Puzzle Outsourcing for IP-Level DoS Resistance

Brent R. Waters
Department of Computer Science
Princeton University
bwaters@cs.princeton.edu

Ari Juels
RSA Laboratories
ajuels@rsasecurity.com

Chris Tunnell
ctunnell@cs.princeton.edu

Edward W. Felten
Department of Computer Science
Princeton University
felten@cs.princeton.edu

Abstract

We explore the use of cryptographic puzzles as a countermeasure to low-level denial-of-service (DoS) attacks, such as IP-layer flooding. In previous work, puzzles have served mainly as tools for DoS mitigation in higher protocol layers, for session-establishment protocols or for applications like e-mail.

In addition to its applicability to IP-level attacks, our approach is distinctive in two regards. First, we illustrate a way in which puzzles serve to protect public channels of communication for a server, rather than specific service requests from clients. We provide a detailed analysis of the resulting quality of service in different attack scenarios.

Second, we propose simple new techniques that permit the *outsourcing* of puzzles, meaning their distribution via a robust external service that we call a *bastion*. Many servers can rely on puzzles distributed by a single bastion. We show how a bastion, somewhat surprisingly, need not know which servers rely on it for puzzle distribution. Indeed, in one of our constructions, a bastion may consist merely of a publicly accessible random data source, rather than a server.

1 Introduction

Denial-of-service (DoS) attacks present a strong and well established threat to the Internet and e-commerce. One proposed countermeasure requires clients to commit resources to an interaction by successfully solving a computational problem known as a *client puzzle* [16, 23] before a server will in turn provide resources to the client. In this way, an attacker is unable to consume a large portion of the resources of a targeted server without commanding and investing considerable resources himself.

While the deployment of client puzzles in attack scenarios seems promising, we have found that most proposed systems of this type have a basic shortcoming. While they help protect higher-level system resources, such as buffer space for TCP connections or the computational resources for SSL session establishment, they remain vulnerable to lower-level IP-flooding attacks [23, 7, 13].

Low-layer flooding attacks have in fact been the most common variety to date on the Internet. See, e.g., [20]. The only plausible basis for defeating such attacks is a very efficient method of verifying the validity of each packet. Yet even the computation of an ordinary hash function such as SHA-1 on a per-packet basis requires too much computational effort by a server or router to be practical. Thus, conventional client-puzzle-based solutions, which require cryptographic computation for verification of each puzzle, cannot serve as a ready defense against packet-flooding and related attacks.

We make two key observations about DoS attacks and systems based on client puzzles. The first is that in typical DoS attacks an attacker commandeers a cohort of machines on the edge of network, but generally does not compromise routers in the middle of the network. Based on this observation,

we consider an attack model that assumes only limited eavesdropping by the adversary. (This assumption is explored further in Section 5.3.) Our second observation is that since puzzle distribution itself is vulnerable to DoS, any viable solution requires a robust point of puzzle distribution. We address these observations in two ways.

Previous schemes involve puzzle distribution on a per-request or per-session basis. We propose a more coarse-grained approach to puzzle distribution. In particular, we introduce the idea of protected communication *channels*. A channel is a virtual traffic partition (which may be designated simply by labelling packets with channel numbers). When at high risk of DoS attack — or in the midst of an attack — a host in our system accepts communication only via a restricted collection of channels.

To contact a host through one of these channels, a client must provide an appropriate token. A token consists of the solution to a client puzzle associated with the particular channel during a particular time interval. Thus, a client may easily attach tokens to every packet it transmits. The host can easily enumerate in advance the set of valid tokens, so the host (or, by proxy, a router) can verify tokens and filter channel traffic very efficiently. The idea, then, is that an adversary with limited computational resources can successfully attack only a limited number of channels. The remaining channels may then support normal communications from benign clients.

As explained above, puzzle-based DoS solutions provide a newly attractive DoS target, namely the point of distribution of puzzles itself. To address this problem, we propose a novel approach to client-puzzle distribution. We show how to *outsource* puzzle distribution to a independent Web Service that offers strong robustness, e.g., a highly distributed content-serving network or well-protected core server. We refer to this service as a *bastion*. A bastion may serve as a leverage point, reducing the basic robustness requirements needed to defend a server against DoS. We present three methods for outsourcing puzzle distribution, each with different requirements on the bastion and defending servers.

Channels and puzzle outsourcing limit the bandwidth over which an adversary is capable of seizing effective control. A complete solution, however, must include policies and mechanisms for translating the adversary's limitation on access to channels into a limitation on the adversary's ability to consume resources and deny them to others. Such a solution must both be fair and allow for maximum utilization of server resources. We design a policy for controlling access to network bandwidth within our channel framework that restricts the adversary's ability to mount an IP flooding attack against the defending server.

We show our methods to be both theoretically sound and implementable in practice using existing Internet protocols, with an added client-side component. (Our method maintains compatibility for unmodified clients, but their traffic does not receive the benefit of our DoS-resistance mechanisms.) We describe a first-stage prototype implementation of our system that is transparent to client and server applications.

The paper is organized as follows. In Section 2 we describe related work. We describe the design of our puzzle construction and distribution methods in Section 3. In Section 4 we give our policy for handling IP flooding attacks. In Section 5 we discuss practical issues that arise in putting out design into practice. Finally, we describe the details of our prototype implementation in Section 6 and conclude in Section 7.

2 Background and Related Work

In the data-security world, the term *puzzle* commonly refers to cryptograms that are solvable with a moderate level of effort. Most cryptographic systems rely on computational problems that are so hard as to be intractable; for example, the (apparently) hard problem of factoring products of large primes underlies the RSA cryptosystem. Puzzles may enhance system security by raising a non-trivial but surmountable barrier to acquisition of some resource. This has the effect of making the resource freely available, while thwarting efforts at unfair or malicious exploitation.

In this paper, we consider the use of puzzles as a countermeasure to DoS attack. Dwork and Naor [16] were the first to propose the use of puzzles for this purpose – in particular, for mitigating spam. Briefly stated, successful delivery of a piece of e-mail in the Dwork and Naor scheme requires that the sender attach a valid puzzle solution. A would-

be spammer therefore faces the deterrent of a large and expensive amount of computation.

As computational time costs money (directly or indirectly), their scheme may be thought of as akin to a micropayment system for postage. Back [8] independently devised and implemented a similar system known as Hash Cash. Gabber et al. describe an extension of the idea in which puzzles are used to establish relationships between corresponding users so as to permit effective isolation of spam.

The Dwork-Naor and Back systems permit precomputation of puzzles, namely the solution of puzzles at a time arbitrarily antedating the sending of the e-mail they are associated with. This achieves the goal of imposing a computational cost on the sending of spam. It is problematic, however, for defense again the common form of DoS in which an attacker seeks to disable a server by overwhelming its resources during some restricted period of time. This type of DoS attack, often referred to as a *flooding attack*, is a common real-world DoS problem. It is our main focus in this paper.

Juels and Brainard [23] addressed the problem of puzzle precomputation and thus flooding with an idea called "client puzzles "; these are puzzles based on session-specific parameters, for application to interactive protocols like TCP SYN and SSL. Aura, Nikander, and Leiwo [7] propose variants aimed specifically at DoS attacks against authentication protocols. Dean and Stubblefield [13] focus on the application of client puzzles to SSL (or TLS), and investigate the thorny deployment issues it poses. Wang and Reiter [37] also consider puzzle deployment for DoS protection in authentication, devising a system in which clients bid for resources by solving puzzles of appropriate difficulty.

In the past couple of years, researchers have proposed a few variants on basic puzzle constructions. Abadi et al. [1] describe a new puzzle construction aiming at a levelling effect among computational platforms, i.e., at permitting more equal resource allocation among fast and slow machines. The puzzles they propose rely primarily on the resource of fast-access memory, which tends to be more equally distributed among computing platforms than raw computational power. Dwork et al. [15] propose some improved constructions in follow-up work. Finally, CAPTCHAs [36] are a kind of puzzle that depend upon human work, rather than machine computation for their solution. All of these puzzle variants may be adapted to our proposal in this paper.

A scheme that we draw on directly for one of our proposed puzzle constructions here is the *time-lock puzzle* construction of Rivest, Shamir, and Wagner [30]. The goal of RSW was to create a kind of time capsule for data. In particular, they wished to construct a cryptogram that would be solvable only at a distant future date – say, in the year 2025. To do so, they proposed use of Moore's Law to estimate future computing power. They show how to craft a puzzle whose solution relies strictly on sequential computation and therefore on raw advances in computational speed, rather than on parallelization.

We do not in fact draw on the functional characteristics designed by RSW for their scheme. Instead, we draw on an incidental algebraic property. An RSW puzzle has the unusual characteristic of being derivable from an entirely random bitstring and a public key. At the same time it offers a shortcut solution to the holder of the corresponding private key. We explain our use of these features in section 3.5.

We omit discussion here of many cryptographic and other uses of puzzles apart from combatting DoS [6, 17, 19, 22, 26].

2.1 Approaches to IP-layer DoS

Puzzles, of course, represent only one approach to DoS mitigation, and have previously seen use mainly at the application or session-establishment level, rather than at lower protocol levels. A goal of our proposal is to address low-level, e.g., IP-layer, attacks.

One of the best known existing approaches to addressing IP-layer attacks is referred to as *traceback*. This involves supplementation of packet data so as to permit tracing of the origins of an attack [3, 10, 12, 31, 34]. Pushback [25] and Path Identification (Pi) [38] are related IP-level approaches to DoS. A drawback to these approaches is that they require modifications to the routing infrastructure. A benefit is that they facilitate gathering of forensic data. Anomaly detection [9, 21] is another actively researched approach to IP-level DoS that involves classification and suppression of suspicious network traffic.

A very practical approach to attacks against certain

protocols, and used in real-world systems to protect the TCP SYN protocol, is known as a *syncookie*. In order to validate the claimed IP address of a client, a server transmits a (cryptographically computed) cookie to the address. The client must transmit this cookie to the server in order to have its service request completed. Thus, while not aimed at IP-layer DoS, syncookies exploit low-level network services to achieve their protection.

An important emerging thread of research on DoS that underlies our work here involves redirection of potentially hostile traffic to robust loci capable of withstanding attack and providing filtering services, as in Stone [35], Andersen [4], and Keromytis et al. [24]. Recently Adkins, Lakshminarayanan, Perrig, and Stoica [2] show how to combine this approach with puzzles; among other ideas, they advocate leveraging the (proposed) Internet Indirection Infrastructure (i3) in such a way that a challenge puzzle is issued for each connection request. Our proposal is similar in flavor, but more lightweight and consequently coarser in nature. A key difference is that we advocate outsourcing from a defending server only of the process of puzzle distribution, rather than broad management of incoming traffic.

In this respect, our proposal is similar to that of Anderson, Roscoe, and Wetherall [5]. They propose that a client use a *token* in order to validate a path to a server; this token serves as a packet-level nonce employable for purposes of filtering by "verification points." A token in the ARW approach serves essentially the same function as a puzzle solution in our own. The security model is similar as well: Anderson et al. assume that adversaries do not eavesdrop extensively on network links. A key difference is the way in which tokens are distributed. ARW propose incremental deployment of an infrastructure of "Request-to-Send" (RTS) servers (and do not detail the critical policy question of how transmitters are authorized to obtain tokens from RTS servers). Bastions in our proposal are analogous to RTS servers. Indeed, our proposal may be viewed as a more practical alternative to RTS servers: Bastions dispose of the need both for an infrastructure of actively intercommunicating servers and for explicit policies around token distribution.

2.2 Our work

Most previous puzzle-based approaches to DoS (as well as other ideas like syncookies) have sought to defend against application-layer attacks. As such, they operate under the assumption that a defending server can dispense puzzles effectively even in the course of a DoS attack. What is assumed to be in jeopardy is the ability to provide resource-intensive services such as SSL connections.

As explained, however, most real-world DoS attacks to date have occurred at the IP layer. With this in mind, our proposal aims to provide DoS protection down to the lowest protocol layers, and in the face of attacks that obstruct all effective outbound communication. In order to accomplish this, we cannot take for granted the ability of the defending server to dispense puzzles.

It is for this reason that we instead adopt the approach of outsourcing the process of puzzle distribution to a robust external service. Viewed another way, our goal is to enable a defending server to leverage the strong robustness of a bastion. We wish to accomplish this in a lightweight manner. In our solution, the bastion only assumes responsibility for distributing puzzles, and not for performing any services or other content distribution on behalf of defending servers. This is important in rendering our solution practical and flexible. (In practice, a bastion might be furnished by a highly distributed and robust content server such as Akamai, or a core Internet service like DNS.) If the bastion timestamps and digitally signs puzzles, then puzzles may in principle be redistributed from any point on the Internet.

In contrast to most previous solutions applicable at the IP layer, like traceback, ours does not require changes to the routing infrastructure. There is a tradeoff, however. Our solution does require software deployment of some kind at the client, like most puzzle-based solutions, an issue that we explore in this paper.

3 Puzzle Construction

In this section, we give detailed descriptions of puzzle constructions geared at the type of puzzle-

outsourcing our scheme requires, along with very informal discussion of their functional and cryptographic properties.

We emphasize that lack of space forbids our including formal definitions and security proofs here; thus what is presented are construction sketches only and heuristic hardness claims. This is not to discount the importance of a formal model here. On the contrary, formal definitions for puzzle hardness [22, 15] are only incipient in the literature, and would naturally require extension to the outsourcing scenario as a prerequisite for security analysis. This is simply beyond the scope of our investigation here.

Let us introduce some notation. Let $f_k : \{0,1\}^* \rightarrow \{0,1\}^k$ be a one-way hash function whose range consists of k-bit strings. It is convenient to model f as a random oracle. The value k is a security parameter; we drop this subscript where appropriate for visual clarity. A parameter l serves to govern the hardness of the puzzle constructions we describe.

For a channel c and timeslot τ and defending server ID, let $\pi_{ID,c,\tau}$ denote a published puzzle. Let $\sigma_{ID,c,\tau}$ denote the corresponding solution (which we assume to be unique).

We let y_{ID} denote the public key associated with a particular defending server ID, while x_{ID} denotes the corresponding private key; we let y and x be the respective keys of the bastion. We omit the subscript ID where context makes it clear.

3.1 Hash-function-inversion puzzle construction

It is possible to perform outsourcing by means of partial hash-function inversion problems like those employed in previous puzzle-based approaches to DoS, e.g., [2, 23].[1] The idea, briefly stated, is as follows. Let $\sigma_{c,\tau}$ be the j-bit secret key for $j > l$. A puzzle is computed as $f(\sigma_{c,\tau})$. To calibrate the hardness of the problem so as to require 2^{l-1} hash-function computations on average, all but l bits of $\sigma_{c,\tau}$ are revealed. Thus, for instance, a puzzle might take the form $\pi_{c,\tau} = (f(\sigma_{c,\tau}), \sigma'_{c,\tau})$, where $\sigma'_{c,\tau}$ consists of all but the first l bits of $\sigma_{c,\tau}$.

[1] Note that a related inversion-based puzzle construction is employed in [18]. That construction does not in general have a unique solution for a given puzzle, and therefore cannot be used conveniently for our purposes, as explained below.

To outsource the construction of such puzzles, we simply let x_{ID} be shared between the defending server and bastion (The secret x_{ID} might be computed as a function of y and y_{ID} via D-H key agreement). Then, we let $\sigma_{ID,c,\tau} = f(c, \tau, x_{ID})$. With this approach, the defending server can quickly compute the set of solutions to puzzles for a given timeslot τ, and can do so without communicating with the bastion.

3.2 Some requirements for our scheme

The simple hash-inversion puzzle just described is a practical one with a well explored basis in the literature. Puzzle outsourcing for our purposes, however, introduces a new set of constraints and requirements.

To begin with, recall that every timeslot and channel in our solution has only one associated puzzle. Hence for any given timeslot the total number of puzzles is equal to the number of valid channels – perhaps on the order of thousands, according to the parameterizations we envision and describe below. In strict contrast to previous puzzle-based DoS systems, the defending server in our scheme can afford to invest fairly considerable computational resources in puzzle construction and solution. Even the computation of a modular exponentiation per puzzle would be acceptable. We therefore have the flexibility to introduce puzzle constructions based on public-key cryptography in our scheme.

At the same time, however, outsourcing imposes a new set of requirements on puzzle construction. We enumerate the most important of these here:

1. **Unique puzzle solutions:** The practicality of our solution depends on the ability of a defending server to precompute puzzle solutions prior to their associated timeslot, and subsequently to check their correctness via a table lookup. As a result, it is important that puzzles have unique solutions (or at least a fairly small number of correct ones).
2. **Per-channel puzzle distribution:** It is desirable for the bastion to be able to compute and disseminate puzzle information on a per-channel basis. In other words, the bastion should be able to publish information for a particular channel number c that may be used to

deduce the corresponding puzzle for any defending server. (Different servers should have different puzzle *solutions*, though, so that one server's ability to enumerate its own puzzle solutions does not open other servers up to attack.)

With this property, the bastion need not even know which defending servers are relying on its services. This reduces the amount of information the bastion must compute and publish, as well as the need for explicit relationships or coordination between defending servers and bastions.

3. **Per-channel puzzle solution:** Another desirable property that is for the work done by a client to apply on a per-channel basis, rather than a per-puzzle basis. In particular, we would like a client that has solved a puzzle for a particular channel to be easily able to compute the token for the same channel number on *any* server.

   As noted above, this does not mean that tokens should be identical across servers – only that there should be considerable overlap in the brute-force computation need to solve the puzzle for a given channel-number across servers. In particular, it is not desirable for one server to be able to use its shortcut to compute the tokens associated with another server, as this would result in a diffusion of trust across all participating servers, rather than in the bastion alone.

   The per-channel puzzle solution property is useful because it allows a client to begin solving puzzles before the user decides which server will be visited.

4. **Random-beacon property:** It is possible to achieve a property even stronger than per-channel puzzle distribution. Ideally, puzzles would not require explicit calculation and publication by a particular bastion. They might instead be derived from the emissions of a *random beacon.*

   We use the term random beacon to refer to a data source that is: (1) Unpredictable, i.e., dependent on a fresh source of randomness; (2) Highly robust, i.e., not subject to manipulation or disruption; and (3) Easily accessible on the Internet. A puzzle construction based on a random beacon would eliminate the need for an explicit bastion service. (Apart from the architectural advantages, this could have the benefit in some circumstances of eliminating any point of legal liability for reliable puzzle distribution.) Hashes of financial-market data, which are broadcast from multiple sources, or even of highly robust Internet news sources would be candidate random beacons.

   Note that not only would the bastion (random beacon) here not have to know what defending servers are relying on its services, it wouldn't even have to know that its data are being used to construct puzzles!

5. **Identity-based key distribution:** When puzzles are based on the public key of a defending server, the public key itself must be distributed via a robust directory. A desirable alternative is identity-based distribution, i.e., the ability to derive the public-key of a particular defending server from the server name and a master key that is common to all defending servers. This is very closely analogous to the well-known primitive of identity-based encryption [11].

6. **Forward security:** A final desirable property is that of forward security. By this we mean that time-limited passive compromise of a bastion should not undermine the DoS protection it confers.

3.3 A Diffie-Hellman-based construction

We now describe a puzzle construction based on Diffie-Hellman key agreement [14]. It possesses all of the properties above except the random-beacon property, i.e., it has properties 1,2,3,5 and 6.

Let G be a group of (prime) order q. Let g be a published generator for the group, and l be a parameter denoting the hardness of puzzles for this construction. (As we explain below, we require a strong, generic-group assumption on G.)

We propose a simple solution in which the bastion selects a random integer $r_{c,\tau} \in_R Z_q$ and a second, random integer $a_{c,\tau} \in_R [r_{c,\tau}, (r_{c,\tau} + l) \bmod q]$. (Recall that l is the hardness parameter for the puzzle.) Let f' in this case be a one-way permutation on Z_q, and let $g_{c,\tau} = g^{f'(a_{c,\tau})}$.

The intuition here is as follows. The value $g_{c,\tau}$ may be viewed as an ephemeral Diffie-Hellman public

key. A puzzle solution for defending server ID is the D-H key that derives from its public key y_{ID} and the ephemeral key $g_{c,\tau}$. Solving a puzzle means solving the associated D-H problem. So as to render the problem tractable via brute force, the bastion specifies a small range $[r_{c,\tau}, (r_{c,\tau} + l) \bmod q]$ of seed values from which its ephemeral key is derived. In other words, the bastion publishes publishes $\pi_{c,\tau} = (g_{c,\tau}, r_{c,\tau})$.

For a client (or attacker) to solve the puzzle requires brute-force testing of all of the seed values. In particular, for a given candidate value a', the client tests whether $g_{c,\tau} = g^{f'(a')}$. For a particular defending server ID, the solution to the puzzle is $\sigma_{ID} = {y_{ID}}^{f'(a_{c,\tau})}$.

A defending server, of course, can use its private key x_{ID} as a shortcut to the solution of the puzzle. The defending server can compute $\sigma_{ID} = {y_{ID}}^{f'(a_{c,\tau})} = {g_{c,\tau}}^{x_{ID}}$. In other words, it essentially computes a Diffie-Hellman key. Thus, for a defending server, solution of a puzzle requires essentially just one modular exponentiation.

On average, solution of a puzzle by a client (or attacker) requires $l/2$ modular exponentiations over G.

Because of the need for very precise characterization of puzzle hardness, we believe that any concrete computational hardness claim would have to depend, in addition to a random-oracle assumption on f', on a generic-model assumption for the underlying group G [32]. It is therefore important to choose G appropriately. (Several common types of algebraic groups are presently believed to have the ideal properties associated with the generic model, e.g., most elliptic curves and the order-q subgroup G of the multiplicative group Z_p^*, where $p = kq + 1$ for small k [32].)

Remark on application of f': Applying f' in the computation of ephemeral key $g_{c,\tau} = g^{f'(a_{c,\tau})}$ is a requirement to break algebraic structure among seed-to-key mappings. If we chose $g_{c,\tau} = g^{a_{c,\tau}}$, for example, then it would be possible to cycle through candidate seed values by computing $g^{r_{c,\tau}}$ and repeatedly multiplying by g.

3.4 Identity-based distribution of public keys

We very briefly and very informally sketch here a technique for distribution of the public keys $\{y_{ID}\}$ of defending servers in an identity-based manner. In other words, we show how y_{ID} may derive from a string representing the identity ID (e.g., a domain name) and a master key.

Employing the notation of [11] (with which we must assume familiarity here for the sake of brevity), let where G and G' are two groups of large prime order q. Let $\hat{e} : G \times G \rightarrow G'$ be an *admissible* bilinear mapping in the sense defined by Boneh and Franklin. For G suitably chosen as a subgroup of the additive group of points of an elliptic curve E/F_p for prime p, $\hat{e}$ may be constructed using the Weil pairing. Recall that when the system is correctly parameterized, it is believed that the Bilinear (Computational) Diffie-Hellman (BCDH) Assumption holds, an essential hardness property for our proposal here. Roughly stated, given $P \in_R G$ and points aP, bP, and cP for $a, b, c \in_R Z_q$, it is hard to compute $\hat{e}(P, P)^{abc}$.

Let x' be the private key of the trusted dealer, and let $y' = x'g$ be an associated public key. Finally, let $d : \{0,1\}^* \rightarrow G$ be an appropriate one-way function mapping identifier strings to group elements in G.

In this scheme, the public key of defending server with identifying string ID is computed simply as $y_{ID} = d(ID)$. The associated private key, computable by the trusted dealer, is $x'y_{ID}$.

As before, we let $a_{c,\tau} \in_R [r_{c,\tau}, (r_{c,\tau} + l) \bmod q]$. The ephemeral key computed by the bastion assumes the form $g_{c,\tau} = f'(a_{c,\tau})g$. The bastion publishes $\pi_{c,\tau} = (g_{c,\tau}, r_{c,\tau})$, just as it does in the D-H puzzle.

The difference for the identity-based variant lies in the form of the puzzle solution. This is defined here to be $\sigma_{ID,c,\tau} = \hat{e}(y_{ID}, g)^{x'f'(a_{c,\tau})}$. (This solution may be hashed for compactness.) After solving for $a_{c,\tau}$, a client may compute this as $\hat{e}(y_{ID}, y')^{f'(a_{c,\tau})}$. The defending server may use its knowledge of x_{ID} as a shortcut. In particular, $\sigma_{c,\tau}^{(ID)} = \hat{e}(x_{ID}, g_{c,\tau}) = \hat{e}(y_{ID}, g)^{x'f'(a_{c,\tau})}$.

By analogy with our D-H construction, the work for brute-force solution here is on average $l/2$ multiplications over the elliptic-curve based group G.

3.5 Puzzle construction using time-lock puzzles

We now propose a puzzle construction that has properties 1,2,4,6 above. It achieves the random-beacon property. It has the disadvantage, though, of lacking an identity-based variant and the per-channel puzzle solution property. Thus, it requires explicit distribution of public keys for defending servers and the client cannot start solving puzzles for a server until it knows which one to target.

This construction is a simple adaptation of the time-lock puzzle scheme proposed by Rivest, Shamir, and Wagner [30]. A public key y_{ID} consists of an n-bit RSA modulus N_{ID}. (See [30] for discussion of restrictions on the choice of N.)

In the original RSW construction, a random value $a \in_R Z_n$ serves as a basis for the puzzle. The prescribed task for solution of the puzzle is the computation of a secret value $b = a^{2^l} \bmod n$ for some appropriate l. The value b serves essentially as a key to the puzzle. The parameter l governs the hardness of the puzzle; in particular, a solver must perform l modular squarings in order to compute b and "unlock" the puzzle.

Knowledge of the factorization of n, however, provides a shortcut in the computation of the secret b. In particular, when l is sufficiently large, it is considerably more efficient to compute $e = 2^l \bmod \phi(n)$ and then $a^e \bmod n$.

As explained above, the original RSW construction aims at creating a kind of digital time-capsule, that is, a cryptogram solvable only in the distant future thanks to advances in computing power. RSW propose that the puzzle constructor determine how hard the puzzle should be, use the shortcut in order to create an encryption key associated with the puzzle, and then erase all data associated with the shortcut, thereby sealing the time-capsule.

The main goal in the RSW design was to render the solution process difficult to parallelize, so that the ability to unlock the puzzle would truly depend upon raw advances in computing power. This property is achieved thanks to the sequential nature of the modular squarings required for the solution. (A puzzle based on hash-function inversion, for example, would not achieve this goal, as it could be divided among many different computing devices.)

We exploit an altogether different property of the RSW construction (one probably not explicitly designed by its inventors). We observe that a time-lock puzzle may be derived very simply from a random string (used to derive a) and an RSA modulus. In particular, no explicit computation by the bastion is required to create a valid time-lock puzzle. This is very different from the case, for instance, with our D-H solution above, which requires computation of an ephemeral D-H key, or from a hash-function-inversion puzzle, which requires the hashing of a secret value.

Given this observation, the puzzle construction is quite simple. Let r_τ be a suitably long random string emitted by a random beacon in timestep τ (say, $n + k$ bits in length for security parameter $k \approx 128$). We let $r_{ID,c,\tau} = f_{n+k}(ID, c, \tau, r_\tau)$. We then compute $\pi_{ID,c,\tau} = a_{c,\tau} = r_{ID,c,\tau} \bmod N_{ID}$.

The solution $\sigma_{ID,c,\tau}$ to this puzzle is simply $(a_{c,\tau})^{2^l} \bmod N_{ID}$. A client (or attacker) must compute this by repeated squarings. It may be computed by the defending server in the obvious way using its shortcut.

Note that from a single random value, puzzles may be computed for an arbitrarily large number of channels. The security parameter l may be set by a defending server as desired. For the defending server, the work to solve a puzzle only requires a modular reduction (whose size is parameterized by l) and an RSA exponentiation. For a client (or attacker), solving the puzzle requires l modular squarings.

4 Policy

Now that we have mechanisms in place to create and solve puzzles, and to verify puzzle solutions efficiently, we must devise a policy for deciding how to use puzzles and their solutions to regulate access to network servers. In this section we construct a policy for regulating bandwidth to the server. However, the channel mechanisms we create can also be used to create policies to protect higher layer protocols in the server.

4.1 Model

We begin by presenting a model for workloads and policies. We will associate one channel with each valid puzzle, and we say a packet is "on" a channel when the packet is tagged with the solution to the corresponding puzzle.

We represent a channel c as a pair (s_c, b_c), where s_c is the solution to the channel's puzzle, and $b_c \geq 0$ is the rate of arrival (at the server) of packets on the channel. A *workload* is a pair (C, M), where C is a set of channels, and M is the maximum bandwidth that the server can handle.

A *policy* is a function that decides how many packets per second to accept on each channel. In other words, given a workload $W = (C, M)$ and a channel $c \in C$, the policy P allows $P(W, b_c)$ packets per second to reach the server on channel c. For brevity, we will sometimes write (P, c) to mean (P, b_c); for a set of channels C, we will sometimes write $P(W, C)$ to mean $\sum_{c \in C} P(W, c)$.

Definition 1 *A policy P is* **feasible** *if, for all workloads $W = (C, M)$, $P(W, C) \leq M$ and for all $c \in C$, $0 \leq P(W, b_c) \leq b_c$.*

A feasible policy is one that blocks some (possibly empty) subset of the packets on each channel, and blocks enough packets that the total number of accepted packets does not exceed the server's capacity M.

Definition 2 *A policy P is* **fair** *if P is feasible and there is a function $T_P(W)$ such that for all workloads W and constants b, $T_P(W) > 0$ and $P(W, b) = \min(b, T_P(W))$.*

Intuitively, this definition imposes a "cap" on the bandwidth that can be used by any single channel. Channels that do not exceed the cap are left alone, but channels that try to exceed the cap have their traffic reduced so that they stay within the cap.

Definition 3 *A policy P is* **maximal** *if P is fair, and if, for all fair policies Q, all workloads $W = (C, M)$, and all channels $c \in C$, $P(W, c) \geq Q(W, c)$.*

A policy is maximal if it gives every channel the maximum possible bandwidth, consistent with fairness. Lemma 5 in Appendix A shows that a maximal policy exists.

4.2 An Example

A simple example will help to justify our definitions. We consider two sets of channels: the channels in L are used by legitimate clients, and those in A are used by an adversary. We assume the legitimate clients' channels all desire the same amount of bandwidth, b_L. The adversary can send as many packets as he likes. How much of the legitimate clients' traffic can the adversary displace?

Theorem 1 *Let b_L be a constant, and let $W = (L, M)$ and $W' = (L \cup A, M)$ be workloads, such that for all $c \in C$, $b_c = b_L$. Let P be a fair, maximal policy. Then*

$$P(W', L) \geq \frac{|L|}{|L| + |A|} P(W, L).$$

(A proof appears in Appendix A.) This theorem tells us that under the assumed conditions, if the adversary wants to take a fraction f of the available bandwidth away from the legitimate clients, he has to do at least a fraction f of the total puzzle-solving work. For example, to displace half of the legitimate clients' traffic, the adversary has to do at least as much puzzle-solving work as all of the legitimate clients put together.

One desirable consequence of this result is that the more popular a site is, the harder it is to attack. More popular sites have a larger number of legitimate clients, so the adversary has to do more puzzle-solving work to displace the same fraction of traffic. These results hold regardless of how many packets the adversary tries to send.

4.3 Randomly Chosen Puzzles

In practice, clients are not divided evenly among channels. Instead, each client will choose a channel at random. In particular, we assume a total of n channels. We assume there are k clients, and each

client chooses one of the channels at random and solves that channel's puzzle. The attacker chooses a channels at random, and solves those channels' puzzles. Those channels are assumed to be totally controlled by the attacker, so that legitimate clients who happen to choose a channel that the attacker chose will get no bandwidth at all. To be specific, we define a randomized function $R_\beta(n,k,a)$, which returns a set of $n-a$ channels, produced by the following procedure:

1. Create n channels, each with offered bandwidth of zero.
2. Divide k clients randomly among the channels, where each client makes an independent, uniform random choice among the n channels. Whenever a client chooses a channel, increase that channel's offered bandwidth by a constant β.
3. Now choose a of the channels (randomly and uniformly), and remove the chosen channels.
4. Return a set containing the remaining channels.

Given these assumptions, the number of clients in any given channel is binomially distributed, according to a distribution function [2] f:

$$f_{n,k}(i) = \binom{k}{i}\left(\frac{1}{n}\right)^i\left(1-\frac{1}{n}\right)^{k-i}.$$

We will analyze the impact of these random distributions by proving a general theorem, and then analyzing two special cases, one where the number of clients is small, and another where it is very large.

We first present the general theorem, whose proof appears in Appendix A.

Theorem 2 *Let P be a maximal policy. Let b, ρ, n, and a be constants. Let $W=(L,M)$ be a workload, chosen by some randomized process such that $|L| = n-a$ and for all $c \in L$, $\mathrm{Prob}(b_c \geq b) \geq \rho$. Let $W' = (L \cup A, M)$ be another workload such that $|A| = a$. Assume $nb \leq M$. Then $E[P(W',L)] \geq \frac{\rho^2(n-a)^2}{a+\rho(n-a)}b$.*

[2] The distributions for various channels are not independent, but this will not be a problem, because our proofs will not require independence.

4.3.1 Small Number of Clients

Next, we will analyze the case where there are relatively few clients. In this case, most clients will have a channel to themselves, but some collisions will occur. The following theorem tells us how much bandwidth the few legitimate clients retain when the system is attacked.

Theorem 3 *Let P be a maximal policy, and let $W=(L,M)$ be a workload, where L is generated according to $R_\beta(n,k,a)$. Let $W' = (L \cup A, M)$ be another workload, where $|A| = a$. Assume $k < n$ and $n\beta \leq M$. Then*

$$E[P(W',L)] \geq \frac{k^2\beta\left(1-\frac{a}{n}\right)^2}{a+k\left(1-\frac{a}{n}\right)}\left(1+O\left(\frac{k}{n}\right)\right).$$

(A proof appears in Appendix A.) When there are very few clients, many channels will be empty, and the nonempty channels will mostly contain a single client. The adversary can displace legitimate bandwidth in one of two ways. First, he can capture a channel that is being used by a legitimate client; a given channel is captured with probability $\frac{a}{n}$. Second, having captured a set of channels, the adversary can send many packets on those channels, thereby causing the policy to impose an equal cap on all channels, in the hope that the cap will impact the channels being used by legitimate clients. Nevertheless, an adversary who wants to displace a large fraction of the clients must control a significant number of channels, and therefore must solve a significant number of puzzles.

4.3.2 Large Number of Clients

Theorem 4 *Let P be a maximal policy, and let $W=(L,M)$ be a workload, where L is generated according to $R_\beta(n,k,a)$. Let $W' = (L \cup A, M)$ be another workload, where $|A| = a$. Assume $\beta = \frac{M}{k}$ and $k > n$. Then*

$$E[P(W',L)] \geq \left(1-\left(\frac{n}{k}\right)^{\frac{1}{3}}\right)^3\left(1-\frac{a}{n}\right)^2 M.$$

(A proof appears in Appendix A.) This theorem gives a lower bound on the legitimate clients' bandwidth, in the case where there are more legitimate

clients than channels. When there are far more clients than channels, the first factor on the right-hand side approaches 1. This factor represents the effect of the number of clients being imbalanced between channels, and the effect of this random imbalance becomes smaller as the number of clients per channel grows. The theorem shows that the adversary must do a very substantial amount of work to displace a large amount of legitimate traffic.

5 Practical Considerations

Putting our design into practice requires us to deal with several practical issues.

5.1 Approximating a maximal policy

Our goal in implementing a policy is to approximate, as nearly as possible, a fair, maximal policy for allocating bandwidth between channels. We do this by adaptively computing a threshold T, and then capping the bandwidth of each channel c at T.

Bandwidth caps are implemented using the standard leaky bucket algorithm [28], with the time constant (that is, the time required for a full bucket to drain) set to one second.

We choose T adaptively. We divide time into one-second intervals, and in each interval we measure N_{allow}, the number of arriving packets that are allowed by the current policy, and N_{deny}, the number of arriving packets that were disallowed by the current policy. We also use N_{max}, which is the maximum number of packets per second that the server can handle.

If $N_{allow} > N_{max}$, then the policy allowed more packets than the server can handle, so we conclude that T is too large, and we multiply T by 0.75.

If $N_{allow} < N_{max}$ and $N_{deny} > 0$, then the policy rejected some packets that it could have accepted safely, so we conclude that T is too small, and we multiply T by 1.05.

During times when the server has adequate bandwidth, our policy will increase T until the server is accepting all of the offered traffic. When the offered traffic exceeds the server's capacity, T will converge to a value that is about right to keep the traffic within the server's capacity, while maintaining fairness between channels.

Clients are not notified directly of the current policy. However, if a client using TCP exceeds its allocation, the resulting dropped packets will be interpreted by the client's TCP stack as congestion, causing the client to slow down. TCP's congestion control algorithm will tend to adjust the client's sending rate to match its allocation.

5.2 Backward compatibility

Our method asks clients to attach a token to every packet they send. Since we can't ask every client to adopt our method simultaneously, we must be able to cope with clients that don't use our method. To do this, we create an imaginary channel, and assign all of the non-compliant traffic to it. We can treat this as the equivalent of many channels if we like, so that the artificial channel gets a bandwidth cap of, say, $20T$.

Of course, an attacker can easily flood this channel with traffic without having to solve any puzzles, so the attack-resistance benefits of our method will apply only to compliant clients.

The backward compatibility mode will also serve clients that have just booted up and have not yet had time to solve any puzzles. Until it solves its first puzzle, a client can simply send packets without tokens, and that traffic will be accepted via the backward compatibility mechanism.

5.3 Eavesdropping attacks

As stated in the introduction we use an attack model where we assume that eavesdropping on the Internet is difficult for typical DoS attackers. However, it is still useful to consider what happens if eavesdropping occurs, in what situations it might occur, and measures that can be used by a client to prevent being eavesdropped upon.

If an attacker is able to eavesdrop on packets sent by a client to the server under attack then he essentially is able to convert the client into a drone

that will solve puzzles for him. This will have two repercussions. First, the attacker will be able to get another channel and consume more resources on the system as a whole. Second, the attacker will occupy the same channel as the client from which he stole the token from and that client will be likely shut out of that channel. Therefore, there is a special incentive for a client not to have his own particular tokens eavesdropped upon.

If core routers on the Internet are difficult to compromise, then the most likely source for eavesdropping attacks are on the edge of the Internet such as a local LAN. If a client suspects that his packets are being eavesdropped upon, then it could send them securely to some part of the net that it believes to be uncorrupted. One way of doing this is to tunnel packets through IPSec [33]. We do not recommend for the server itself (or a nearby router) to act as an endpoint for such a tunnel, as the IPSec protocol could become a DoS vulnerability itself.

5.4 Number of channels

An important resource in our system is the number of channels. If there are too few channels a well equipped adversary can deny service to many users by a colliding with them on the same channel. We want to be able to resist an attacker who controls a few thousand zombie machines.

There are two limitations on the number of channels. The first is on the amount of memory at the edge router protecting the service. The token and bandwidth control data will take about 20 bytes each, therefore an edge router with 100MB of memory has enough space for about 5 million channels.

The second limitation on the number of channels comes from the ability of the server to calculate all the solutions to puzzles (using its shortcut). At the beginning of each time cycle the bastion will publish the puzzles using one of the methods of Section 3. During this time cycle a server machine (using the shortcut) will solve for the tokens of every channel it maintains and the clients will solve for a token. These tokens will then be valid on the next cycle.

To be concrete we examine the Diffie-Hellman puzzle method of Section 3.3. If the server has one machine dedicated to solving tokens for its puzzles then it will be able to solve about 200 puzzles per second if we take the time for an exponentiation operation to be 5ms. Suppose that each legitimate client has the same computing power as an adversary and that we want each legitimate client to be guaranteed to solve one puzzle in the time cycle. Then we need to set the average difficulty of the puzzle to be half of a time cycle and a 5,000 machine strong adversary will solve approximately 10,000 puzzles each cycle. If we want the adversary to be able to displace at most 5 percent of the clients by collision that means we must have 200,000 channels. The server will then need a time period of a little under 17 minutes for one machine to solve all the tokens for all of these channels.

A seventeen-minute cycle will force each client to operate in backward-compatibility mode for the first eight minutes it is up, on average. In this mode it may find itself unable at first to access any hosts that are under attack. However, once it has solved its first puzzle, it can ensure that every subsequent puzzle is solved before the solution is needed.

Clients benefit when they use a puzzle construction that allows a single puzzle-solving step to be used to access any server, as with our Diffie-Hellman and IBE constructions. Because of this property, a client can solve puzzles in advance, without needing to know in advance which servers it will be interacting with. Of course, this same property applies to attackers too, but it will do little good as attackers rarely need to decide on the spur of the moment which target to attack. (Because puzzles last only for one or two time intervals, the attacker cannot stockpile puzzle solutions.) Admittedly, the adversary will be able to use its work to launch concurrent denial-of-service attacks. However, any one server is still protected and an adversary's ability to launch a denial-of-service attack on the several sites at once will be limited by other factors.

Using different variants will alter the parameters of our system. A server machine using an identity-based distribution will require a time cycle of about 10 times longer (or more machines) due to the computation required to calculate the bilinear map.

The hash-function-inversion method a machine working for the server can calculate the solutions (with its shortcut) quickly enough so that the time-cycle will be determined by other factors such as how frequently the bastion can change without causing synchronization problems.

Finally, a client solving puzzles of the time-lock form will not be able to use his work across servers so a user will experience a time-lag every time he moves to a new site under DoS attack.

We point out for comparison that other work using the client puzzle paradigm considers adversaries consisting of a much smaller number of zombie machines [23]. The user latency observed in other systems with an attacker of 5,000 zombie machines will be much larger than that given their analysis.

5.5 A user's view

Now that we have explored likely system parameters for our scheme we are able to describe our system from the perspective of a user. Suppose we are using the D-H variant of our scheme and a user is browsing the web and none of the sites visited so far have been under attack. During this time the user's machine has been solving puzzles for channels during its idle cycles as a precaution against encountering servers that are under attack. Suppose now that the user switches to a site that is under attack. (When poor performance is observed the client machine can guess that a site is under attack and attach tokens. If this guess is incorrect the tokens will just be ignored.) The user's machine will already have solved a puzzle that can lead to a channel for that site and the user can immediately use that solution. This proactive approach to solving puzzles leads to minimal latency observed by the user.

If the time-lock variant is used a client's machine will not know which puzzles to solve until a user indicates she is interested in a site that is under attack. In this case the experienced user latency could be around the time cycle period for publishing puzzles. We believe that 30 seconds might be an appropriate value. If only one server machine is dedicated to solving puzzles then an attacker of approximately 150 machines can cause a 5 percent collision rate. Once a user indicates that she is interested in a site, she will need to wait for his machine to solve the first puzzle. However, the user's machine will continue to solve puzzles until the user is finished and there will only be one upfront delay experienced by the user.

6 Implementation

We now describe our prototype implementation of our system. Our puzzle construction is one based on the hash-function inversion technique described in Section 3.1. We use SHA-1 as our one-way hash function.

The bastion is currently implemented as a simple content server from which a client will download and solve a puzzle from. (Currently, the puzzle solver is manually invoked by the user.)

The solution to a puzzle is a 64-bit token that will be attached as an IP option [29] onto each outgoing packet to the server under attack. We use the Iptables [27] tool of the Netfilter framework to implement this. (See Figure 1.) Suppose the server under attack is at IP address 168.200.55.123. Then the client will execute the command `iptables -A OUTPUT -d 168.200.55.123/32 -j QUEUE`. This command will cause all packets destined for 168.200.55.123 to be sent to a userspace program. The user space program is able to interface with the queue using Netfilter's libipq API. This program will receive the current token solutions from the client puzzle solver and append these to each caught packet before sending it back to the routing layer.

On the server side of our implementation a router sits between the server and the network. We again use the Netfilter framework to implement our mechanism. A userspace program will receive all packets destined for the server and check to make sure a valid token exists in the IP option field. Packets which have a valid token will be accepted and those that do not will be dropped. Valid tokens are communicated by a server side program. Additionally, the token is left on as an IP option so that the server has the ability to use it to protect other layers of the protocol stack.

We tested our system over the network and found that users were indeed able to access the server using our prototype implementation. The primary goal of the prototype was to test the feasibility of our solution using available technology. We note that our solution does actually require any changes to the network nor any kernel modifications to Linux assuming the netfilter toolkit is installed. In practice we expect that a more efficient mechanism than using libipq will be used at least for the server side.

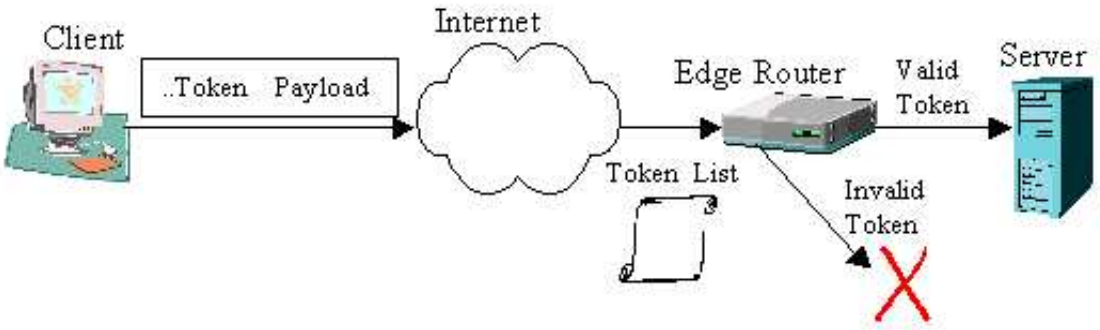

Figure 1: The client attaches a the solved token onto the header of each packet as an IP option. The edge router before the server will check each packet for a valid token on its token list. If one exists the packet is passed on to the server otherwise it is dropped.

Additionally, the policy mechanisms described in Section 4 will be applied in any industrial implementation.

We are currently implementing the D-H variant of puzzle distribution and this will be completed in time for the final paper deadline. In the final paper, we intend to include a URL from which readers can download the source code for our prototype.

7 Conclusion

We have examined the problem of defending a server against IP-level denial-of-service attacks using client puzzles. We observe that since puzzle distribution itself can be subject to attack, any viable system must have a robust method of puzzle distribution. We developed a new model for puzzle distribution using a robust service that we call a bastion. The bastion distributes puzzles and solutions to the puzzles allow clients access to communication channels. Within this model we develop different cryptographic techniques for puzzle dispersement. Additionally, we develop a policy mechanism for allocating bandwidth amongst channels. Finally, we implemented a prototype of our system that works on today's Internet with minimal additions to standard clients.

References

[1] M. Abadi, M. Burrows, M. Manasse, and T. Wobber. Moderately hard, memory-bound functions. In *NDSS '03*, pages 107–121. Internet Society, 2003.

[2] D. Adkins, K. Lakshminarayanan, A. Perrig, and I. Stoica. Taming IP packet flooding attacks. In *HotNets-II*. ACM Press, 2003. To appear.

[3] M. Adler. Tradeoffs in probabilistic packet marking for IP traceback. In *STOC '02*, pages 407–418. ACM Press, 2002.

[4] D. G. Andersen. Mayday: Distributed filtering for Internet services. In *USENIX Symposium on Internet Technologies and Systems (USITS)*, 2003. To appear.

[5] T. Anderson, T. Roscoe, and D. Wetherall. Preventing Internet denial-of-service with capabilities. In *HotNets-II*. ACM Press, 2003. To appear.

[6] S. Ar and J. Cai. Reliable benchmarks using numerical instability. In *Proceedings of the 1993 ACM Symposium on Discrete Algorithms (SODA)*, pages 34–43, 1993.

[7] T. Aura, P. Nikander, and J. Leiwo. DoS-resistant authentication with client puzzles. In *8th International Workshop on Security Protocols*, pages 170–181. Springer-Verlag, 2000. LNCS no. 2133.

[8] A. Back. Hashcash - a denial-of-service countermeasure, 2002. Original system developed in 1997. Manuscript. Referenced 2004 at http://www.hashcash.org/hashcash.pdf.

[9] P. Barford, J. Kline, D. Plonka, and A. Ron. A signal analysis of network traffic anomalies. In *Internet Measurement Workshop*, 2002.

[10] S. Bellovin, M. Leech, and T. Taylor. ICMP traceback messages, 2003. Internet Draft.

[11] D. Boneh and M. Franklin. Identity based encryption from the Weil pairing. *SIAM J. of Computing*, 32(3):586–615, 2003.

[12] D. Dean, M. Franklin, and A. Stubblefield. An algebraic approach to IP traceback. *Information and System Security*, 5(2):99–137, 2002.

[13] D. Dean and A. Stubblefield. Using client puzzles to protect TLS. In *10th USENIX Security Symposium*, pages 1–8, 2001.

[14] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976.

[15] C. Dwork, A. Goldberg, and M. Naor. On memory-bound functions for fighting spam. In D. Boneh, editor, *CRYPTO '03*, pages 426–444. Springer-Verlag, 2003. LNCS no. 2729.

[16] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In Ernest F. Brickell, editor, *CRYPTO '92*, pages 139–147. Springer-Verlag, 1992. LNCS no. 740.

[17] M.K. Franklin and D. Malkhi. Auditable metering with lightweight security. In R. Hirschfeld, editor, *Financial Cryptography '97*, pages 151–160. Springer-Verlag, 1997. LNCS no. 1318.

[18] E. Gabber, M. Jakobsson, Y. Matias, and A. Mayer. Curbing junk e-mail via secure classification. In R. Hirschfeld, editor, *Financial Cryptography '98*. Springer-Verlag, 1998.

[19] D. Goldschlag and S. Stubblebine. Publicly verifiable lotteries: Applications of delaying functions. In R. Hirschfeld, editor, *Financial Cryptography '98*. Springer-Verlag, 1998.

[20] A. Harrison. The denial-of-service aftermath. *CNN.com*, 14 February 2000.

[21] A. Hussain, J. Heidemann, and C. Papdopolous. A framework for classifying denial-of-service attacks. In *ACM SIGCOMM*, 2003.

[22] M. Jakobsson and A. Juels. Proofs of work and bread pudding protocols. In *Communications and Multimedia Security*, pages 258–272. Kluwer Academic, 1999.

[23] A. Juels and J. Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *Proceedings of the 1999 ISOC Network and Distributed System Security Symposium*, pages 151–165, 1999.

[24] A. D. Keromytis, V. Misra, and D. Rubenstein. SOS: Secure overlay services. In *ACM SIGCOMM*, pages 61–72. ACM Press, 2002.

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

[26] R. Merkle. Secure communications over insecure channels. *Communications of the ACM*, 21(8):294–299, April 1978.

[27] The Netfilter/Iptables Project. Web site at http://www.netfilter.org.

[28] Larry L. Peterson and Bruce S. Davie. *Computer Networks: A Systems Approach*. Morgan Kaufmann Publishers, San Fransico, CA, first edition, 1996.

[29] J. Postel. RFC 791: Internet Protocol, September 1981.

[30] R.L. Rivest, A. Shamir, and D. Wagner. Timelock puzzles and timed-release crypto. Technical Report MIT/LCS/TR-684, MIT, 1996.

[31] S Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical network support for IP traceback. In *ACM SIGCOMM 2000*, pages 295–306, 2000.

[32] C.-P. Schnorr and M. Jakobsson. Security of discrete log cryptosystems in the random oracle and generic model. In *The Mathematics of Public-Key Cryptography*. The Fields Institute, 1999.

[33] IP Security Protocol Charter. Web site at http://www.ietf.org/html.charters/ipsec-charter.html.

[34] D. X. Song and A. Perrig. Advanced and authenticated marking schemes for IP traceback. In *IEEE INFOCOM*, pages 878–886, 2001.

[35] R. Stone. CenterTrack: An IP overlay network for tracking DoS floods. In *USENIX Security '00*, 2000.

[36] L. von Ahn, M. Blum, N.J. Hopper, and J. Langford. CAPTCHA: Using hard AI problems for security. In E. Biham, editor, *Eurocrypt '03*, pages 294–311. Springer-Verlag, 2003. LNCS no. 2656.

[37] X. Wang and M. K. Reiter. Defending against denial-of-service attacks with puzzle auctions. In *IEEE Symposium on Security and Privacy*, pages 78–92, 2003.

[38] A. Yaar, A. Perrig, and D. Song. Pi: A path identification mechanism to defend against DdoS attacks. In *IEEE Symposium on Security and Privacy*, pages 93–109, 2003.

Appendix A: Proofs

This appendix contains proofs for the theorems that appear in the main text of the paper. Each "Theorem" in this appendix is duplicated from the main text, with the proof added here. Anything labelled "Lemma" appears only in this appendix.

For convenience, we repeat here the definitions from the main paper.

Definition 1 *A policy P is* **feasible** *if, for all workloads $W = (C, M)$, $P(W, C) \leq M$ and for all $c \in C$, $0 \leq P(W, b_c) \leq b_c$.*

Definition 2 *A policy P is* **fair** *if P is feasible and there is a function $T_P(W)$ such that for all workloads W and all constants b, $T_P(W) > 0$ and $P(W, b) = \min(b, T_P(W))$.*

Definition 3 *A policy P is* **maximal** *if P is fair, and if, for all fair policies Q, all workloads $W = (C, M)$, and all channels $c \in C$, $P(W, c) \geq Q(W, c)$.*

Lemma 1 *If P and Q are fair policies, $W = (C, M)$ is a workload, $\gamma \in C$, and $P(W, \gamma) > Q(W, \gamma)$, then for all $c \in C$, $P(W, c) \geq Q(W, c)$.*

Proof: Since P and Q are fair, they have threshold functions T_P and T_Q respectively. By assumption,

$$\min(b_\gamma, T_P(W)) > \min(b_\gamma, T_Q(W)).$$

Since the left-hand side is at most b_γ, the right-hand side must be less than b_γ, so it follows that the right-hand side is not equal to b_γ. Thus we have

$$\min(b_\gamma, T_P(W)) > T_Q(W).$$

The left-hand side is at most $T_P(W)$, giving

$$T_P(W) > T_Q(W).$$

From this, given any $c \in C$, we can easily derive

$$\min(b_c, T_P(W)) \geq \min(b_c, T_Q(W)),$$

which completes the proof.

Lemma 2 *Let P be a maximal policy and $W = (C, M)$ be a valid workload. If there exists $\gamma \in C$ such that $P(W, \gamma) < b_\gamma$, then $P(W, C) = M$.*

Proof: By contradiction. Assume the contrary: $P(W, \gamma) < b_\gamma$ and $P(W, C) < M$ and P is maximal. Since P is fair, there is some function T_P such that $P(W, b) = \min(b, T_P(W))$. Define $Q(W, b) = \min(b, T_P(W)\frac{M}{P(W,C)})$. Note that Q is fair. By assumption, $P(W, \gamma) < b_\gamma$, so $P(W, \gamma) = T(W)$. Since $P(W, C) < M$, it follows that $P(W, \gamma) < \min(b_\gamma, T_P(W)\frac{M}{P(W,C)})$. Applying the definition of Q, we get $P(W, \gamma) < Q(W, \gamma)$, which contradicts the assumption that P is maximal.

Lemma 3 *There exists a fair policy.*

Proof: The policy $P((C, M), b) = \frac{M}{|C|}$ is fair.

Lemma 4 *Let F be the set of fair policies, and let $P^*(W, c) = \max_{P \in F} P(W, c)$. Then for any workload $W = (C, M)$, there exists a fair policy P_W such that for all $c \in C$, $P^*(W, c) = P_W(W, c)$.*

Proof of lemma: By contradiction. Assume the contrary: for all fair policies P, there exists $c_P \in C$ such that $P^*(W, c_P) > P(W, c_P)$. Let Q be a fair policy such that for all fair policies R, $Q(W, C) \geq R(W, C)$. (Clearly there must be at least one such policy.) By assumption, there is some $c_Q \in C$ such that $P^*(W, c_Q) > P(W, c_Q)$. By the definition of P^*, there is some fair policy S such that $S(W, c_Q) = P^*(W, c_Q) > P(W, c_Q)$. But now Lemma 1 implies that $S(W, C) > P(W, C)$, so Q was chosen incorrectly, which is a contradiction. Therefore the lemma is true.

Lemma 5 *Let F be the set of fair policies, and let $P^*(W,c) = \max_{P \in F} P(W,c)$. Then P^* is maximal.*

Proof: Lemma 4 implies that P^* is feasible, for if it were not then some P_W would be infeasible. The same lemma also implies that P^* is fair, (with $T_{P^*}(W) = T_{P_W}(W)$). It remains to show only that the last clause of the definition of "maximal" holds; and this follows from the definition of P^*, completing the proof.

Lemma 6 *Let $W = (C, M)$ and $W' = (C', M)$ be workloads, let P be a maximal policy, and let $\gamma \in C$ be a channel. If $C \subseteq C'$, then $P(W', \gamma) \leq P(W, \gamma)$.*

Proof: By contradiction. Assume to the contrary that $P(W,\gamma) < P(W',\gamma)$. It follows that $T_P(W) < T_P(W') \leq b_\gamma$, so $P(W,\gamma) < b_\gamma$. Now Lemma 2 tells us that

$$M = P(W,\gamma) + \sum_{c \in C - \{\gamma\}} \min(b_c, T_P(W)).$$

It follows that

$$M < P(W,\gamma) + \sum_{c \in C - \{\gamma\}} \min(b_c, T_P(W')),$$

or

$$M < \sum_{c \in C} P(W', c) \leq \sum_{c \in C'} P(W', c).$$

This implies that P is not feasible. This contradiction completes the proof.

Theorem 1 *Let b_L be a constant, and let $W = (L, M)$ and $W' = (L \cup A, M)$ be workloads, such that for all $c \in C$, $b_c = b_L$. Let P be a fair, maximal policy. Then*

$$P(W', L) \geq \frac{|L|}{|L| + |A|} P(W, L).$$

Proof: If $b_L \leq T_P(W')$, then the result holds trivially, since the bandwidth of channels in L is not capped. In the alternative case, Lemma 2 tells us that

$$M = \sum_{a \in A} \min(b_a, T_P(W')) + \sum_{c \in L} \min(b_L, T_P(W')).$$

It follows that

$$M \leq \sum_{a \in A} T_P(W') + \sum_{c \in L} T_P(W') = (|A| + |L|) T_P(W').$$

Since P is feasible, $P(W,L) \leq M$, which implies that $P(W,L) \leq (|L| + |A|) T_P(W')$. Rearranging, we get

$$T_P(W') \geq \frac{P(W,L)}{|L| + |A|}.$$

Since bandwidth of L is capped in W', we have

$$P(W', L) = \sum_{c \in L} T_P(W') = |L| T_P(W').$$

Combining the last two results, we get $P(W', L) \geq \frac{|L|}{|L|+|A|} P(W, L)$, which completes the proof.

Lemma 7 *Let P be a maximal policy. Let $W = (L, M)$ and $W' = (L \cup A, M)$ be two workloads. Let $\lambda \subseteq L$ be a set of channels such that for all $c \in \lambda$ $P(W', c) < (1 - \beta) P(W, c)$. Then $|A| > \frac{\beta}{1-\beta} |\lambda|$.*

Proof: Since P is feasible,

$$P(W, L) \leq M. \tag{1}$$

Since some channels in W' (e.g. the ones in λ) have their bandwidth capped, we know by Lemma 2 that all of the bandwidth is allocated in W':

$$P(W', L \cup A) = M. \tag{2}$$

Combining (1) and (2), we get

$$P(W, L) \leq P(W', L \cup A).$$

Breaking each side into its constituent parts, we get

$$P(W,\lambda) + P(W, L - \lambda) \leq P(W',\lambda) + P(W', L - \lambda) + P(W', A). \tag{3}$$

By Lemma 6,

$$\text{for all } c \in L : P(W, c) \geq P(W', c). \tag{4}$$

It follows that

$$P(W', L - \lambda) \leq P(W, L - \lambda),$$

so we can substitute on the right side of (3) and cancel like terms to get

$$P(W,\lambda) \leq P(W',\lambda) + P(W', A). \tag{5}$$

By assumption, $P(W',\lambda) < (1 - \beta) P(W,\lambda)$, so we can substitute on the left side of (5) to get

$$\frac{1}{1 - \beta} P(W',\lambda) < P(W',\lambda) + P(W', A).$$

Combining terms yields

$$\frac{\beta}{1-\beta}P(W',\lambda) < P(W',A).$$

Because P is a fair policy, there must be some threshold $T_P(W')$; and because the bandwidth for each member of λ must be capped in W', we can conclude that $P(W',\lambda) = |\lambda|T_P(W')$. We also know that $P(W',A)$ is at most $|A|T_P(W')$, so we deduce that

$$\frac{\beta}{1-\beta}|\lambda|T_P(W') < |A|T_P(W'),$$

which implies that $\frac{\beta}{1-\beta}|\lambda| < |A|$, completing the proof.

Lemma 8 *Let P be a maximal policy, and let $W = (L,M)$ and $W' = (L \cup A, M)$ be workloads. Let λ be a set of channels such that $\lambda \subseteq L$. Let b_m be a constant such that for all $c \in \lambda$, $b_c \geq b_m$. Then $P(W',\lambda) \geq \frac{|\lambda|^2}{|A|+|\lambda|}P(W,b_m)$.*

Proof: By contradiction. Assume to the contrary that $P(W',\lambda) < \frac{|\lambda|^2}{|A|+|\lambda|}P(W,b_m)$. Then there must be some $\gamma \in \lambda$ such that $\min(b_\gamma, T_P(W')) = P(W',\gamma) < \frac{|\lambda|}{|A|+|\lambda|}P(W,b_m)$. Since the right-hand side is less than b_m, so that $b_\gamma \geq b_m$, it must be that $T_P(W') < \frac{|\lambda|}{|A|+|\lambda|}P(W,b_m)$.

Now let c be an arbitrary member of λ. Then $P(W',c) = \min(b_c, T_P(W')) \leq T_P(W') < \frac{|\lambda|}{|A|+|\lambda|}P(W,b_m)$. Since this is true for an arbitrary member of λ, it must be true for all members of λ.

Now we can apply Lemma 7, with $\beta = \frac{|A|}{|A|+|\lambda|}$ (so that $1-\beta = \frac{|\lambda|}{|A|+|\lambda|}$), to get

$$|A| > \frac{\beta|\lambda|}{1-\beta} = \frac{\frac{|\lambda||A|}{|A|+|\lambda|}}{\frac{|\lambda|}{|A|+|\lambda|}} = |A|.$$

This contradiction completes the proof.

Lemma 9 *Let P be a maximal policy, and let $W = (L,M)$ and $W' = (L \cup A, M)$ be workloads. Let $\lambda \subseteq L$ be chosen by some randomized process, such that $E[|\lambda|] \geq \mu$, and that for all $c \in \lambda$, $b_c \geq b_m$. Then $E[P(W',\lambda)] \geq \frac{\mu^2}{|A|+\mu}P(W,b_m)$.*

Proof: By definition,

$$E[P(W',\lambda)] = \sum_{i=0}^{|L|} \text{Prob}(|\lambda| = i)E[P(W',\lambda)|(|\lambda| = i)].$$

Using Lemma 8 to bound the last expectation, we get

$$E[P(W',\lambda)] \geq \sum_{i=0}^{|L|} \text{Prob}(|\lambda| = i)\frac{i^2}{|A|+i}P(W',b_m),$$

which simplifies to

$$E[P(W',\lambda)] \geq P(W',b_m)E\left[\frac{|\lambda|^2}{|A|+|\lambda|}\right].$$

Because $f(x) = \frac{x^2}{|A|+x}$ has positive second derivative, we have

$$E[P(W',\lambda)] \geq P(W',b_m)\frac{\mu^2}{|A|+\mu},$$

which completes the proof.

Theorem 2 *Let P be a maximal policy. Let b, ρ, n, and a be constants. Let $W = (L,M)$ be a workload, chosen by some randomized process such that $|L| = n-a$ and for all $c \in L$, $\text{Prob}(b_c \geq b) \geq \rho$. Let $W' = (L \cup A, M)$ be another workload such that $|A| = a$. Then $E[P(W',L)] \geq \frac{\rho^2(n-a)^2}{a+\rho(n-a)}P(W,b)$.*

Proof: Let $\lambda = \{c \in L | b_c \geq b\}$. By linearity of expectation, $E[|\lambda|] = \rho(n-a)$. Using Lemma 9, we conclude that $E[P(W',L)] \geq \frac{\rho^2(n-a)^2}{a+\rho(n-a)}P(W,b)$, which completes the proof.

Theorem 3 *Let $W = (L,M)$ be a workload, where L is generated according to $R_\beta(n,k,a)$. Let $W' = (L \cup A, M)$ be another workload, where $|A| = a$. Assume $k < \frac{n}{2}$ and $k\beta \leq M$. Then $E[P(W',L)] \geq \frac{k^2\beta\left(1-\frac{a}{n}\right)^2}{a+k\left(1-\frac{a}{n}\right)}\left(1+O\left(\frac{k}{n}\right)\right)$.*

Proof: Applying Lemma 9, with $b = \beta$ and $\rho = 1 - f_{n,k}(0)$, we get

$$E[P(W',L)] \geq \frac{(1-f_{n,k}(0))^2(n-a)^2}{a+(1-f_{n,k}(0))(n-a)}P(W,\beta). \tag{6}$$

Evaluating $f_{n,k}(0)$, we get

$$1 - f_{n,k}(0) = 1 - \left(1-\frac{1}{N}\right)^k = \frac{k}{n}\left(1+O\left(\frac{k}{n}\right)\right).$$

Since the right side of (6) decreases when $1-f_{n,k}(0)$ decreases, we can substitute to get

$$E[P(W',L)] \geq \frac{k^2\left(1-\frac{a}{n}\right)^2}{a+k\left(1-\frac{a}{n}\right)}P(W,\beta)\left(1+O\left(\frac{k}{n}\right)\right).$$

Since $k\beta \leq M$, it follows that $P(W,\beta) = \beta$, so

$$E\left[P(W',L)\right] \geq \frac{k^2\beta\left(1-\frac{a}{n}\right)^2}{a+k\left(1-\frac{a}{n}\right)}\left(1+O\left(\frac{k}{n}\right)\right),$$

which completes the proof.

Theorem 4 *Let P be a maximal policy, and let $W = (L,M)$ be a workload, where L is generated according to $R_\beta(n,k,a)$. Let $W' = (L \cup A, M)$ be another workload, where $|A| = a$. Assume $\beta = \frac{M}{k}$ and $k > n$. Then $P(W,L) \geq \left(1-\left(\frac{n}{k}\right)^{\frac{1}{3}}\right)^3\left(1-\frac{a}{n}\right)^2 M$.*

Proof: Choose an arbitrary channel $c \in L$. That channel's offered bandwidth, b_c, is distributed binomially, with expectation $\frac{k\beta}{n}$ and variance $\frac{k\beta^2}{n}\left(1-\frac{k}{n}\right)$. Applying Chebyshev's Inequality, we get

$$\text{Prob}\left(\left|b_c - \frac{k\beta}{n}\right| > \sqrt{\frac{k\beta^2\left(1-\frac{k}{n}\right)}{\epsilon n}}\right) < \epsilon,$$

where $0 < \epsilon < 1$. It follows that

$$\text{Prob}\left(b_c > \beta\left(\frac{k}{n} - \sqrt{\frac{k}{n\epsilon}}\right)\right) > 1-\epsilon.$$

Now we apply Theorem 2 to get

$$E\left[P(W',L)\right] \geq \frac{(1-\epsilon)^2(n-a)^2}{a+(1-\epsilon)(n-a)}P\left(W,\beta\left(\frac{k}{n}-\sqrt{\frac{k}{n\epsilon}}\right)\right).$$

Since $1-\epsilon < 1$, we can replace $(1-\epsilon)$ by 1 in the denominator, and simplify to get

$$E\left[P(W',L)\right] \geq \frac{(1-\epsilon)^2(n-a)^2}{n}P\left(W,\beta\left(\frac{k}{n}-\sqrt{\frac{k}{n\epsilon}}\right)\right).$$

Because $\beta k = M$, the $P(W,.)$ term on the right side is not capped, so that

$$E\left[P(W',L)\right] \geq \frac{(1-\epsilon)^2(n-a)^2}{n}\beta\left(\frac{k}{n}-\sqrt{\frac{k}{n\epsilon}}\right).$$

Choosing $\epsilon = \left(\frac{n}{k}\right)^{\frac{1}{3}}$, and recalling that $M = \beta k$, we simplify to get

$$E\left[P(W',L)\right] \geq \left(1-\left(\frac{n}{k}\right)^{\frac{1}{3}}\right)^3\left(1-\frac{a}{n}\right)^2 M,$$

which completes the proof.