An Integrated Approach to Cryptographic Mitigation of Denial-of-Service Attacks

Jothi Rangasamy Douglas Stebila Colin Boyd

Juan González Nieto

Information Security Institute, Queensland University of Technology,
GPO Box 2434, Brisbane QLD 4001, Australia
{j.rangasamy,stebila,c.boyd,j.gonzaleznieto}@qut.edu.au

ABSTRACT

Gradual authentication is a principle proposed by Meadows as a way to tackle denial-of-service attacks on network protocols by gradually increasing the confidence in clients before the server commits resources. In this paper, we propose an efficient method that allows a defending server to authenticate its clients gradually with the help of some fast-to-verify measures. Our method integrates hash-based client puzzles along with a special class of digital signatures supporting fast verification. Our hash-based client puzzle provides finer granularity of difficulty and is proven secure in the puzzle difficulty model of Chen *et al.* (2009). We integrate this with the fast-verification digital signature scheme proposed by Bernstein (2000, 2008). These schemes can be up to 20 times faster for client authentication compared to RSA-based schemes. Our experimental results show that, in the Secure Sockets Layer (SSL) protocol, fast verification digital signatures can provide a 7% increase in connections per second compared to RSA signatures, and our integration of client puzzles with client authentication imposes no performance penalty on the server since puzzle verification is a part of signature verification.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Client/Server*; D.4.6 [**Operating Systems**]: Security and Protection—*Authentication*

General Terms

Security

Keywords

Denial-of-Service, Client Puzzles, Bernstein's Signatures, Secure Sockets Layer (SSL)

1. INTRODUCTION

Denial of Service (DoS) attacks are a growing concern as they aim to disrupt the availability of a target server by exhausting its resources [8]. Thus, when the end host or server is under DoS attack, its service will be unavailable to the legitimate clients. In recent years, several major Internet e-commerce sites were disabled by DoS attacks. Some of the victims were eBay, Yahoo, Amazon, and Microsoft's name server [20].

Authentication is a promising way to treat DoS attacks by restricting connections only to authorised users. However, authentication itself is typically a computationally intensive process. This means that the authentication protocol can become a target of DoS attacks as the attackers can cause the server to perform expensive operations by sending a large number of (bogus) connection requests. Therefore it can cause the same problem it aimed to solving [23].

The SSL/TLS protocol[1] is the most widely used and trusted protocol for secure transactions for sensitive applications ranging from on-line banking and stock trading to e-commerce. A SSL/TLS client is allowed to trigger the SSL/TLS server to perform an expensive RSA operation. By exploiting this, an attacker could easily mount a DoS attack on SSL-based e-commerce sites. On-line shoppers get frustrated and often leave the site without completing their purchase. It was estimated that several billion dollars are lost annually in revenue from e-commerce transactions aborted due to Web performance issues [10, 30].

1.1 Our contributions

In this work, we present an efficient protocol for stronger authentication in the presence of denial of service attacks. The main motivation for this work is to prevent DoS attacks on secure web servers by introducing fast-to-verify authentication measures that reduce the costs incurred by a defending server and at the same time increase the cost of mounting an attack.

Finer granularity hash-based client puzzles. We propose a new client puzzle based on hash functions; our puzzle construction can be seen as a generalisation of the client puzzle of Aura *et al.* [3] with finer granularity in the puzzle's difficulty. We show that our construction is secure in the Chen

[1]Secure Sockets Layer (SSL) version 3.1 is known as Transport Layer Security (TLS) version 1.0).

et al. model for puzzle difficulty [9].

Stronger authentication with fast verification signatures. DoS countermeasures such as client puzzles and cookies give weak authentication. How can we efficiently achieve stronger authentication of clients? We propose a gradual authentication protocol which uses client puzzles and signature schemes for achieving weak and stronger authentication, respectively. By using Bernstein's fast verification digital signature scheme [5], for which verification needs only a few multi-precision integer operations, a defending server can verify client signatures with very low cost. We carefully integrate the weak authentication (client puzzles) with the strong authentication (digital signatures) such that client puzzle verification adds no cost.

Performance analysis. Although client puzzles have been known about for some time, there has been very little experience in using puzzles in practice, particularly in the context of authenticated key exchange. We have implemented our techniques to determine the performance improvement that can be expected in SSL. Our measurements show that our proposed approach can protect SSL servers effectively from DoS attacks that exploit the cost of SSL key agreement.

Outline. In the following sections, we describe our gradual authentication technique and apply it to the TLS protocol to overcome DoS attacks. We organise the rest of the paper as follows: Section 1.2 presents related work, Section 2 presents our client puzzle and its security analysis, Section 3 provides the design of our client authentication technique and its performance results, and Section 4 presents performance results obtained by adding our mechanism to the SSL/TLS protocol. Section 5 concludes and discusses future work.

1.2 Background and related work

Throughout this paper, we are interested in DoS attacks that attempt to consume a server's limited resources such as CPU cycles, memory and network bandwidth. In recent years, a number of techniques, such as client puzzles and stateless connections, have been proposed for deterring resource-consuming DoS attacks [2, 14].

Juels and Brainard [14] first used client puzzles to deter connection depletion attacks. This lead to the development of several client puzzles to protect authentication protocols against DoS attacks [3, 26]. A client puzzle or cryptographic puzzle is a moderately hard computational problem issued by a defending server in response to a client request for its service. The client must return a solution before the server will continue with the protocol. Puzzles are designed so that solving a puzzle is an acceptable cost for legitimate clients but, when trying to establish multiple connections in parallel, the cost will be a significant restraint for attackers.

Stateless connections are another countermeasure proposed to prevent memory-based DoS attacks [2]. This is implemented normally via the use of cookies, a technique proposed by Karn [15]. When an authentication protocol is run, the defending server sends all the session-related information in the form of a cookie. Getting back the cookie in the next round of communication indicates to the server that the client is able to receive messages sent to the IP address from which the communication was initiated. This also protects the server from memory-based DoS attacks in which the attacker uses spoofed addresses.

There are very few implementations showing the efficacy of client puzzles in mitigating DoS attacks available in the literature. Juels and Brainard used client puzzles to counter TCP SYN flooding [14]. They mentioned that SSL also has a similar problem. Wang and Reiter also implemented client puzzles in the TCP stack to show that their puzzle auction mechanism incurs a negligible cost [27]. Feng *et al.* applied client puzzles at the network layer to deter TCP SYN flooding [12].

Both cookies and client puzzles can provide a weak form of authentication, but authentication of the client cannot be ignored altogether and must be completed at some stage in the protocol execution. To thwart DoS attacks, Meadows suggested starting with weak authentication when the protocol execution begins and then gradually increasing the strength of authentication as the protocol runs. This strategy for balancing authentication and computational expenditure is called *gradual authentication* [18, 19].

Protocol designers have tried to reduce the impact of DoS attacks by implementing several countermeasures as a preamble to the network protocols [25, 16, 22, 7]. Dean and Stubblefield first used client puzzles to protect TLS [11]. To the best of our knowledge, we are the first to present a countermeasure in the form of digital signatures. This provides an integrated gradual authentication mechanism that combines a type of hash-based client puzzles with the proposed signature-based countermeasure within an authentication process. Our integrated approach only increases the cost of mounting an attack with client puzzles but adds no extra cost on the server side to verify puzzle solutions. Moreover if puzzle verification fails then the rest of the signature verification is terminated. Although the proposed approach is suitable in general for key exchange protocols that use signatures for client authentication, we focus on preventing DoS attacks on TLS.

2. OUR CLIENT PUZZLE

This section describes our new client puzzle which achieves both efficiency and finer granularity. Granularity indicates how well a server can shift from one difficulty level to the next level to impose a reasonable cost on clients. The more difficulty levels in a puzzle scheme, the more flexible it is for the server. The proposed client puzzle is based on the construction of Aura *et al.* which is an interactive version of Back's proof of work function, Hashcash [4]. We also discuss briefly Aura *et al.*'s puzzle, identify its shortcomings and propose improvements.

Client puzzle of Aura et al. *[3].* In 2000, Aura *et al.* proposed a client puzzle scheme (which we call APuz) based on the problem of finding partial hash preimage of a prespecified "target" string of a given length. In their puzzle protocol, the server sends a periodically generated nonce N_S and the difficulty level Q to the client. The client first generates his own nonce N_C and then finds a required partial collision X such that $H(C, N_S, N_C, X)$ has the required number, Q, of zeros in the output. Figure 1 describes the puzzle APuz of Aura *et al.*

Note that the probabilities of finding partial collisions for any Q-bit string and for the Q-bit string 0^Q are the same, provided H is random. Let d be the decimal value of a Q-bit string: d can be any integer in $[0, 2^Q - 1]$. But the reason to set d to be zero in APuz is that it is simpler and convenient to compare trial collisions. To date, APuz is the most efficient client puzzle proposed in the literature requiring just a single hash computation to verify a puzzle solution.

$$H(C, N_S, N_C, X) \quad = \quad \underbrace{000\ldots000}_{Q\ bits}Y$$

H	$\rightarrow$	a cryptographic hash function
C	$\rightarrow$	the client identity
N_S	$\rightarrow$	the server nonce
N_C	$\rightarrow$	the client's nonce
X	$\rightarrow$	the puzzle solution
Q	$\rightarrow$	the puzzle difficulty level
$\underbrace{000\ldots000}_{Q\ bits}$	$\rightarrow$	the Q-long string of bits 0
Y	$\rightarrow$	the rest of the hash value; can be anything

Figure 1: Client puzzle APuz of Aura *et al.* [3]

Drawbacks. There are two reasons why we need to improve APuz. The first reason is that it fails to perform the functionality of cookies, testing the reachability of the clients at the claimed address: the server publishes its nonce and the puzzle difficulty level, periodically. Therefore puzzles are not uniquely generated as there is no client identity involved in puzzle generation. The second reason is that some puzzles may not have solutions [17]: unlike Juels-Brainard type puzzles, puzzles are not verified during their generation and hence there is no guarantee that a client will find a solution producing the required number of zeros on the hash output. Moreover, the average time needed to solve an instance of APuz is exponential in Q, the difficulty level. Hence, it introduces a big gap between the two nearest difficulties.

2.1 Proposed Client Puzzle GPuz

In this section, we provide an idea to address the issues discussed above. The idea is the following: relax the condition on a puzzle solution so that the probability of solving a puzzle can be increased. That is, instead of fixing the numerical value of the Q-bit target string as $d = 0$, we can relax that the numerical value d of the Q-bit target string could be anything in $[0, D]$, so that $0 \le d \le D < 2^Q$. Note that, while the probability of solving a puzzle with the value of a given target Q-bit string being D is $\frac{1}{2^Q}$, the probability of solving a puzzle with the value of target Q-bit string being d, $0 \le d \le D$ becomes $\frac{D+1}{2^Q}$, assuming the hash function is an ideal random function. Note that we have only increased a client's chance of finding solutions. It is still possible that the client may not find solutions within its running time.

For clarity, we fix the final Q-bits of the hash output as the target string. That is, for a given puzzle with (Q, D), a solution x should satisfy $(h(x) \mod 2^Q) \le D$. For instance, if SHA-1 is used and $(Q, D) = (32, 65537)$, then a string x is a solution, only if (SHA-1(x) $\mod 2^{32}) \le 65537$. Each hash application outputs a number in the interval $[0, 2^{160} - 1]$. If the required part of the output is below the target value, then a solution is found. If not, the process is repeated by incrementing x by one. Note that the lower the target value, the harder the problem solving. Therefore, in addition to Q, a defending server has one more parameter in D to adjust the puzzle difficulty levels. In this way our method allows the server to set more difficulty levels quite smoothly by increasing the target space.

We now describe our client puzzle, GPuz, that is a generalisation of APuz. Figure 2 illustrates the proposed puzzle. Let ID_S and ID_C be the identities of the server and the client, respectively. When a defending server receives a request for service, it sends a challenge consisting of a periodically generated server nonce N_S, a uniquely generated challenge Z and the hardness parameters (Q, D).

Once the client has received the challenge it generates a new nonce, N_C, sets $M = Z||N_S||N_C||ID_S||ID_C$ and finds a partial hash preimage of a Q-bit target string whose integer value is in the range $[0, D]$. That is, the client has to find X such that $H(M, X) \mod 2^Q \le D$.

Now the client replies with its nonce, N_C, and the puzzle solution X, and the other values it received. Then the server first checks if the response is fresh by checking the existence of N_C in the memory and then performs the reachability test by recomputing Z and matching it with the received one. Finally, the server checks if the last Q bits of $H(M, X)$ have integer value in $[0, D]$.

Bitcoin. Our proposed client puzzle is similar to the mathematical problem used in the Bitcoin peer-to-peer cryptocurrency scheme [21]. The main unit of currency of this digital currency system are called Bitcoins. These are not physical objects but numbers produced by network nodes. A network node generates a new coin whenever it finds a solution to a certain mathematical problem. The mathematical problem used in the Bitcoin system is similar to our proposed client puzzle with SHA-256 as the hash function, and the hexadecimal value of D as $0000000000081CD2||0^{48}$, and $Q = 256$.

2.2 Security of Client Puzzles

Recently, Chen *et al.* described a security model for client puzzles, by considering a game between a challenger and an adversary [9]. They proposed formal definition for client puzzles and two properties: puzzle unforgeability and puzzle difficulty. We now give a brief overview of the client puzzle and puzzle difficulty definitions of Chen *et al.*

Definition 1. A client puzzle CPuz consists of the following algorithms [9]:

- Setup: On input 1^k for security parameter,
  - Establishes long term secret key space sSpace, hardness space QSpace, string space strSpace, puzzle instance space puzSpace, and solution space solnSpace
  - Selects the long term puzzle generation key $s \xleftarrow{\$}$ sSpace, and
  - Outputs s and public parameters params.
- GenPuz: Generates a puzzle instance puz $\in$ puzSpace based on inputs long-term secret $s \in$ sSpace, puzzle difficulty $Q \in$ QSpace and string str $\in$ strSpace.
- FindSoln: Finds a potential solution soln $\in$ solnSpace for puzzle puz $\in$ puzSpace within running time t.
- VerAuth: Checks the authenticity of puzzle puz $\in$ puzSpace using long-term secret $s \in$ sSpace.
- VerSoln: Checks the correctness of a potential solution soln $\in$ solnSpace for puzzle puz $\in$ puzSpace.

Definition 2. (Puzzle difficulty) Let Q be a difficulty parameter and let k be a security parameter. Let $\epsilon_{k,Q}(\cdot)$ be

Client		**Server**
	Setup phase	
		Select a secret $K \xleftarrow{\$} \{0,1\}^k$ and generate a nonce $N_S \xleftarrow{\$} \{0,1\}^k$, periodically. Pick the puzzle difficulty level Q and an Q-bit integer D, based on the availability of system resources.
	Puzzle phase	
	$\xrightarrow{\texttt{Request}}$	Compute $Z \leftarrow H(K, Q, D, N_S, ID_C)$
Generate a nonce $N_C \xleftarrow{\$} \{0,1\}^k$ Set $M \leftarrow Z \Vert N_S \Vert N_C \Vert ID_S \Vert ID_C$ Find X such that $H(M, X) \mod 2^Q \leq D$	$\xleftarrow{Q, D, N_S, Z}$	
	$\xrightarrow{Q, D, N_S, Z, N_C, X}$	Check if (N_S, Q, D) is recent and N_C is not used before. Check if $Z \overset{?}{=} H(K, Q, D, N_S, ID_C)$ Set $M \leftarrow Z \Vert N_S \Vert N_C \Vert ID_S \Vert ID_C$ Check if $H(M, X) \mod 2^Q \leq D$

Figure 2: Our client puzzle, GPuz.

a family of monotonically increasing functions and $\mathcal{A}$ be a probabilistic polynomial time algorithm.

For the puzzle difficulty of CPuz, consider the game $\mathsf{Exec}^{Q,\mathsf{DIFF}}_{\mathcal{A},\mathsf{CPuz}}(k)$, between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$, defined as follows:

- The challenger $\mathcal{C}$ sets $(\mathsf{params}, s) \leftarrow \mathsf{Setup}(1^k)$ and output params is given to $\mathcal{A}$.
- The adversary $\mathcal{A}$ can ask any number of CreatePuzSoln(str) queries. For each such query $\mathcal{C}$ generates a fresh puzzle $\mathsf{puz} \leftarrow \mathsf{GenPuz}(s, Q, \mathsf{str})$, finds a solution soln for puz, and returns (puz, soln) to $\mathcal{A}$.
- At any point in time, $\mathcal{A}$ can make a single $\mathsf{Test}(\mathsf{str}^\dagger)$ query. For this query, $\mathcal{C}$ creates a challenge puzzle $\mathsf{puz} \leftarrow \mathsf{GenPuz}(s, Q, \mathsf{str}^\dagger)$ and returns it to $\mathcal{A}$.
- $\mathcal{A}$ outputs a potential solution $\mathsf{soln}^\dagger$. $\mathcal{C}$ returns true if $\mathsf{VerSoln}(\mathsf{puz}^\dagger, \mathsf{soln}^\dagger) = \mathsf{true}$, and false otherwise.

A client puzzle CPuz is said to be $\epsilon_{k,Q}(\cdot)-\mathsf{DIFF}$ if

$$\mathsf{Succ}^{Q,\mathsf{DIFF}}_{\mathcal{A},\mathsf{CPuz}}(k) = \Pr[\mathsf{Exec}^{Q,\mathsf{DIFF}}_{\mathcal{A},\mathsf{CPuz}}(k) = 1] \leq \epsilon_{k,Q}(t)$$

for all probabilistic algorithms $\mathcal{A}$ running in time at most t.

Note that, instead of using the running time of the adversary, puzzle difficulty can also be defined in terms of the number of oracle queries made by an adversary in the random oracle model. In the following sections, we consider each query to the oracle as a step taken by the adversary to solve a puzzle.

2.3 Specification and Security of GPuz

Let $H : \{0,1\}^* \rightarrow \{0,1\}^{2k}$ be a random oracle and $k \in \mathbb{N}$. GPuz is a client puzzle consisting of the following algorithms:

- Setup: On input 1^k for security parameter,
  - Establishes long term secret key space $\mathsf{sSpace} \leftarrow \mathcal{K}$, hardness space $\mathsf{QSpace} \leftarrow \{(Q, D) : Q \in \{0, \ldots, 2k\}, D \in \{0, \ldots, 2^Q - 1\}\}$, string space $\mathsf{strSpace} \leftarrow \{0,1\}^*$, puzzle instance space $\mathsf{puzSpace} \leftarrow \{0,1\}^k \times \mathsf{QSpace}$, and solution space $\mathsf{solnSpace} \leftarrow \{0,1\}^*$.
  - Sets $K \xleftarrow{\$} \mathcal{K}$.
- $\mathsf{GenPuz}(K, (Q, D), \mathsf{str})$: Generates a puzzle instance $\mathsf{puz} = ((Q, D), N_S, Z)$ where $N_S \xleftarrow{\$} \{0,1\}^k$ and Z is computed as $Z \leftarrow H(K, (Q, D), N_S, \mathsf{str})$.
- $\mathsf{FindSoln}(\mathsf{puz}, t)$: For i from 0 to $\max\{t, 2^Q\}$. Set $\mathsf{soln} \leftarrow X \in \{0,1\}^*$: if the first Q bits of $H(\mathsf{puz} : \mathsf{soln})$ have decimal value in $[0, D]$, then output soln.
- $\mathsf{VerAuth}(K, \mathsf{puz}')$: For a puzzle $\mathsf{puz}' = ((Q', D'), N'_S, Z')$, computes $Z \leftarrow H(K, (Q', D'), N'_S, \mathsf{str}')$. If $Z = Z'$, this outputs true and otherwise outputs false.
- $\mathsf{VerSoln}(\mathsf{puz}', \mathsf{soln}')$: If the decimal value of the first Q bits of $H(\mathsf{puz}', \mathsf{soln}')$ does not exceed D, then return true. Otherwise, return false.

From the design principle of our client puzzle GPuz, the unforgeability property of Chen *et al.* is easily achieved here. Since a keyed hash function H is used for puzzle generation, except for the key holder, the probability of generating a valid looking puzzle is negligible in the security parameter, the length of the key.Now we show that GPuz satisfies the difficulty definition of the Chen *et al.* model.

THEOREM 1. *Let* $\epsilon_{k,(Q,D)}(q) = \frac{(D+1)(q+1)}{2^Q}$ *and* H *be a random oracle. Then* GPuz *is an* $\epsilon_{k,(Q,D)}(q)$*-DIFF client puzzle, where* q *is the number of distinct queries to* H.

Proof: We prove the theorem using a counting argument. Let $\mathcal{A}$ be a probabilistic algorithm. Now, fix Q and D. Then, to win the difficulty game, an optimal strategy for $\mathcal{A}$ is to make at most 2^Q oracle calls to H. Let $\bar{X}_1, \bar{X}_2, \ldots \bar{X}_q$ be the q distinct queries made to H. Let $\bar{X} = \{\bar{X}_1, \bar{X}_2, \ldots \bar{X}_q\} \subseteq \{0,1\}^*$ and $\bar{Y}_i$ be the random variable taking values in $\{0,1\}$ such that $\bar{Y}_i = 1$ iff $H(\bar{X}_i) \mod 2^Q \leq D$. Let A_i be the event that $\bar{X}_i$ is a valid solution to GPuz. Then, $\Pr(A_i) = \Pr(\bar{Y}_i = 1) = (D+1)/2^Q$.

Let B be the event that there exists at least one $\bar{X}_j \in \bar{X}$ such that $H(\bar{X}_j) \mod 2^Q \leq D$. That is, $B = \bigcup_{i=1}^{q} A_i$. Then,

$$\Pr(B) = \Pr(\bigcup_{i=1}^{q} A_i) \leq \sum_{i=1}^{q} \Pr(A_i) \leq \frac{(D+1)q}{2^Q}.$$

Note that for any adversary $\mathcal{A}$ making q queries to H, the probability that $\mathcal{A}$ returns a value $\bar{X}_i$ with the left most Q bits of $H(\bar{X}_i)$ being smaller than D is at most $\frac{(D+1)q}{2^Q}$ and if that fails, then he has to guess the solution at random. Thus,

$$\begin{aligned}\mathsf{Succ}_{\mathcal{A},\mathsf{GPuz}}^{(Q,D),\mathsf{DIFF}}(k) &= \Pr[\mathsf{Exec}_{\mathcal{A},\mathsf{GPuz}}^{(Q,D),\mathsf{DIFF}}(k) = 1] \\ &= \Pr[\mathcal{A}\text{wins}|B]\Pr(B) \\ &\quad + \Pr[\mathcal{A}\text{wins}|\bar{B}]\Pr(\bar{B}) \\ &\leq 1 \times \tfrac{(D+1)q}{2^Q} + \tfrac{(D+1)}{2^Q} \times (1 - \tfrac{q}{2^Q}) \\ &\leq \tfrac{(D+1)(q+1)}{2^Q} = \epsilon_{k,(Q,D)}(q).\end{aligned}$$

Strongly difficult puzzles of Stebila et al. *[24].* Stebila *et al.* extended the difficulty notion of Chen *et al.*'s model by showing that for a powerful adversary, solving n puzzles should not be easier than solving one puzzle n times [24]. It can be shown that GPuz is an $\epsilon_{k,(Q,D),n}(q)$-strongly-difficult interactive client puzzle with

$$\epsilon_{k,(Q,D),n}(q) = \left(\frac{(D+1)(q+n)}{n2^Q}\right)^n.$$

3. A GRADUAL AUTHENTICATION PROTOCOL

In this section, we propose an authentication scheme for a defending server to authenticate its clients in a gradual manner. The scheme combines client puzzles for achieving weak authentication with a signature scheme proposed by Bernstein [5] for achieving strong, fast authentication.

3.1 Bernstein's Fast-Verification Digital Signatures

The Rabin-Williams signature system [28] is provably as secure as factorization and had the most efficient verification scheme known until 2000, when Bernstein proposed a Rabin-Williams variant with the same security and signing speed but much faster verification. We refer to this scheme as FVDS (Fast-Verification Digital Signatures).

Definition 3. (FVDS Signature Scheme [5]) Let $\mathcal{M}$ be a set of messages and let $H : \mathcal{M} \times \{r \in \mathbb{Z} : 0 \leq r < 2^\ell\} \rightarrow \{h \in \mathbb{Z} : 0 < h < 2^L, h \mod 8 = 1\}$ be a hash function. The signature schemes consists of the following algorithms:

- KeyGen: Generate an RSA private key $sk = (p,q)$ and corresponding public key $pk = n = pq$ so that $|n| = L$.
- $\mathsf{Sign}(sk = (p,q), m)$: Compute a signature (r,h,f,t,s) such that $0 \leq r < 2^\ell$, $h = H(m,r)$, $f \in \{-2,-1,1,2\}$, $0 \leq s < 2^L$, $0 \leq t < 2^L$, and $s^2 = f \cdot h + t \cdot n$.
- $\mathsf{Verify}(pk = n, m, (r,h,f,t,s))$: Check if $h = H(m,r)$ and $s^2 = f \cdot h + t \cdot n$.

Security. FVDS is secure – existentially unforgeable against adaptive chosen message attacks [13] – under the assumption that factorization of an RSA modulus is harder. Depending on the exact range of r and how f is computed, the security reduction may be tight [6].

Efficiency. Compared with other signatures schemes like RSA and DSA, verification for FVDS needs only a few operations: one hash function, one modular squaring, and two modular multiplications.

Interestingly, the verification operation can be made to work with even smaller integers. The core of the verification operation is checking if $s^2 = f \cdot h + t \cdot n$. As suggested by Bernstein [5], a verifier could do this check modulo a smaller secret random prime. For example, the verifier could pick a small random prime c, compute $s' = s \mod c$, $t' = t \mod c$, $n' = n \mod c$, and $h' = h \mod c$, and then check if $(s')^2 - t' \cdot n' - f \cdot h' \mod c = 0$. For an appropriately sized c (say 115 bits), the chance of fooling the modified verification algorithm is negligible. Alternatively, one could perform this check for several very small (say, 32-bit) primes, as long as the product of these primes exceeded a certain value.

We implemented the FVDS scheme in the OpenSSL open source cryptographic library [29], both as a standalone signature scheme and for use in the SSL protocol (we discuss the latter issue in the next section). In particular, we modified OpenSSL version 1.0.0 to support FVDS using the built-in big-integer operations, and compared the performance of this scheme with the built-in RSA implementation. We carried out performance evaluations on one core of a machine with an Intel Core 2 Duo 2.53GHz (T9400) processor.

Our results are reported in Table 1. FVDS verification significantly outperforms RSA verification; for example, for a 1024-bit modulus, a 64-bit computer can verify 6 times as many FVDS signatures compared to RSA signatures. We observed, however, that there is little performance enhancement from verifying FVDS with smaller primes: except in the case of very large (4096-bit moduli) keys, the cost of several modular reductions (s', t', n', and $h' \mod c$) followed by 1 small mod-squaring and 2 small mod-mults outweighs the cost of 1 full mod-squaring and 2 full mod-mults.[2]

3.2 Signature-based client authentication with client puzzles

We have integrated our proposed client puzzle scheme (Section 2) with the FVDS scheme to develop a client authentication protocol that resists DoS attacks. The main benefit of this tight integration is to provide gradual authentication at reduced cost.

We now discuss the proposed scheme in detail; Figure 3 summarizes the proposed protocol.

- *Setup.* Each client obtains a FVDS public-key / private-key pair using the KeyGen algorithm, and each server obtains an authentic copy of each client's public key.
- *Periodic precomputation.* A defending server periodically selects a secret K, generates N_S and sets puzzle difficulty parameters as (Q,D), based on the availability of system resources.

[2]Where this tradeoff point appears depends on the degree of optimization in the implementation. While our implementation may not achieve the performance extremes of highly-optimized hand-coded assembly, we believe our implementation reflects the performance one might expect from a practical real-world implementation since the modular arithmetic is a widely used, well-developed, mature, and fairly optimized code base.

	32-bit i386 build				64-bit x86_64 build			
modulus	RSA	FVDS			RSA	FVDS		
(bits)	$e = 65537$	full verify	32-bit c	128-bit c	$e = 65537$	full verify	32-bit c	128-bit c
1024	14013	112690	79502	70543	29726	180938	108950	95268
1536	6782	68165	72252	59412	14947	122558	94707	81212
2048	3949	52036	55838	50534	8844	103962	84319	71418
4096	1013	20688	42650	33557	2354	46532	58393	47481

Table 1: Signature verification performance in operations per second (OpenSSL 1.0.0 (modified), Intel Core 2 Duo 2.53GHz T9400, one core).

- *Challenge.* When a server receives a request, it issues the tuple (Q, D, N_S) along with a uniquely generated challenge Z. Here Z is a client-dependent string computed as $Z = H_K(N_S, Q, D, ID_C)$ where ID_C is the client's identity.
- *Solution and client authentication.* Upon receipt of (N_S, Q, D, Z), the client first generates a nonce N_C, sets $M \leftarrow Z||N_S||N_C||ID_S||ID_C$ and finds a value X such that $H(M, X) \mod 2^Q \leq D$. That is, client finds a solution to the associated puzzle. It sets $h \leftarrow H(M, X)$ and computes an FVDS signature $\sigma \leftarrow (X, h, f, t, s)$ on the string M. It finally sends N_S, N_C,Z along with the computed FVDS signature σ to the server.
- *Verification.* First the server checks if N_S is recent and N_C exists in the list of successfully completed connections. Then it performs the reachability test by recomputing Z and matching it with the received one. Finally, the server sets $M \leftarrow Z||N_S||N_C||ID_S||ID_C$ and verifies the FVDS signature $\sigma = (X, h, f, t, s)$ on it by first checking if $h \overset{?}{=} H(M, X)$ and $h \mod 2^Q \leq D$ and then $s^2 \overset{?}{=} f \cdot h + t \cdot n$. If none of the above checks fails, then the server authenticates the client. Otherwise it sends service failure to client.

Note that clients do not have to solve puzzles unless the server is under attack. Therefore, when there is no DoS attack, the server in key establishment protocols can immediately engage in a signature-based authentication by setting the puzzle difficulty Q to 0. In this case, any random value of X is accepted as a solution of a given puzzle.

Implementation. In order to see if the proposed client authentication scheme is a good DoS countermeasure, we carried out performance evaluations as in Section 3.1. For this experiment we modified OpenSSL version 1.0.0 to support signature-based client authentication scheme using the built-in big-integer operations, and compared the performance of this scheme with the built-in RSA implementation. Our results are presented in Table 2.

The experiment for the FVDS-based client authentication scheme is very similar to the one described in Section 3.1. The only difference between the two experiments is a hash operation being performed for puzzle verification in addition to RSA signature verification. Note that results for FVDS in Table 1 and Table 2 are the same as the hash operation is already a part of FVDS signature verification.

Our FVDS-based client authentication scheme significantly outperforms an RSA-based scheme; for example, for a 4096-bit modulus, a 32-bit computer can do 20 times as many FVDS-based protocol runs compared to RSA-based protocol runs. Our technique facilitates counterbalancing computational expenditure by requiring clients to perform more work and servers to perform significantly less work, thereby enabling the server to process more requests. Performance results in Table 2 demonstrate that our suggested method can speed up the client authentication process by a factor of between 6 to 20, depending on the RSA key size. In defense against DoS attacks, this improvement is considerable. Note that the speed up factor increases as the RSA key size increases even with the fixed public exponent. Therefore, our scheme is more favourable for future security levels.

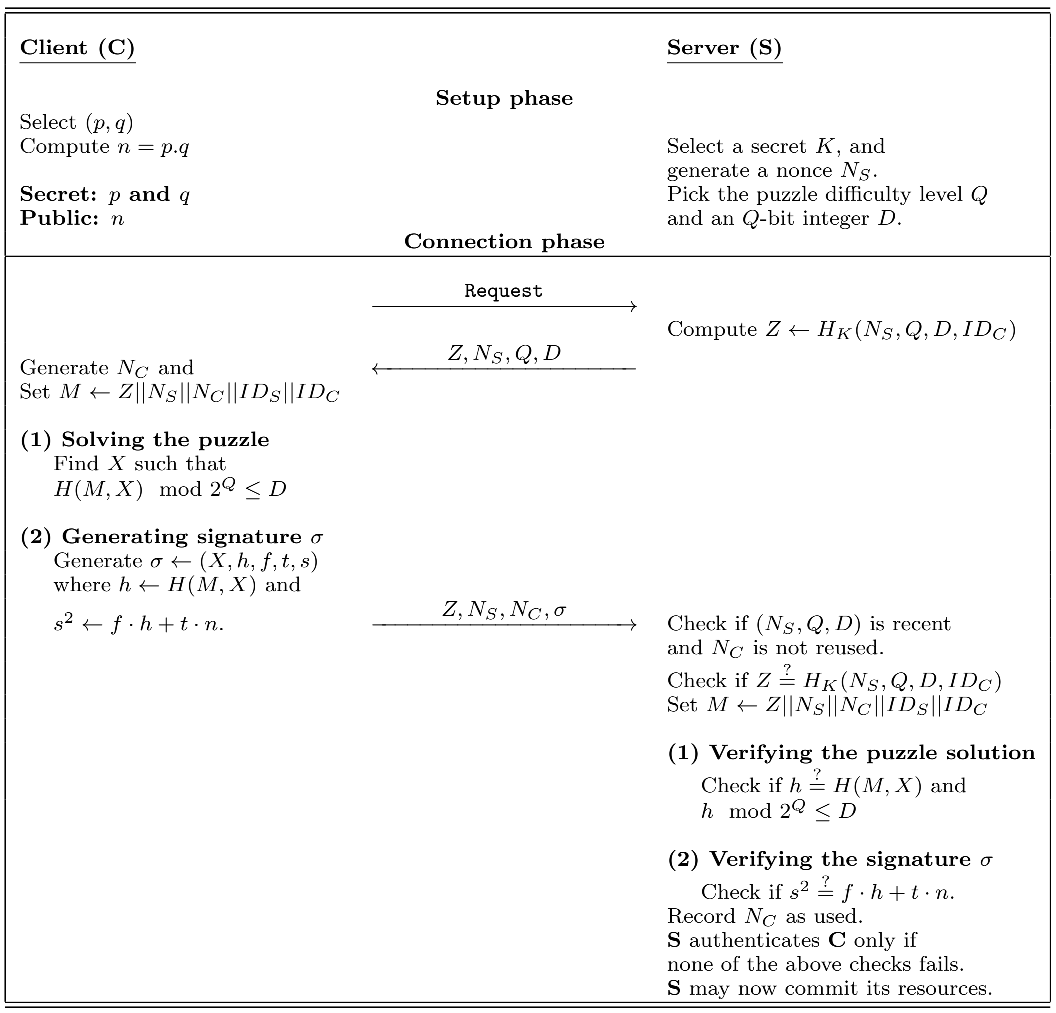
Moreover our FVDS-based client authentication technique can be adapted to work with any network protocol to mitigate DoS attacks effectively. To show its promising performance when adapted in a security protocol, we implemented the proposed FVDS-based client authentication scheme in the OpenSSL open source cryptographic library [29] for use in the SSL protocol. Our results imply that the proposed scheme is a promising and useful companion tool for SSL client puzzles in defense against DoS attacks. More details on this experiment and its results are provided in the following section.

4. DOS RESISTANCE IN SSL

The Secure Sockets Layer (SSL) protocol does not offer any built-in denial of service resistance features. Given its wide-spread deployment on the Internet, and that a server can be directed to do expensive operations with a single simple message from the client, it presents a denial of service risk. Our focus is on mutual authentication in SSL, where both clients and servers authenticate themselves.

We have modified the SSL protocol in two ways to improve its denial of service resistance. The main modification we have made is the implementation of a *cipher suite* involving the FVDS protocol from Section 3.2. In order to evaluate the effectiveness of this new cipher suite, we implemented an alternative client puzzle mechanism and analyzed various attack scenarios.

There are a number of different DoS attack strategies open to an adversary. The most obvious attack is simply to send garbage for both the puzzle solution and the signature. The server will then check the puzzle solution and, in most cases, verification will fail and the connection will be aborted. A more sophisticated attack is when the adversary spends computational effort in solving the puzzle but sends garbage for the signature. In this case the server will check both puzzle and signature before aborting the connection. This latter strategy will benefit most from our FVDS solution and we can expect the 6 to 20 times improvement reported in Section 3.2. In the experiments described in this section we concentrate on the former attack strategy. More modest



Client (C) | **Server (S)**

Setup phase

Client (C):
Select (p, q)
Compute $n = p.q$

Secret: p **and** q
Public: n

Server (S):
Select a secret K, and
generate a nonce N_S.
Pick the puzzle difficulty level Q
and an Q-bit integer D.

Connection phase

$$\xrightarrow{\texttt{Request}}$$

Server (S): Compute $Z \leftarrow H_K(N_S, Q, D, ID_C)$

$$\xleftarrow{Z, N_S, Q, D}$$

Client (C):
Generate N_C and
Set $M \leftarrow Z||N_S||N_C||ID_S||ID_C$

(1) Solving the puzzle
Find X such that
$H(M, X) \mod 2^Q \leq D$

(2) Generating signature σ
Generate $\sigma \leftarrow (X, h, f, t, s)$
where $h \leftarrow H(M, X)$ and
$s^2 \leftarrow f \cdot h + t \cdot n$.

$$\xrightarrow{Z, N_S, N_C, \sigma}$$

Server (S):
Check if (N_S, Q, D) is recent
and N_C is not reused.
Check if $Z \overset{?}{=} H_K(N_S, Q, D, ID_C)$
Set $M \leftarrow Z||N_S||N_C||ID_S||ID_C$

(1) Verifying the puzzle solution
Check if $h \overset{?}{=} H(M, X)$ and
$h \mod 2^Q \leq D$

(2) Verifying the signature σ
Check if $s^2 \overset{?}{=} f \cdot h + t \cdot n$.
Record N_C as used.
S authenticates **C** only if
none of the above checks fails.
S may now commit its resources.

Figure 3: DoS resistant authentication with fast verification signatures and hash-based puzzles

	32-bit i386 build		64-bit x86_64 build	
modulus (bits)	RSA ($e = 65537$) with GPuz	FVDS (full verify) with built-in puzzle	RSA ($e = 65537$) with GPuz	FVDS (full verify) with built-in puzzle
1024	13970	112690	29630	180938
1536	6757	68165	14891	122558
2048	3943	52036	8710	103962
4096	1011	20688	2354	46532

Table 2: Performance of client authentication with puzzle verification in operations per second (OpenSSL 1.0.0 (modified), Intel Core 2 Duo 2.53GHz T9400, one core).

gains can be expected because the faster signature will be only one part of the overall computation at the server.

4.1 Overview of SSL

The SSL protocol has two main components; the Handshake protocol and the Record Layer protocol. The goal of the Handshake protocol is to negotiate a common cipher suite for an SSL client and server, authenticate each other, and establish a shared master secret using public-key cryptographic algorithms.

We now briefly explain how the SSL handshake protocol establishes a master secret key and checks authentication. An overview of the messages in the SSL handshake protocol, including our modifications, appears in Figure 4.

Our focus is on RSA-based key transport cipher suites. The client initiates a session by sending a `ClientHello` message to the server. The server responds with a `ServerHello` message and sends a `ServerCertificate` message containing the server's RSA public key and other information. The client picks a random pre-master secret key, encrypts it under the server's RSA public key, and sends it in the `ClientKeyExchange` message; it also sends a `ClientCertificate` message containing its certificate. The server decrypts the pre-master secret and both parties derive the master secret key by hashing the pre-master secret with the transcript. The parties then exchange `Finished` messages to provide authentication: the client signs a transcript of messages using its signing key, while the server hashes a transcript of messages under a key derived from the master secret key. The master secret key is also used to derive encryption keys for a symmetric cipher, which are then used to protect application data transmitted in the Record protocol.

In the above RSA-based handshake, the server must perform one RSA private key operation – decrypting the pre-master secret – and one public key operation – verifying the client's signature; depending on how client certificates are managed, the server may also need to perform public key operations to validate the client's certificate.

4.2 Modifications to SSL

Client puzzles. In order to support client puzzles, we additionally modified the SSL protocol to carry puzzle data where necessary. The client indicates its support of puzzles with an extension to the `ClientHello` message. The puzzle itself is sent by the server as an extension to the `ServerHello` message. The client includes its puzzle solution in a new `PuzzleSolution` message.

One of the challenges in using client puzzles in SSL is the limited interaction flow between client and server. Ideally, the server should issue a puzzle and then receive and verify the solution before performing any expensive operations. However, in many cipher suites, such as those using ephemeral Diffie-Hellman key agreement, the server must perform an expensive private key operation in the `ServerKeyExchange` message, before having received the client's `PuzzleSolution` message. As such, our client puzzle technique is most applicable to cipher suites without a `ServerKeyExchange` message, such as RSA-based key transport.

FVDS cipher suite. We added a new cipher suite that uses FVDS for client authentication, RSA for key transport, AES128-CBC for symmetric encryption, and SHA-1 as the hash function. This cipher suite also supports an optional client puzzle integrated with the FVDS scheme as described in Section 3.1.

4.3 Performance analysis of SSL with new countermeasures

In order to evaluate the effectiveness of the proposed countermeasure, we made further modifications to OpenSSL (beyond those in Section 3.1) to include support for a hash-based client puzzle and for the FVDS-based authentication protocol with built-in puzzle. We also modified the Apache web server (version 2.2.15) as needed to support these changes. We used the http_load package [1] which can generate many client requests over either http or https (when used with OpenSSL); our modifications ensured that http_load could use the denial of service countermeasures as well.

Our experiments involved a single server (a Linux server with an Intel Core 2 Duo 2.53 GHz (T9400) CPU with 4 GB of RAM, running in an x86_64 architecture) and multiple client machines across a dedicated network with no other traffic or programs running.

We compared three cipher suites. All three cipher suites used RSA-based public key transport, AES128-CBC symmetric encryption, and SHA-1 as the hash function. The difference was in client authentication: one cipher suite used no client authentication, one used RSA signatures for client authentication, and the last used FVDS for client authentication. All public keys were 1024-bit keys.

We performed the following experiments. The results, in connections per second, are reported in Table 3. We ran each test 5 times and averaged the results.

- Test 1: "no puzzle". This test established a baseline of the number of connections per second each cipher suite could handle without any puzzles or denial of service countermeasures. (In other words, for the FVDS-based cipher suite, FVDS signatures from Section 3.1 were used for client authentication but the FVDS-puzzle protocol of Section 3.2 was not used.)

- Test 2: "hash:12, legitimate solutions". This test included a simple hash-based client puzzle based on the hash-inversion puzzle of Aura *et al.* [3]; the client needs to find a preimage x such that the hash value $H(x)$ starts with at least 12 zero bits (where H is the SHA-1 hash function). Our clients simulate solving the puzzle legitimately, allowing us to determine the maximum number of legitimate connections per second the server can handle.

- Test 3: "fvds:12, legitimate solutions". This test, only for the FVDS-based cipher suite, is similar to Test 2 except that the hash-based puzzle is integrated with the FVDS signature generation/verification as in Section 3.2, with $Q = 12$ and $D = 0$.

- Test 4: "hash:12 / fvds:12, garbage solutions". In this test, the clients do not solve the puzzle, instead sending as many fake requests as possible.

- Test 5: "hash:12 / fvds:12, mix legitimate/garbage". In this test, 100 legitimate clients are simulated, as well as a large number of attacking clients sending fake requests.

Observations.

Client		**Server**
`ClientHello` with `PuzzlesSupported` extension†	$\longrightarrow$	

		`ServerHello`
		with `Puzzle` extension†
		`ServerCertificate`*
		`ServerKeyExchange`*
		`CertificateRequest`*

`PuzzleSolution`†	$\longleftarrow$	`ServerHelloDone`

`ClientCertificate`*
`ClientKeyExchange`
`CertificateVerify`*
`Finished`
	$\longrightarrow$	

	$\longleftarrow$	`Finished`
application layer data	$\longleftrightarrow$	application layer data

Figure 4: Messages exchanged in the SSL handshake protocol. * denotes optional messages. † denotes messages added by us for denial of service resistance.

	Key transport $\longrightarrow$	**RSA-1024**	**RSA-1024**	**RSA-1024**
	Client authentication $\longrightarrow$	**none**	**RSA-1024**	**FVDS-1024**
Server configuration	**Client's puzzle strategy**	$\downarrow$	$\downarrow$	$\downarrow$
1: no puzzle		1924	1621	1732
2: hash:12	legitimate solutions	1911	1597	1719
3: fvds:12	legitimate solutions	N/A	N/A	1732
4: hash:12 / fvds:12	garbage solutions	5109	3734	4030
5: hash:12 / fvds:12	mix legitimate/garbage	100 legitimate 4302 garbage	100 legitimate 2767 garbage	100 legitimate 3022 garbage

Table 3: Number of SSL connections per second.

- Adding RSA-based client authentication results in a 16% performance decrease compared to unauthenticated connections.

- Client authentication using FVDS instead of RSA signatures allows for 7% more connections per second. While still an improvement, this is quite small compared to the 6-fold increase in the number of signature verifications per second reported in Table 1: this is because signature verification is a relatively small part of the overall server cost, which is dominated by the cost of the RSA private key decryption operation.

- Verification of fvds:12 puzzles (based on the protocol in Section 3.2) does not add any cost for legitimate connections, whereas verifying a separate hash:12 puzzle adds a 1.5% performance penalty for RSA-based cipher suites.

- In attack scenarios, FVDS-based cipher suites can handle 8% more fake connections than RSA-based cipher suites.

5. CONCLUSION

Denial of service attacks are a challenging threat that cannot be completely prevented. There are many vectors of attack, and in this paper we focused on DoS attacks that exhaust server resources by causing a server to perform many expensive operations. DoS countermeasures, such as client puzzles, can discourage attackers by increasing the resources required to mount an attack.

Our gradual authentication scheme provides an effective, multi-layer, integrated approach to denial of service resistance. Our use of fast-verification digital signatures can provide client authentication between 6 and 20 times faster than conventional approaches, and our integration of puzzle verification in the signature scheme gives puzzle verification at no additional cost.

We integrated our techniques into the SSL protocol and tested our techniques on a dedicated network. Our experimental results indicate that the proposed approach can be effective in mitigating DoS attacks on SSL servers. While there are speed increases offered by fast-verification digital signatures, further improvements could be had if the server's cost of key transport was reduced, which we believe is an important subject of future research in DoS-resistant protocols for real-world networks.

6. ACKNOWLEDGMENTS

The authors would like to thank the ASIACCS anonymous referees for their reviews. The authors are also grateful to Lakshmi Devi Kuppusamy for her valuable comments and helpful advice and Hua Liu for his assistance in experiment implementation. This work was supported by the Australia-India Strategic Research Fund project TA020002.

7. REFERENCES

[1] ACME Labs. http_load, March 2006. URL: http://www.acme.com/software/http_load/.

[2] T. Aura and P. Nikander. Stateless connections. In Y. Han, T. Okamoto, and S. Qing, editors, *Proceedings*

of the First International Conference on Information and Communications Security (ICICS) 1997, volume 1334 of *LNCS*, pages 87–97. Springer, 1997.

[3] T. Aura, P. Nikander, and J. Leiwo. DoS-resistant authentication with client puzzles. In B. Christianson, B. Crispo, J. A. Malcolm, and M. Roe, editors, *Security Protocols: 8th International Workshop*, volume 2133 of *LNCS*, pages 170–177. Springer, 2000.

[4] A. Back. Hashcash: A denial-of-service countermeasure. 2002. URL: http://www.hashcash.org/papers/hashcash.pdf.

[5] D. J. Bernstein. A secure public-key signature system with extremely fast verification, August 2000. URL: http://cr.yp.to/papers.html#sigs.

[6] D. J. Bernstein. Proving tight security for Rabin-Williams signatures. In N. Smart, editor, *Advances in Cryptology – Proc. EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 70–87. Springer, 2008.

[7] C. Castelluccia, E. Mykletun, and G. Tsudik. Improving secure server performance by re-balancing SSL/TLS handshakes. In *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*, pages 26–34. ACM, 2006.

[8] CERT. Denial of service attacks. URL: http://www.cert.org/tech_tips/denial_of_service.html, 3 May 2010.

[9] L. Chen, P. Morrissey, N. P. Smart, and B. Warinschi. Security notions and generic constructions for client puzzles. In M. Matsui, editor, *Advances in Cryptology – Proc. ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 505–523. Springer, 2009.

[10] C. Coarfa, P. Druschel, and D. Wallach. Performance analysis of TLS web servers. *ACM Transactions on Computer Systems*, 24(1):39–69, 2006.

[11] D. Dean and A. Stubblefield. Using client puzzles to protect TLS. In *Proc. 10th USENIX Security Symposium*, 2001.

[12] W. Feng, E. Kaiser, and A. Luu. Design and implementation of network puzzles. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 4, pages 2372–2382. IEEE, 2005.

[13] S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

[14] A. Juels and J. Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *Proc. Internet Society Network and Distributed System Security Symposium (NDSS) 1999*, pages 151–165. Internet Society, 1999.

[15] P. Karn and W. A. Simpson. Photuris: Session-key management protocol, March 1999. RFC 2522. URL: http://www.ietf.org/rfc/rfc2522.txt.

[16] C. Kaufman. Internet Key Exchange (IKEv2) protocol, December 2005. RFC 4306. URL: http://www.ietf.org/rfc/rfc4306.txt.

[17] V. Laurens, A. El-Saddik, and A. Nayak. Requirements for client puzzles to defeat the denial of service and the distributed denial of service attacks. *International Arab Journal of Information Technology*, 3(4):326–333, 2006.

[18] C. Meadows. A formal framework and evaluation method for network denial of service. In *Proc. 12th IEEE Computer Security Foundations Workshop (CSFW) 1999*, page 4. IEEE, 1999.

[19] C. Meadows. A cost-based framework for analysis of denial of service networks. *Journal of Computer Security*, 9(1/2):143–164, 2001.

[20] D. Moore, C. Shannon, D. J. Brown, G. M. Voelker, and S. Savage. Inferring internet denial-of-service activity. *ACM Transactions on Computer Systems (TOCS)*, 24(2):115–139, 2006.

[21] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009. URL: http://www.bitcoin.org/bitcoin.pdf.

[22] J. Smith, J. González Nieto, and C. Boyd. Modelling denial of service attacks on JFK with Meadows's cost-based framework. In *Proceedings of the 2006 Australasian Workshops on Grid Computing and e-research*, volume 54, pages 125–134. Australian Computer Society, Inc., 2006.

[23] J. Smith, S. Tritilanunt, C. Boyd, J. González Nieto, and E. Foo. Denial of service resistance in key establishment. *International Journal of Wireless and Mobile Computing*, 2(1):59–71, 2007.

[24] D. Stebila, L. Kuppusamy, J. Rangasamy, C. Boyd, and J. González Nieto. Stronger difficulty notions for client puzzles and denial-of-service-resistant protocols. In A. Kiayias, editor, *Topics in Cryptology – CT-RSA 2011 – The Cryptographers' Track at the RSA Conference*, volume 6558 of *LNCS*, pages 284–301. Springer, 2011. URL: http://eprint.iacr.org/2010/649.pdf

[25] D. Stebila and B. Ustaoglu. Towards denial of service resilient key agreement protocols. In C. Boyd and J. González Nieto, editors, *Proceedings of the 14th Australasian Conference on Information Security and Privacy (ACISP) 2009*, volume 5594 of *LNCS*, pages 389–406. Springer, 2009.

[26] S. Tritilanunt, C. Boyd, E. Foo, and J. González Nieto. Toward non-parallelizable client puzzles. In F. Bao, S. Ling, T. Okamoto, H. Wang, and C. Xing, editors, *Cryptology and Network Security (CANS) 2007*, volume 4856 of *LNCS*, pages 247–264. Springer, 2007.

[27] X. Wang and M. K. Reiter. Defending against denial-of-service attacks with puzzle auctions. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy (SP'03)*, pages 78–92. IEEE Press, 2003.

[28] H. Williams. A modification of the RSA public-key encryption procedure. *IEEE Transactions on Information Theory*, 26(6):726–729, 1980.

[29] E. A. Young and T. J. Hudson. The OpenSSL project, 2007. URL: http://www.openssl.org.

[30] Zona Research. The need for speed II. April, 2001. URL: http://www.keynote.com/downloads/Zona_Need_For_Speed.pdf.