version of the scheme in [20].

References

[1] M. Abadi, M. Burrows, M. Manasse, and T. Wobber, *Moderately hard, memory-bound functions*, ACM Trans. Internet Technol. 5(2) (2005), pp. 299–327.

[2] T. Aura, P. Nikander, and J. Leiwo, *DoS-resistant authentication with client puzzles*, Security Protocols, 8th International Workshop, Cambridge, UK, 2000, pp. 170–177.

[3] L. Babai and E. Szemeredi, *On the complexity of matrix group problems i*, Annual IEEE Symposium on Foundations of Computer Science, Singer Island, Florida, 1984, pp. 229–240.

[4] A. Back, *Hashcash –amortizable publicly auditable cost functions*, 2002. Available at http://www.hashcash.org/papers/amortizable.pdf

[5] D. Boneh and M. Naor, *Timed Commitments*, CRYPTO '00: Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology, Springer, Berlin, Heidelberg, 2000, pp. 236–254.

[6] N. Borisov, *Computational Puzzles as Sybil Defenses*, P2P '06: Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing, IEEE Computer Society, Washington, DC, 2006, pp. 171–176.

[7] S. Brands, *Untraceable off-line cash in wallets with observers (extended abstract)*, in *Advances in Cryptology – CRYPTO 1993, Santa Barbara, California, USA*, Lecture Notes in Computer Science, Vol. 773, D.R. Stinson, ed., Springer, Berlin, Heidelberg, 1993, pp. 302–318.

[8] J. Cai, R.J. Lipton, R. Sedgewick, and A.C. Yao, *Towards uncheatable benchmarks*, Structure in Complexity Theory Conference, San Diego, CA, 1993, pp. 2–11.

[9] L. Chen, P. Morrissey, N. Smart, and B. Warinschi, *Security notions and generic constructions for client puzzles*, Advances in Cryptology – Asiacrypt 2009, Lecture Notes in Computer Science, Vol. 5912, Springer, Berlin, Heidelberg, 2009, pp. 505–523.

[10] D. Dean and A. Stubblefield, *Using Client Puzzles to Protect TLS*, SSYM'01: Proceedings of the 10th Conference on USENIX Security Symposium, USENIX Association, Berkeley, CA, 2001, pp. 1–1.

[11] S. Doshi, F. Monrose, and A.D. Rubin, *Efficient memory bound puzzles using pattern databases*, Applied Cryptography and Network Security, 4th International Conference, ACNS 2006, 2006, pp. 98–113.

[12] C. Dwork, A. Goldberg, and M. Naor, *On memory-bound functions for fighting spam*, in *Advances in Cryptology – CRYPTO 2003, Santa Barbara, California, USA*, Lecture Notes in Computer Science, Vol. 2729, Dan Boneh, ed., Springer, Berlin, Heidelberg, 2003, pp. 426–444.

[13] C. Dwork and M. Naor, *Pricing via processing or combatting junk mail*, in *Advances in Cryptology – CRYPTO '92, Santa Barbara, California, USA*, Lecture Notes in Computer Science, Vol. 740, E.F. Brickell, ed., Springer, Berlin, Heidelberg, 1992, pp. 139–147.

[14] W. Feng, E. Kaiser, W. Feng, and A. Luu, *The Design and Implementation of Network Puzzles*, Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2005), 2005, pp. 2372–2382.

[15] M.K. Franklin and D. Malkhi, *Auditable metering with lightweight security*, in *Financial Cryptography, First International Conference, FC '97, Proceedings, Anguilla, British West Indies*, Lecture Notes in Computer Science, Vol. 1318, R. Hirschfeld, ed., Springer, Berlin, Heidelberg, 1997, pp. 151–160.

[16] N.A. Fraser, D.J. Kelly, R.A. Raines, R.O. Baldwin, and B.E. Mullins, *Using client puzzles to mitigate distributed denial of service attacks in the tor anonymous routing environment*, IEEE International Conference on Communications, ICC '07, 2007, pp. 1197–1202.

[17] J.A. Garay and M. Jakobsson, *Timed release of standard digital signatures*, in *Financial Cryptography, 6th International Conference, FC 2002, Southampton, Bermuda*, Lecture Notes in Computer Science, Vol. 2357, M. Blaze, ed., Springer, Berlin, Heidelberg, 2002, pp. 168–182.

[18] D. Goldschlag and S. Stubblebine, *Publicly Verifiable Lotteries: Applications of Delaying Functions*, FC '98: Proceedings of the Second International Conference on Financial Cryptography, Springer-Verlag, Berlin, Heidelberg, 1998, pp. 214–226.

[19] M. Jakobsson and A. Juels, *Proofs of Work and Bread Pudding Protocols*, CMS '99: Proceedings of the IFIP TC6/TC11 Joint Working Conference on Secure Information Networks, Kluwer, B.V., Deventer, The Netherlands, 1999, pp. 258–272.

[20] A. Juels and J.G. Brainard, *Client Puzzles: A Cryptographic Countermeasure against Connection Depletion Attacks*, Proceedings of the Network and Distributed System Security Symposium, NDSS 1999, 1999, pp. 151–165.

[21] B. Laurie and R. Clayton, *Proof-of-work proves not to work*, Third Annual Workshop on Economics of Information Security (WEIS04), Minneapolis, MN, USA, 2004.

[22] G. Leander and A. Rupp, *On the equivalence of rsa and factoring regarding generic ring algorithms*, in *Advances in Cryptology – ASIACRYPT 2006, Shanghai, China*, Lecture Notes in Computer Science, Vol. 4284, X. Lai and K. Chen, eds., Springer, Berlin, Heidelberg, 2006, pp. 241–251.

[23] M. Lee and C. Fung, *A public-key based authentication and key establishment protocol coupled with a client puzzle*, J. Am. Soc. Inf. Sci. Technol. 54(9) (2003), pp. 810–823.

[24] Y. Lei, S. Pierre, and A. Quintero, *Client Puzzles Based on Quasi Partial Collisions against DoS Attacks in UMTS*, Proceedings of the 64th IEEE Vehicular Technology Conference, Montreal, Canada, 2006, pp. 1–5.

[25] D. Liu and L. Jean Camp, *Proof of Work can Work*, Fifth Workshop on the Economics of Information Security (WEIS06), Cambridge, UK, 2006.

[26] I. Martinovic, F.A. Zdarsky, M. Wilhelm, C. Wegmann, and J.B. Schmitt, *Wireless Client Puzzles in IEEE 802.11 Networks: Security by Wireless*, WiSec '08: Proceedings of the first ACM Conference on Wireless Network Security, Alexandria, VA, USA, ACM, New York, NY, 2008, pp. 36–45.

[27] R.C. Merkle, *Secure communications over insecure channels*, Commun. ACM, 21(4) (1978), pp. 294–299.
[28] V.I. Nechaev, *Complexity of a determinate algorithm for the discrete logarithm*, Math. Notes 55(2) (1994), pp. 91–101.
[29] P. Ning, A. Liu, and W. Du, *Mitigating dos attacks against broadcast authentication in wireless sensor networks*, ACM Trans. Sens. Netw. 4(1) (2008), pp. 1–35.
[30] R.L. Rivest, A. Shamir, and D.A. Wagner, *Time-lock puzzles and timed-release crypto*, Tech. Rep. MIT/LCS/TR-684, Massachusetts Institute of Technology, 1996.
[31] V. Shoup, *Lower bounds for discrete logarithms and related problems*, in *Advances in Cryptology – EUROCRYPT '97, Konstanz, Germany*, Lecture Notes in Computer Science, Vol. 1233, Springer, Berlin, Heidelberg, 1997, pp. 256–266.
[32] S. Tritilanunt, C. Boyd, E. Foo, and J.M. González Nieto, *Toward non-parallelizable client puzzles*, Cryptology and Network Security, 6th International Conference, CANS 2007, 2007, pp. 247–264.
[33] X. Wang and M.K. Reiter, *Defending Against Denial-of-service Attacks with Puzzle Auctions*, SP '03: Proceedings of the 2003 IEEE Symposium on Security and Privacy, IEEE Computer Society, Washington, DC, 2003, pp. 78–92.
[34] X. Wang and M.K. Reiter, *Mitigating Bandwidth-exhaustion Attacks using Congestion Puzzles*, CCS '04: Proceedings of the 11th ACM conference on Computer and Communications Security, 2004, pp. 257–267.
[35] X. Wang and M.K. Reiter, *A multi-layer framework for puzzle-based denial-of-service defense*, Int. J. Inf. Secur. 7(4) (2008), pp. 243–263.
[36] B. Waters, A. Juels, J.A. Halderman, and E.W. Felten, *New Client Puzzle Outsourcing Techniques for DoS Resistance*, Proceedings of the 11th ACM Conference on Computer and Communications Security, CCS 2004, 2004, pp. 246–256.