A Guided Tour Puzzle for Denial of Service Prevention

Mehmud Abliz[*]   Taieb Znati[*][†]

[*]*Department of Computer Science*
[†]*Telecommunication Program*
University of Pittsburgh
Pittsburgh, Pennsylvania 15260
{*mehmud, znati*}*@cs.pitt.edu*

Abstract—**Various cryptographic puzzle schemes are proposed as a defense mechanism against denial of service attack. But, all these puzzle schemes face a dilemma when there is a large disparity between the computational power of attackers and legitimate clients: increasing the difficulty of puzzles might unnecessarily restrict legitimate clients too much, and lower difficulty puzzles cannot sufficiently block attackers with large computational resources. In this paper, we introduce *guided tour puzzle*[1], a novel puzzle scheme that is not affected by such resource disparity. A guided tour puzzle requires a client to visit a predefined set of nodes, called *tour guides*, in a certain sequential order to retrieve an n-piece answer, one piece from each tour guide that appears in the tour. This puzzle solving process is non-parallelizable, thus cheating by trying to solve the puzzle in parallel is not possible. Guided tour puzzle not only achieves all previously defined desired properties of a cryptographic puzzle scheme, but it also satisfies more important requirements, such as *puzzle fairness* and *minimum interference*, that we identified. The number of tour guides required by the scheme can be as few as two, and this extra cost can be amortized by sharing the same set of tour guides among multiple servers.**

I. INTRODUCTION

A denial of service (DoS) or a distributed denial of service (DDoS) attack is an attempt by malicious parties to prevent legitimate users from accessing a service, usually by depleting the resources of the server which hosts that service and making the service unavailable to legitimate users. The targeted resource of the attack can be bandwidth, CPU, memory, disk capacity, or combination of the above. In many cases, denial of service attacks are easy to mount in that the server commits significant amount of resources to process a request which consumes very few resources to generate by a client.

Cryptographic puzzles are proposed to defend against such denial of service attacks by better balancing the computational load of client and server. In a cryptographic puzzle scheme, a client is required to solve a cryptographic puzzle and submit the puzzle solution as proof of work before the server commits substantial resources to its request. A cryptographic puzzle is a computational problem whose solution

requires moderate amount of cryptographic operations from the solver, and the amount of work required is guaranteed by the security of both the puzzle construction method and the cryptographic algorithm used. In most puzzle schemes, each puzzle requires an approximately fixed number of cryptographic operations, such as hashing, modular multiplication, or modular exponentiation, to compute the puzzle solution. Thus, the more an attacker wants to overwhelm the server, the more puzzles she has to compute, consequently the more computational resources of her own she needs to consume. The construction and verification of the puzzle are designed to be very efficient to avoid DoS on the puzzle scheme itself.

Since first introduced by Dwork and Naor in [1] to combat junk e-mails, cryptographic puzzles are extended to defeat various attacks such as denial of service, Sybil attacks etc. Moreover, new ways of constructing and distributing puzzles are introduced one after another. But none of the proposed schemes so far tried to solve what we call the *resource disparity* problem. A resource disparity problem in puzzle schemes appear when there exists a large disparity between the computational power of attackers and legitimate clients, significantly reducing the effectiveness of cryptographic puzzles against DoS attacks. We use an example of the most commonly used puzzle, hash-reversal puzzle, to explain the problem.

In a hash reversal puzzle, such as [2][3][4][5][6], a puzzle with difficulty d takes on average 2^{d-1} hash operations to compute its solution. Say a server can handle 1000 requests per second, and a subset of clients have limited resources, and can perform 10^5 hash operations per second at maximum. Assuming these clients can spend at most 50% of their computational power on computing puzzles, then the server should allow at least $\frac{2^{20}}{0.5 \times 10^5} \approx 20$ seconds between sending a puzzle and receiving its solution, for a hash-reversal puzzle with difficulty $d = 21$. An attacker with an ASIC designed for hash computation can perform 10^9 hash operations per second [7], so she can compute $\frac{20sec}{2^{20}/10^9} = 20,000$ puzzles with the same difficulty during that 20-second period. This means an attacker with just one such ASIC component can easily overwhelm the server. For more powerful servers, an attacker can still launch successful denial of service attack using multiple such dedicated devices.

---

[1]It is called guided tour puzzle because completing a guided tour is like solving a maze (a tour puzzle) to a malicious client that does not follow the rules of the puzzle scheme.

The server can try to further block such attackers by increasing the difficulty of the puzzle, for example setting $d = 30$, but that will restrict the legitimate clients too much. Making it worse, some clients with limited computational power might never be able to solve the puzzle on time, hence never being able to get service. In fact, the server should not even expect a client to make 50% of its computational power available for puzzle computation, since a client should use majority of its resources on normal user operations. Thus, even assuming the ratio of computational powers of legitimate clients and malicious clients is a small number, such as 40 [8], the disparity between their *available* computational power is amplified to a far bigger number due to the fact that only a limited percentage of the resources are available for puzzle computation at legitimate clients whereas malicious clients can try to make the most of their already strong computational power.

In this paper, we introduce a novel puzzle scheme, called *guided tour puzzle*, that is not affected by the resource disparity problem. We use the network round-trip delay as the unit work in the puzzle solving process, and require a client to complete a guided tour by visiting a predefined set of nodes, called *tour guides*, in a certain sequential visiting order. Contribution of our work is the following:

- We study the state of the art in the cryptographic puzzles, and identify a comprehensive list of requirements that a puzzle scheme should satisfy.
- We further extend this list by defining two highly desired properties of a puzzle scheme that have not been considered by previous works.
- We explore a novel approach to designing cryptographic puzzles by introducing guided tour puzzle. Through analysis and experiments, we show that guided tour puzzle achieves the desirable properties of a good puzzle scheme. In particular, we show how guided tour puzzle achieves *puzzle fairness* and *minimum interference* properties, and why achieving them is essential to the effectiveness of a puzzle scheme in preventing denial of service attacks.
- We also provide a comprehensive survey of various cryptographic puzzle schemes.

The rest of the paper is organized as follows. Section II discusses a comprehensive set of requirements of a puzzle scheme. Section III defines design goals of guided tour puzzle and assumed threat model, followed by a formal introduction of basic guided tour puzzle. Several improvements to the basic scheme is proposed in Section IV. In Section V, we use analysis and measurement techniques to show that guided tour puzzle satisfies our requirements and design goals. Section VI provides an overview of the related work in cryptographic puzzle area, and we discuss future improvements to our scheme and give a conclusion of the paper in Section VII.

II. DESIRED PROPERTIES

Various cryptographic puzzle schemes are proposed so far, each focusing only on a subset of requirements for cryptographic puzzles. We made an endeavor to provide a more comprehensive list of requirements that a cryptographic puzzle scheme should satisfy.

A. General Properties

We use *general properties* to refer to puzzle properties that are discussed to some extent in the existing literature.

Computation guarantee. The computation guarantee means a cryptographic puzzle guarantees a lower and upper bound on the number of cryptographic operations spent on a client to find the puzzle answer. In other words, a malicious client should not be able to solve a puzzle spending significantly less number of operations than required.

Efficiency. The construction, distribution and verification of a puzzle by the server should be efficient, in terms of CPU, memory, bandwidth, hard disk etc. Specifically, puzzle construction, distribution and verification should add minimal overhead to the server to prevent the puzzle scheme from becoming an avenue for denying service [3].

Adjustability of difficulty. This property is also referred to as puzzle granularity [9]. Adjustability of puzzle difficulty means the cost of solving the puzzle can be increased or decreased in fine granularity. Adjustability of difficulty is important, because finer adjustability enables the server to achieve better trade-off between blocking attackers and the service degradation of legitimate clients.

Correlation-free. A puzzle is considered correlation-free if knowing the solutions to all previous puzzles seen by a client does not make solving a new puzzle any easy. Apparently, if a puzzle is not correlation-free, then it allows malicious clients to solve puzzles faster by correlating previous answers.

Stateless. A puzzle is said to be stateless if it requires the server to store no client information or puzzle-related data in order to verify puzzle solutions. Requiring the server to use a small and fixed memory for storing such information is also acceptable in most cases.

Tamper-resistance. A puzzle scheme should limit replay attacks over time and space. Puzzle solutions should not be valid indefinitely and should not be usable by other clients [3].

Non-parallelizability. Non-parallelizability means the puzzle solution cannot be computed in parallel using multiple machines [9]. Non-parallelizable puzzles can prevent attackers from distributing computation of a puzzle solution to a group of machines to obtain the solution quicker.

B. Puzzle Fairness and Minimum Interference

Puzzle fairness and minimum interference are two very important properties that are little discussed and not addressed by previous schemes.

Puzzle fairness. Puzzle fairness means a puzzle should take same amount of time for all clients to compute, regardless of their CPU power, memory size, and bandwidth. If a puzzle can achieve fairness, then a powerful DoS attacker can effectively be reduced to a legitimate client. Not being able to achieve fairness leads to the resource disparity problem we mentioned earlier.

Minimum interference. This property requires that puzzle computation at the client should not interfere with user's normal operations. If a puzzle scheme takes up too much resources and interfere with users' normal computing activity, users might disable the puzzle scheme or even try to avoid using any service that deploys such a puzzle scheme.

III. GUIDED TOUR PUZZLE

In this section, we first describe our goals in designing guided tour puzzle, and introduce a threat model that is stronger than a model assumed by previous puzzle schemes. Next, we present a basic version of the guided tour puzzle.

A. Design Goals

Although we introduce guided tour puzzle in a denial of service prevention setting, we expect tour puzzles to be used to defend against various attacks such as e-mail spams, Sybil attacks etc. With this in mind, guided tour puzzle aims to achieve all the desired properties of cryptographic puzzles introduced in Section II. Among those properties, *puzzle fairness* and *minimum interference* are not addressed by any previous puzzle scheme, and are our main goals in designing guided tour puzzle. Moreover, guided tour puzzle strives to achieve better effectiveness against both DoS and DDoS attacks.

B. Threat Model

Our threat model assumes a stronger attacker than all previous schemes. First we assume an attacker can have best commercially available hardware and bandwidth resources. Meanwhile, attacker can coordinate all of her available computation resources perfectly so as to take maximum advantage of the resources. Next, attacker can eavesdrop on all messages sent between a server and any legitimate client. We assume that attacker can modify only a limited number of clients' messages that are sent to the server. This assumption is reasonable since if an attacker can modify all clients' messages, then she can launch DoS much easily just by dropping all messages sent by all clients to the server.

Attacker can attack any part of the puzzle scheme, whether it is puzzle construction, puzzle distribution, or puzzle verification. Attacker can try to launch denial of service attack on new components that introduced by our puzzle scheme. Attacker may also attempt to solve puzzles faster than legitimate clients using various methods, such as guessing, correlating previous puzzle answers etc.

C. Basic Scheme

We consider an Internet-scale distributed system of clients and servers. *Attacker* is a malicious entity whose aim is to prevent legitimate clients from receiving service of a server. A server operates as a standard server if it is not under attack. When the server suspects it is under attack or its load is above certain threshold, it replies to all client requests with a 'service restricted' message, indicating that a puzzle needs to be solved in order to receive service. A client then completes a tour puzzle by visiting a set of special nodes called *tour guides*, in a certain sequential order. Nodes within or outside of a server's domain can assume the role of tour guides. A single tour guide might appear multiple times during a tour, so a *stop* is used to represent a single appearance of a tour guide in a tour.

A client computes the index of the tour guide at the first stop using a hash value inside the server's 'service restricted' message, and it can lookup the address of the tour guide at the first stop, or any other tour guide for that matter, from a mapping of indices to tour guide address. Then starting from the first stop, the client contacts each stop and receives a reply message each containing a unique hash value. The hash value in the reply message from previous stop is used for computing the index of the next stop tour guide, and also sent to the next stop as one of the inputs to the calculation of the next hash value. Reply message from the tour guide at the last stop contains a hash value, which will be sent to the server as puzzle answer. The server grants the client service if the answer is valid. The rest of the section describes guided tour puzzle in more detail using formal notations.

First, we set up N tour guides in the system, where N must be at least two. The server establishes a shared secret k_{js} with each tour guide G_j using a secure channel, where $0 \leq j < N$. The server also generates a short-lived secret key K_s for calculating the first hash value returned to a client in a tour. The difficulty of a puzzle is controlled by the tour length L in the guided tour puzzle. Figure 1 shows an example of a guided tour when $N = 2$ and $L = 5$.
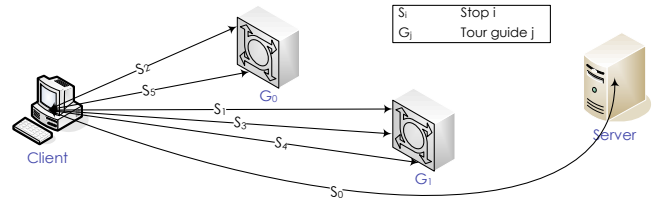

Figure 1. Example of a guided tour when $N = 2$ and $L = 5$. The order of the tour is: $G_1 \rightarrow G_0 \rightarrow G_1 \rightarrow G_1 \rightarrow G_0$.

Notations we used for describing guided tour puzzle is summarized in Table I.

1) Operation at the server:

- When the server receives a request from client x without a puzzle answer attached, it replies with a 2-

Table I

NOTATIONS USED IN DESCRIBING GUIDED TOUR PUZZLE SCHEME.

A_x	Address of client x
G_j	j-th tour guide, where $0 \le j < N$
N	Number of tour guides in the system
k_{js}	Shared key between the server and j-th tour guide
K_s	Secret key only known to the server
S_l	Index of the tour guide at the l-th stop, where $0 \le l < L$
L	Length of a guided tour
ts	Timestamp
$hash$	A cryptographic hash function
B	Size of a $hash$ digest in bits

tuple (h_0, L) as a puzzle. The server computes hash value h_0 by

$$h_0 = hash(A_x||L||ts||K_s) \tag{1}$$

where, $||$ means concatenation, A_x is the address (or any unique value) of the client x, ts is a coarse timestamp, and $hash$ is a cryptographic hash function such as SHA-1 [10]. The granularity of ts can be decided based on the minimum time it takes to complete the tour on average.

- When the server receives a request from client x with a puzzle answer (h_0, h_L) attached, it first checks to see if h_0 sent by the client is equal to the h_0 computed using formula (1). If so, the server further verifies h_L by computing h_L using formula (2). Since the server has shared keys $k_{1s}, k_{2s}, \ldots, k_{Ns}$, it can compute the chain of hashes without contacting any tour guide.

2) Operation at a tour guide: When a tour guide G_j receives a 3-tuple $(h_l, l+1, L)$ from a client x, it replies with a hash value h_{l+1} which is computed by

$$h_{l+1} = hash(h_l||l+1||L||A_x||ts||k_{js}) \tag{2}$$

where, $l+1$ means the tour guide G_j is at the $(l+1)$-th stop of the client's tour, and $0 \le l < L$.

3) Operation at a client:

- When client x receives a 2-tuple (h_0, L) from the server as a reply to its service request, the client understands it has to finish a length L guided tour before it can receive service. So, client starts its tour by computing S_1, the index of the first tour guide in the tour, using formula (3). The client then sends a 3-tuple $(h_0, 1, L)$ to the tour guide G_{S_1}.
- When client x receives a hash value h_l from a tour guide during its tour, it computes the index of the next tour guide S_{l+1} by

$$S_{l+1} = (h_l \bmod N) \tag{3}$$

The client then sends a 3-tuple $(h_l, l+1, L)$ to the tour guide $G_{S_{l+1}}$.
- When client x receives hash value h_L from the tour guide at the last stop of the tour, it re-sends its original

service request together with the puzzle answer (h_0, h_L) to the server. The client receives service after its puzzle answer verified by the server.

Each hash value in a tour puzzle is computed using the previous hash value as one of its inputs. A client is forced to wait until it receives h_l from the l-th stop before it can ask $(l+1)$-th stop to send h_{l+1}. Thus the main time consumed at the client is the round-trip delay, and the time spend at the client for calculating the next stop is trivial.

Next, we present a few improvements to the basic scheme we just introduced.

IV. IMPROVEMENTS TO THE BASIC SCHEME

Although our basic scheme already has many advantages over existing puzzle schemes, there is still room for much improvement. In this section, we first improve the fault-tolerance and robustness of tour guides. Then, we modify the puzzle verification at the server to increase its verification efficiency.

A. Tour Guides

In the guided tour puzzle scheme, we introduced multiple tour guides to share the server's workload of managing puzzles. An attacker might adopt her strategy and attack one of the tour guides, and indirectly launch DoS attack on the server. Moreover, in the basic scheme, the failure of a single tour guide affects many clients in the system. Specifically, the probability of a failed tour guide never appearing in a single tour is $(\frac{N-1}{N})^L$, assuming that a tour guide appearing at the stop i and stop j are independent events for any $i < L, j < L$ and $i \ne j$. The probability of none of the M tours include the failed tour guide is $(\frac{N-1}{N})^{M \times L}$, which is a very small number for a large value of M and L.

We propose two improvements to the basic scheme in order to achieve better fault-tolerance as well as robustness against DoS on the tour guides.

1) Two operation modes of tour guides: This improvement defines each tour guide to operate in two different modes: *active* and *tarpit*. When a tour guide is in active mode, it behaves the same as in the basic scheme with slight difference. However, when a tour guide is in tarpit mode, it prevents a malicious client from completing any tour and directs the malicious client to other tarpit tour guides to keep it busy. A tour guide considers a client malicious if the client contacts it during its tarpit mode period, since an honest client who is following the exact guidance of active tour guides will never be directed to a tarpit tour guide.

We divide the time into smaller time periods of length t, where t is on the order of several minutes, and use T_i to denote the i-th time period. All tour guides share a common secret k_{cs} with the server. The server or any tour guide can compute the mode a_{G_j} of a tour guide G_j for the time period T_i, by

$$a_{G_j} = LSB[hash(j||T_i||k_{cs})] \tag{4}$$

where, j is the index of the tour guide G_j, $LSB[\cdot]$ means the least significant bit. This state information can be stored in an array $TGS[\ ]$, where $TGS[j] = a_{G_j}$. As we can see, a_{G_j} takes a value of 0 or 1, where 0 denotes a tarpit mode and 1 denotes an active mode. Since lower bits of output of a secure hash function is uniformly distributed [11], we can be certain that on average 50% of the tour guides will be in active mode, while the remaining half is in the tarpit mode.

The operation on a client is essentially the same as in the basic scheme, except now the client does not have to compute the index of next tour guide since it will be given in the reply message from the previous tour guide. Only the operations on the server and tour guides that are different from the ones in the basic scheme are described in the following.

Operation on the server: When the server receives a request from client x without a puzzle answer attached, it replies with a 3-tuple (h_0, L, S_1), where S_1 is computed using the function introduced next.

Operation on a tour guide: When a tour guide G_j receives a 3-tuple $(h_l, l + 1, L)$ from a client x, there are two cases depending on the current mode of G_j.

- $TGS[j] = 1$ (active). The tour guide replies with a 2-tuple (h_{l+1}, S_{l+1}), where the index S_{l+1} can be computed using the simple function given below:

  $d \Leftarrow h_{l+1} \bmod N$
  while $TGS[d] \neq 1$ **do**
  $\quad d \Leftarrow d + 1$
  end while
  $S_{l+1} \Leftarrow d$

- $TGS[j] = 0$ (tarpit). The tour guide replies with a 2-tuple $(r, \hat{S}_{l+1})$, where r is a B bit random number, $\hat{S}_{l+1}$ is computed in the exact opposite way of computing S_{l+1}. That is, instead of finding the index of next active tour guide, it finds the index of next tarpit.

The rationale behind adopting a tarpit mode for tour guide is as follows. Attacking a tarpit tour guide does not have any effect on legitimate clients, since legitimate clients will never visit a tarpit tour guide. A better strategy for the attacker is to figure out the tour guides in active mode during each time period, and only attack these active guides. In order to do that, an attacker has to follow the exact guidance of the tour guides to finish multiple tours during each T_i, since one tour might not include all active tour guides for that time period. Therefore, by the time the attacker figures out all active tour guides by completing one or multiple tours, an active tour guide will most likely switch into tarpit mode.

2) Puzzle construction at the server: We can further improve the fault-tolerance of tour guides by allowing a client to contribute some randomness to the computation of the hash value h_0 in formula (1). The client x sends a randomly generated nounce n_x to the server, and the server will compute h_0 using the new formula

$$h_0 = hash(n_x||A_x||L||ts||K_s) \qquad (5)$$

By sending different n_x, client x can affect the value h_0, consequently affecting the entire tour in an 'uncertain' manner. Here, 'uncertain' means that client x can experience different guided tour by sending different n_x, but it cannot decide which tour guides will be in the tour and in what order. The benefit of adopting such a modification is that a client can try to avoid the failing tour guide by changing n_x and trying to have different tour. The client sends n_x to the server as part of the puzzle answer at the end, hence the server does not have to remember it.

B. Puzzle Verification at the Server

We can improve the efficiency of the puzzle answer verification at the server by letting the server to pre-compute puzzle answers during idle CPU cycles, and store the answer in a bloom filter [12] indexed by A_x. To avoid an attacker launch memory exhaustion attack on the server, we can use a fixed-size bloom filter. Using a 0.1% false positive bloom filter that uses 14.4 bits to store an element, we can store more than $\frac{2^{20} \times 8}{2^4} = 2^{19}$ puzzle answers in a 1MB bloom filter. Now the verification of puzzle answer takes only a single memory lookup.

V. ANALYSIS

In this section we use analytical reasoning and experiment results to show how guided tour puzzle can meet our proposed design goals.

A. General Puzzle Properties

For each property, we briefly explain how that property is achieved in guided tour puzzle.

Computation guarantee. Each client is required to compute L modulo operation in order to find out the next tour guide in a tour. Since this operation is the easiest way to find the right tour guides, there is no other way that takes lesser number of operations, achieving computation guarantee for all clients.

Efficiency. In guided tour puzzle, construction of a puzzle takes only a single hash operation to compute h_0 at the server, and verification of a puzzle answer takes one memory lookup in the improved scheme. Transferring of puzzle from server to the client requires $B/8$ plus few extra bytes, where B is usually $160 \sim 256$ bits.

Adjustability of difficulty. The difficulty of a tour puzzle is adjusted by adjusting the tour length L, which can be increased or decreased by one. Therefore, guided tour puzzles provide linear adjustability of difficulty.

Correlation-free. Guided tour puzzles are correlation-free, because knowing all previous puzzle answer does not help solve the current puzzle in any way. This property is
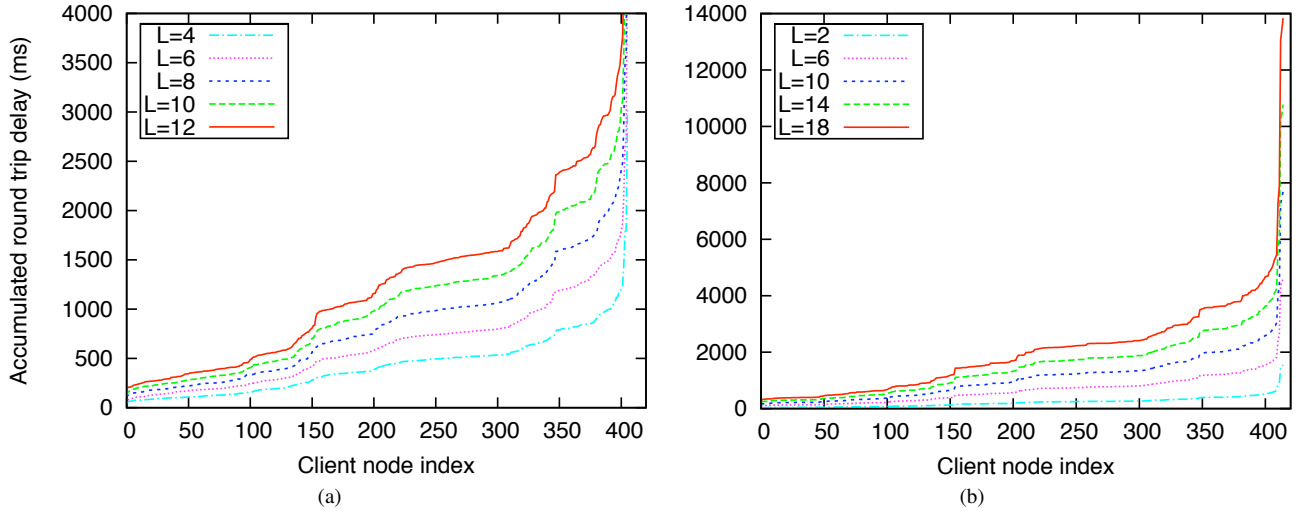
Figure 2. The tour delays of clients. (a) Example of tour delay at a single point in time (20:11:26 on May 23, 2009), the number of tour guides $N = 4$. (b) Average tour delay of all client nodes for two-week period when $N = 4$.

provided by the security of the one-way hash chain we used in guided tour puzzle.

Stateless. Guided tour puzzle does not require the server to store any client or puzzle related information, except for the cryptographic keys that are used for the hash calculation. Puzzle answer verification using memory lookup does require few megabytes of memory space in total, but it is negligible considering the large memory size of modern servers.

Tamper-resistance. The coarse timestamp used in the computation of each h_l guarantees a limited validity period of a puzzle answer. Meanwhile, the puzzle answer computed by one client cannot be used by any other client, since a value unique to each client is included in the computation of each h_l.

Non-parallelizability. Guided tour puzzle cannot be computed in parallel. An attacker with N malicious clients can assign each client to contact one tour guide, and try to compute the puzzle answer in parallel. But each malicious client has to first get a h_l from the tour guide it is responsible for, and sends it to the next malicious client that is responsible for the next tour guide in the tour. Thus even with multiple malicious clients, attacker still has to compute the puzzle answer sequentially.

B. Achieving Puzzle Fairness

In guided tour puzzle, the time delay enforced on a client mainly comes from the round trip to multiple tour guides. The advantage of this is that nobody, not even a powerful attacker, can control the round trip delay occurred in an Internet-scale distributed system. Due to the variation in the round trip delay across multiple clients, it is possible that the sum of round trip delays, which we will refer to as *tour delay* from now on, experienced by an attacker is much

smaller than by a legitimate client for a single tour. However, it can also be the opposite. Meanwhile, the variation in average tour delay across multiple clients is within a small factor as shown next by the experiment results. Although a small variation in the tour delay is inevitable, it cannot be effectively manipulated by an attacker to achieve unfair advantage over legitimate clients, regardless of attacker's CPU, memory, or bandwidth advantage. Therefore, guided tour puzzle achieves a fairness that is far better than any existing puzzle scheme can.

An attacker can try to minimize the puzzle solving time by using multiple malicious clients, where each malicious client is responsible for contacting the tour guide closest to it. But this kind of attacker actually cannot gain significant advantage over a legitimate client, because each malicious client has to wait one round-trip time to get the reply of the tour guide it is closest to, and figure out the index of the next tour guide, then spend another half a round trip delay to send this information to the malicious client closest to the next tour guide. Furthermore, the extra one-way delay is likely to be large, because the next tour guide is more likely to be far from the previous malicious client due to the 'greedy' positioning of malicious clients.

Next, we use experiment results to show that the variation in the tour delays across multiple clients is within a small factor for a large-scale distributed system like Internet. This variation should not be confused with the delay variation across multiple round trips for a fixed sender-receiver pair.

We used measurement data from PlanetLab Scalable Sensing Service (S^3) [13] that are collected over two-week period. PlanetLab has a collection of over 1000 nodes distributed across the globe, and provides a realistic network testbed that experiences congestion, failures, and diverse link
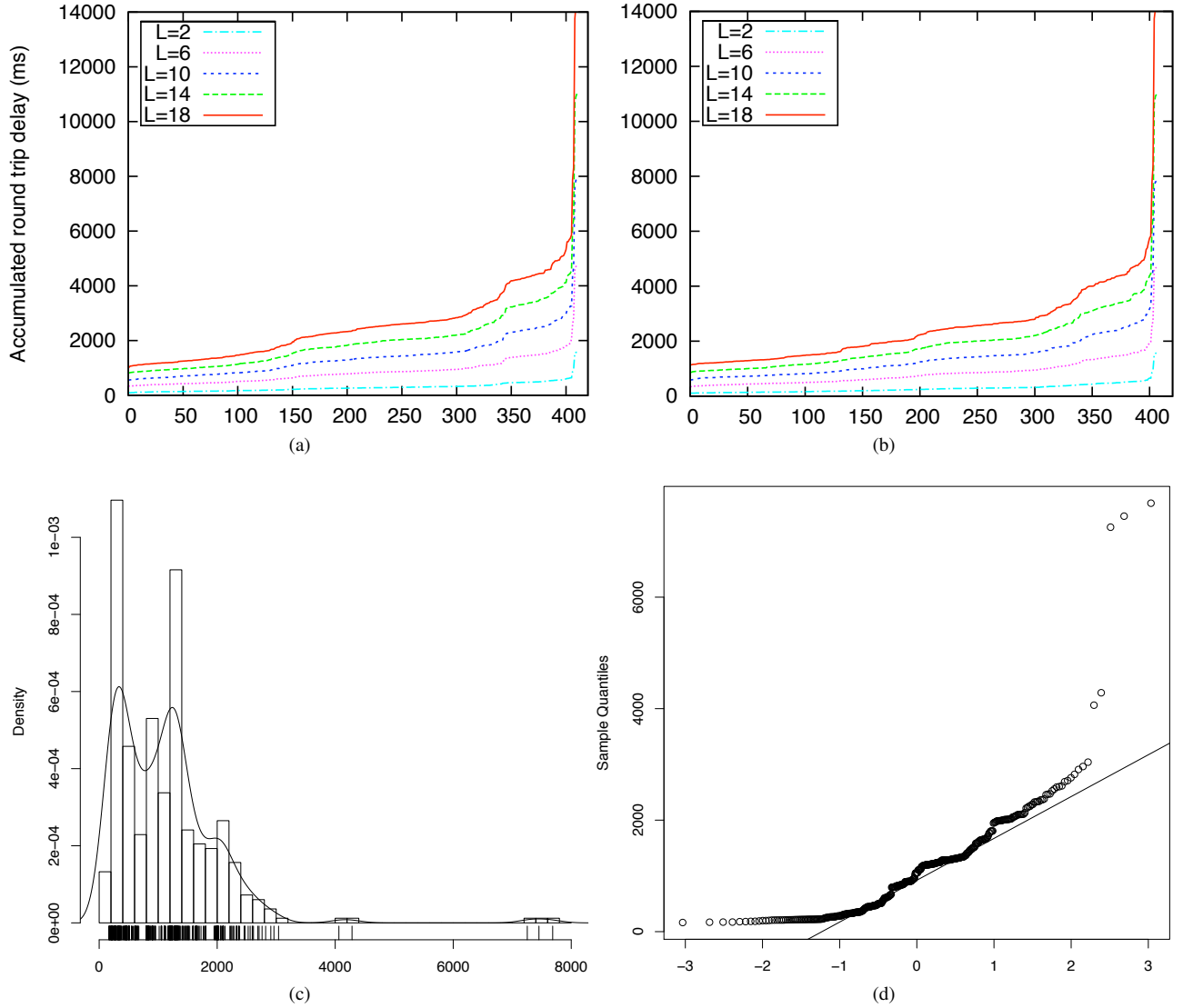
Figure 3. (a), (b) Average tour delay of all clients when the number of tour guides $N = 8$ and $N = 12$ respectively. (c) Probability density of tour delay (unit: millisecond) when $N = 4$ and tour length $L = 10$. (d) Q-Q plot of tour delay against normal distribution when $N = 4$, $L = 10$.

behaviors [14]. S^3 provides end-to-end latency data for all pairs of nodes in PlanetLab. About half of all the PlanetLab nodes have latency data available throughout the two weeks we collected data, so our experiments use these nodes only.

We choose 20 PlanetLab nodes at maximum as tour guides, based on the better connectivity of these nodes to all other PlanetLab nodes, and do not try to optimally pick them to achieve the least delay variation. The remaining PlanetLab nodes are treated as client nodes. The number of tour guides N is varied from 4 to 20, and the tour length L is varied from 2 to 18. For each (N, L) pair, we compute guided tours using formula (1) and (2) for all client nodes, and compute a tour delay for each tour based on the collected data. As an example of tour delay at a single point in time,

Figure 2a shows the tour delays of all client nodes for the setting $N = 4$ and $L = 4, 6, 8, 10, 12$ on May 23, 2009. For this particular data, the ratio of tour delays of the client with the most delay and the client with the least delay is 13, when the 4% clients nodes with exceptionally large delays are excluded.

To give a better idea of how the tour delay vary across clients on average, we averaged tour delays of all clients over two-week period. To find the average tour delay of a client for a specific (N, L) setting, all tour delays of the client for that (N, L) during the two-week period are taken average. Then, the average tour delays are sorted from the least to the most, in order to provide a better view of delay variation across all clients. Figure 2b and Figure 3a and 3b

shows the average tour delays computed using this method for all client nodes when N=4, 8, 12. Results for other values of N are skipped due to space limitation, but they are very similar to the results shown here. When excluding 5% client nodes with exceptionally large delays, the ratio of tour delays of the client with the most delay and the client with the least delay is around 5. This disparity is several orders of magnitude smaller when compared to the disparity in available computational power (which can be in thousands). Figure 3c and 3d shows that majority of tour delays are clustered within a tight area of delay and the distribution of tour delays closely simulates a normal distribution. Overall, experiment results strongly supported our claim about the fairness of guided tour puzzle.

C. Achieving Minimum Interference

In guided tour puzzle scheme, a client has to perform two types of operations: modulo operations for computing the index of the next tour guide, and sending packets to tour guides. To complete a guided tour puzzle with length L, a client only needs to perform L modulo operations plus send and receive a total of $2 \times L$ packets with about 20~32 bytes of data payload, where L is usually a small number below 30. This creates negligible CPU and bandwidth overhead even for small devices such as PDAs or cell phones.

D. Effectiveness Against DoS and DDoS

We now analyze the effectiveness of guided tour puzzle against DoS and DDoS attacks. Since guided tour puzzle can be used independently or in combination with other mechanisms to prevent denial of service attacks at the various levels of the network, we do not discuss our scheme's denial of service capability in terms of a specific layer of the network. One might argue that the bandwidth available to the server can be flooded, making the server unable to use the protection provided by puzzles. We believe that deploying capability-based denial of service prevention schemes [15] [16] in conjunction with guided tour puzzle can prevent such flooding attacks.

1) DoS Attack: If an attacker with single malicious client launches denial of service attack on the server, guided tour puzzle can easily prevent the attack regardless of attacker is spoofing a single or multiple addresses. As with other cryptographic puzzle schemes, guided tour puzzle imposes a commitment on a client before granting service, and effectively controls the request arrival rate. In essence, the commitment imposed on clients by all cryptographic puzzles is time. All previous puzzle schemes try to achieve this time commitment at the client by means of computations that require significant CPU and/or memory overhead, but they unfairly give advantage to strong attackers and are obtrusive to end users. Guided tour puzzle instead achieves this time commitment in a guaranteed and unobtrusive manner, and most of all, all clients have to commit same amount of time

to complete a tour puzzle regardless the amount of resources available to them. Therefore, a single attacker essentially reduced to a single legitimate client, and that only increases the number of legitimate clients by one.

2) DDoS Attack: In a DDoS attack, an attacker perpetrate attack on the victim using multiple malicious clients, thus the power of the attacker is roughly multiplied by the number of malicious clients she has. Previous cryptographic puzzle defense mechanisms against DDoS suffer from the resource disparity problem we discussed in Section I. Since guided tour puzzle achieves puzzle fairness, the number of malicious clients required to send requests at a rate that reaches the server's maximum capacity is orders of magnitude larger than the number of malicious clients required when previous puzzle schemes are used. The protection of guided tour puzzle comes from the fact that it effectively reduces a malicious client into a legitimate client. Of course, attacker can still overwhelm the server when she has enough malicious clients, but so does the same number of legitimate clients.

Fortunately, a server with guided tour puzzle can still prevent itself from crashing, despite the fact that there are too many clients (whether they malicious or not), by increasing the puzzle difficulty and imposing longer delays at the client. Meanwhile, guided tour puzzle achieves minimum degradation in the service quality when compared with previous puzzle schemes. The degradation of service by such large amounts of malicious clients is extremely hard to prevent without being able to differentiate malicious clients from the legitimate ones.

VI. Related Work

Currently there are many different type of DoS and DDoS defense mechanisms such as filtering based [17][18], traceback and pushback based [19][20][21], capability based [16][15] and cryptographic puzzle based defense mechanisms. Due to the enormity of various such proposals, this related work survey only focuses on cryptographic puzzle based mechanisms.

A. Client Puzzles

Dwork and Noar [1] were the first to introduce the concept of requiring a client to compute a moderately hard but not intractable function, in order to gain access to a shared resource. However this scheme is not suitable for defending against the common form of DoS attack due to its allowance of puzzle solution pre-computations.

Juels and Brainard [2] introduced a hash function based puzzle scheme, called *client puzzles*, to defend against connection depletion attack. Client puzzles addresses the problem of puzzle pre-computation. Aura et al. [4] extended the client puzzles to defend DoS attacks against authentication protocols, and Dean and Stubblefield [5] implemented a DoS resistant TLS protocol with the client puzzle extension.

Wang and Reiter [6] further extended the client puzzles to prevention of TCP SYN flooding, by introducing the concept of *puzzle auction*. Price [22] explored a weakness of the client puzzles and its above mentioned extensions, and provided a fix for the problem by including contribution from the client during puzzle generation.

Waters et al. [23] proposed outsourcing of puzzle distribution to an external service called *bastion*, in order to secure the puzzle distribution from DoS attacks. However, the central puzzle distribution can be the single point of failure, and the outsourcing scheme is also vulnerable to the attack introduced by Price [22].

Wang and Reiter [24] used a hash-based puzzle scheme to prevent bandwidth-exhaustion attacks at the network layer. Feng et al. argued in [25] that a puzzle scheme should be placed at the network (IP) layer in order to prevent attacks against a wide range of applications and protocols. And Feng and Kaiser et al. [3] implemented a hint-based hash reversal puzzle at the IP layer to prevent attackers from thwarting application or transport layer puzzle defense mechanisms.

Portcullis [8] by Parno et al. used a puzzle scheme similar to the puzzle auction by Wang [6] to prevent denial-of-capability attacks that prevent clients from setting up capabilities to send prioritized packets in the network. Portcullis moves the puzzle generation to clients, eliminating the puzzle construction overhead at the server. However, clients willing to solve harder puzzles that require more computation are given higher priority, thus giving unfair advantage to powerful attackers.

B. Non-Parallelizable Puzzles

Non-parallelizable puzzles prevents a DDoS attacker that uses parallel computing with large number of compromised clients to solve puzzles significantly faster than average clients. Rivest at al. [26] designed a *time-lock puzzle* which achieved non-parallelizability due to the lack of known method of parallelizing repeated modular squaring to a large degree [26]. However, time-lock puzzles are not very suitable for DoS defense because of the high cost of puzzle generation and verification at the server.

Ma [27] proposed using *hash-chain-reversal puzzles* in the network layer to prevent against DDoS attacks. Hash-chain-reversal puzzles have the property of non-parallelizability, because inverting the digest i in the chain cannot be started until the inversion of the digest $i+1$ is completed. However, construction and verification of puzzle solution at the server is expensive. Furthermore, using a hash function with shorter digest length does not guarantee the intended computational effort at the client, whereas using a longer hash length makes the puzzle impossible to be solved within a reasonable time.

Another hash chain puzzle is proposed by Groza and Petrica [28]. Although this hash-chain puzzle provides non-parallelizability, it has several drawbacks. The puzzle construction and verification at the server is relatively expensive,

and the transmission of a puzzle to client requires high-bandwidth consumption.

More recently Tritilanunt et al. [9] proposed a puzzle construction based on the subset sum problem [29], and suggested using an improved version [30] of *LLL lattice reduction* algorithm by Lenstra et al. [31] to compute the solution. Problems with the subset sum puzzles include high memory requirements and the failure of LLL in dealing with large instance and high density problems.

C. Memory-Bound Puzzles

Abadi et al. [32] argued that memory access speed is more uniform than the CPU speed across different computer systems, and suggested using memory-bound function in puzzles to improve the uniformity of puzzle cost across different systems. Dwork et al. [33] further investigated Abadi's proposal and provided an abstract memory-bound function with a amortized lower bound on the number of memory accesses required for the puzzle solution. Although these results are promising, there are several issues need to be solved regarding memory-bound puzzles.

First, memory-bound puzzles assume a upper-bound on the attacker machine's cache size, which might not hold as technology improves. Increasing this upper-bound based on the maximum cache size available makes the memory-bound puzzles too expensive to compute by average clients. Secondly, deployment of proposed memory-bound puzzle schemes require fine-tuning of various parameters based on a system's cache and memory configurations. Furthermore, puzzle construction in both schemes is expensive, and bandwidth consumption per puzzle transmission is high. Last, but not least, clients without enough memory resources, such as PDAs and cell phones, cannot utilize both puzzle schemes, hence require another service that performs the puzzle computation on their behalf.

D. Related Work Summary

None of the puzzle schemes in all three categories we discussed provides solution to the resource disparity problem. Moreover, the puzzle computation interferes with the concurrently running user applications on client machines, in the form of cache displacement (memory-bound puzzles) or competing for CPU power (CPU-bound puzzles).

VII. Conclusion and Future Work

In this paper, we showed that most of the existing cryptographic puzzle schemes do not consider the resource disparity between clients. Although some proposals suggested using memory-bound puzzles, practicality of such schemes is still an open question. We argued using examples that resource disparity reduces or even removes the effectiveness of cryptographic puzzle schemes as a defense against denial of service attacks. We introduced guided tour puzzle, and showed that the guided tour puzzle achieves all desired

properties of an effective and efficient cryptographic puzzle scheme. In particular, we showed how guided tour puzzle achieves puzzle fairness, minimum interference properties, and how guided tour puzzle can achieve better defense against denial of service attacks.

As a future work, we would like to further improve guided tour puzzle in terms of the following. First, we would like to eliminate the need for the server's involvement in the puzzle generation process. Although currently puzzle construction requires only one hash operation at the server, we think this can be eliminated. Second, locations of tour guides most likely to have direct impact on the optimality of the guided tour puzzle, hence further investigation is needed to find out optimal ways to position tour guides in the network. Last but not least, more extensive evaluation of guided tour puzzle using both simulation and practical network testbed is needed to further consolidate our analysis of guided tour puzzle.

REFERENCES

[1] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail," in *CRYPTO '92*, Santa Barbara, CA, 1992, pp. 139–147.

[2] A. Juels and J. Brainard, "Client puzzles: A cryptographic countermeasure against connection depletion attacks," in *the 1999 Network and Distributed System Security Symposium (NDSS '99)*, San Diego, CA, 1999, pp. 151–165.

[3] W. Feng, E. Kaiser, and A. Luu, "The design and implementation of network puzzles," in *IEEE INFOCOM '05*, vol. 4, Miami, FL, 2005, pp. 2372–2382.

[4] T. Aura, P. Nikander, and J. Leiwo, "DoS-resistant authentication with client puzzles," in *8th International Workshop on Security Protocols*, vol. 2133, 2000, pp. 170–181.

[5] D. Dean and A. Stubblefield, "Using client puzzles to protect TLS," in *10th USENIX Security Symposium*, Washington DC, 2001, pp. 1–8.

[6] X. Wang and M. K. Reiter, "Defending against denial-of-service attacks with puzzle auctions," in *IEEE Symposium on Security and Privacy*, Washington DC, 2003, pp. 78–92.

[7] "Fast sha-1 hash core for asic," Helion Technology Limited, 2005. [Online]. Available: http://www.heliontech.com/downloads/sha1_asic_fast_helioncore.pdf

[8] B. Parno, D. Wendlandt, E. Shi, A. Perrig, B. Maggs, and Y. Hu, "Portcullis: Protecting connection setup from denial-of-capability attacks," in *ACM SIGCOMM '07*, Kyoto, Japan, 2007, pp. 289–300.

[9] S. Tritilanunt, C. Boyd, E. Foo, and J. M. González, "Toward non-parallelizable client puzzles," in *6h International Conference on Cryptology and Network Security*, vol. 4856, Singapore, 2007, pp. 247–264.

[10] *Secure Hash Standard*, National Institute of Standards and Technology (NIST) Std., 1995.

[11] B. Mulvey, "Evaluation of hash functions," 2006. [Online]. Available: http://bretm.home.comcast.net/~bretm/hash/

[12] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13(7), pp. 422–426, 1970.

[13] "Planet-lab scalable sensing service," Scalable Sensing Service. [Online]. Available: http://networking.hpl.hp.com/s-cube/

[14] "About planet lab," Planet Lab. [Online]. Available: http://www.planet-lab.org/about

[15] X. Yang, D. Wetherall, and T. Anderson, "A DoS-limiting network architecture," in *ACM SIGCOMM '05*, Philadelphia, 2005, pp. 241–252.

[16] A. Yaar, A. Perrig, and D. Song, "SIFF: A stateless Internet flow filter to mitigate DDoS flooding attacks," in *IEEE Symposium on Security and Privacy*, 2004, pp. 130–143.

[17] D. G. Andersen, "Mayday: Distributed filtering for Internet service," in *4th USENIX Symposium on Internet Technologies and Systems*, Seattle, WA, 2003.

[18] R. Thomas, B. Mark, T. Johnson, and J. Croall, "Netbouncer: Client-legitimacy-based high-performance DDoS filtering," in *3rd DARPA Information Survivability Conference and Exposition*, 2003, pp. 14–25.

[19] S. M. Bellovin, M. Leech, and T. Taylor, "ICMP traceback messages," IETF Draft, 2003.

[20] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Practical network support for IP traceback," in *ACM SIGCOMM '00*, vol. 30(4), Stockholm, Sweden, 2000, pp. 295–306.

[21] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker, "Controlling high bandwidth aggregates in the network," *ACM Computer Communication Review*, vol. 32(3), pp. 62–73, 2002.

[22] G. Price, "A general attack model on hash-based client puzzles," in *9th IMA Conference on Cryptography and Coding*, vol. 2898, Cirencester, UK, 2003, pp. 319–331.

[23] B. Waters, A. Juels, J. A. Halderman, and E. W. Felten, "New client puzzle outsourcing techniques for dos resistance," in *11th ACM CCS*, 2004, pp. 246–256.

[24] X. Wang and M. K. Reiter, "Mitigating bandwidth-exhaustion attacks using congestion puzzles," in *11th ACM Conference on Computer and Communications Security*, Washington DC, 2004, pp. 257–267.

[25] W. Feng, "The case for TCP/IP puzzles," in *ACM SIGCOMM Future Directions in Network Architecture (FDNA '03)*, Karlsruhe, Germany, 2003.

[26] R. L. Rivest, A. Shamir, and D. A. Wagner, "Time-lock puzzles and timed-release crypto," MIT, Cambridge, Massachusetts, Tech. Rep., 1996.

[27] M. Ma, "Mitigating denial of service attacks with password puzzles," in *International Conference on Information Technology: Coding and Computing (ITCC '05)*, vol. 2, Las Vegas, 2005, pp. 621–626.

[28] B. Groza and D. Petrica, "On chained cryptographic puzzles," in *3rd Romanian-Hungarian Joint Symposium on Applied Computational Intelligence (SACI '06)*, Timisoara, Romania, 2006.

[29] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. MIT Press and McGraw-Hill, 2001, ch. 35.

[30] M. J. Coster, A. Joux, B. A. Lamacchia, A. M. Odlyzko, C. Schnorr, and J. Stern, "Improved low-density subset sum algorithms," *Computational Complexity*, vol. 2(2), 1992.

[31] A. K. Lenstra, H. W. Lenstra, and L. Lovász, "Factoring polynomials with rational coefficients," *Mathematische Annalen*, vol. 261(4), pp. 515–534, 1982.

[32] M. Abadi, M. Burrows, M. Manasse, and T. Wobber, "Moderately hard, memory-bound functions," in *10th Annual Network and Distributed System Security Symposium (NDSS '03)*, San Diego, CA, 2003, pp. 25–39.

[33] C. Dwork, A. Goldberg, and M. Naor, "On memory-bound functions for fighting spam," in *CRYPTO '03*, 2003, pp. 426–444.