Available online at www.sciencedirect.com

SciVerse ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

Modular square root puzzles: Design of non-parallelizable and non-interactive client puzzles

Yves Igor Jerschow [a,*], Martin Mauve [b]

[a] Institute for Experimental Mathematics, University of Duisburg-Essen, Ellernstraße 29, 45326 Essen, Germany
[b] Institute of Computer Science, Heinrich Heine University, Universitätsstraße 1, 40225 Düsseldorf, Germany

ARTICLE INFO

Article history:
Received 31 May 2012
Received in revised form
3 November 2012
Accepted 19 November 2012

Keywords:
Client puzzles
Denial of Service (DoS)
Network protocols
Authentication
Computational puzzles

ABSTRACT

Denial of Service (DoS) attacks aiming to exhaust the resources of a server by overwhelming it with bogus requests have become a serious threat. Especially protocols that rely on public key cryptography and perform expensive authentication handshakes may be an easy target. A well-known countermeasure against resource depletion attacks are client puzzles. The victimized server demands from the clients to commit computing resources before it processes their requests. To get service, a client must solve a cryptographic puzzle and submit the right solution. Existing client puzzle schemes have some drawbacks. They are either parallelizable, coarse-grained or can be used only interactively. In case of interactive client puzzles where the server poses the challenge an attacker might mount a counterattack on the clients by injecting faked packets with bogus puzzle parameters bearing the server's sender address. In this paper we introduce a novel scheme for client puzzles which relies on the computation of square roots modulo a prime. Modular square root puzzles are non-parallelizable, i.e., the solution cannot be obtained faster than scheduled by distributing the puzzle to multiple machines or CPU cores, and they can be employed both interactively and non-interactively. Our puzzles provide polynomial granularity and compact solution and verification functions. Benchmark results demonstrate the feasibility of our approach to mitigate DoS attacks on hosts in 1 or even 10 Gbit networks. In addition, we show how to raise the efficiency of our puzzle scheme by introducing a bandwidth-based cost factor for the client. Furthermore, we also investigate the construction of client puzzles from modular cube roots.

1. Introduction

Denial of Service (DoS) attacks aiming to exhaust the resources of a server by overwhelming it with bogus requests pose an increasing threat to network protocols not only in the Internet. Corporate Intranets and public local area networks like Wi-Fi hotspots also constitute promising targets for mounting an effective DoS attack. Especially protocols that perform authentication and key exchange relying on expensive public key cryptography are likely vulnerable to DoS, e.g., SSL/TLS, IPsec, or IEEE 802.1X (EAPOL). By flooding valid-looking requests, for example authentication handshakes, an attacker may try to overload his victim. But even services that do not involve expensive operations can be vulnerable to DoS attacks that exploit worst-case behavior of classical data structures like hash tables (Crosby and Wallach, 2003). A well-known countermeasure against resource exhaustion are *client puzzles* (Juels and Brainard, 1999; Back, 2002; Aura et al., 2001). A server being under attack processes requests only from those clients that themselves spend resources in solving a cryptographic puzzle and submit the right solution. By imposing a computational task on the client the victimized

* *Corresponding author*. Tel.: +49 2011837637.
E-mail addresses: jerschow@iem.uni-due.de (Y.I. Jerschow), mauve@cs.uni-duesseldorf.de (M. Mauve).
0167-4048/$ – see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cose.2012.11.008

server dramatically cuts down the number of valid requests that the attacker can emit. However, benign hosts having only a single request are hardly penalized. The puzzle difficulty (i.e., the time it takes the client to solve the challenge) should be adjustable from easy to hard, while puzzle verification must be always cheap so that it can be performed at full link speed. Otherwise an attacker could mount a second DoS flooding attack with bogus puzzle solutions to overwhelm the server. A widely-used cost function for client puzzles is the reversal of a one-way hash function by brute force. Verifying such a puzzle involves only a single hash operation.

Client puzzles can be *interactive* or *non-interactive*. In the first case, as shown in Fig. 1, the server constructs the puzzle upon receiving a request and demands from the client to solve it before continuing with the protocol. In the latter case the client constructs the puzzle by itself, solves it and attaches the solution to its request. An important characteristic of client puzzles is *granularity*, i.e., the ability to finely adjust the puzzle difficulty to different levels. Another desirable property is *non-parallelizability*, which prevents an attacker from obtaining the solution faster than scheduled by distributing the puzzle to multiple CPU cores or to other compromised machines (Tritilanunt et al., 2007; Schaller et al., 2007; Karame and Čapkun, 2010). Existing client puzzle schemes are either parallelizable, coarse-grained or can be used only interactively. Interactive puzzles have the drawback that the packet with the puzzle parameters sent from server to client lacks authentication. A second DoS attack against the clients with faked packets pretending to come from the defending server and containing bogus puzzle parameters may thwart the clients' connection attempts. Such a counter-attack becomes feasible if no address authenticity is provided by the underlying layers, e.g., if operating at the link layer. To the best of our knowledge, no puzzle scheme proposed in the literature provides all the desired properties.

In this paper we introduce a novel scheme for client puzzles based on the computation of square roots modulo a prime. *Modular square root puzzles* are non-parallelizable, can be employed both interactively and non-interactively and provide polynomial granularity. We construct the puzzle for a particular request by assigning to it a unique quadratic residue a modulo a prime p. Then the client solves the puzzle by extracting the modular square root x of a and sends it to the server as proof of work. Computation is performed by repeated squaring, which is assumed to be an intrinsically sequential process. Fig. 2 illustrates our scheme in a non-interactive scenario. Verifying the puzzle on the server side is easy—it requires a single modular squaring operation and a few hash operations. Puzzle difficulty can be tuned by selecting a larger or smaller prime modulus. We evaluate the performance of modular square root puzzles by benchmarking the verification throughput and the solution time for different levels of difficulty. The results demonstrate the feasibility of our approach to mitigate DoS attacks on hosts having 1 or even 10 Gbit links. To compensate for raising verification costs in high-speed networks we strengthen our puzzle scheme by introducing a bandwidth-based cost factor for the client. Furthermore, we also investigate the construction of client puzzles from modular cube roots.

The remainder of this paper is organized as follows. In the next section, we discuss existing approaches for DoS protection with the aid of puzzles. Section 3 introduces algorithms for computing modular square roots, investigates parallelization aspects, and forms the mathematical basis for our client puzzles. In Section 4 we then describe how to construct, solve and verify a modular square root puzzle, which can be employed in a non-interactive or interactive manner. Section 5 evaluates the performance of our puzzle scheme and extends it by a bandwidth-based cost factor. Finally, we conclude the paper with a summary in Section 6.

2. Related work

Comprehensive surveys on DoS/DDoS attacks and proposed defense mechanisms can be found in Peng et al. (2007), Douligeris and Mitrokotsa (2004), Mirkovic and Reiher (2004). Peng et al. (2007) classify four categories of defense: (1) attack prevention, (2) attack detection, (3) attack source identification, and (4) attack reaction. Juels and Brainard (1999) introduced *client puzzles* to protect servers from TCP SYN flooding attacks. This countermeasure falls into the last category and constitutes a currency-based approach where clients have to pay before getting served. Being under attack, a server distributes to its clients cryptographic puzzles in a stateless manner asking them to reverse a one-way hash function by brute force. The difficulty of the puzzle is chosen depending on the attack strength. Only after receiving a correct solution from the client the server allocates resources for the dangling

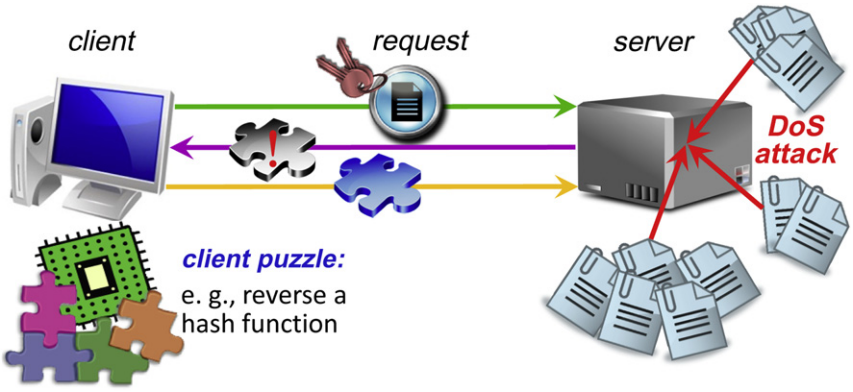

Fig. 1 – Interactive client puzzle scheme.

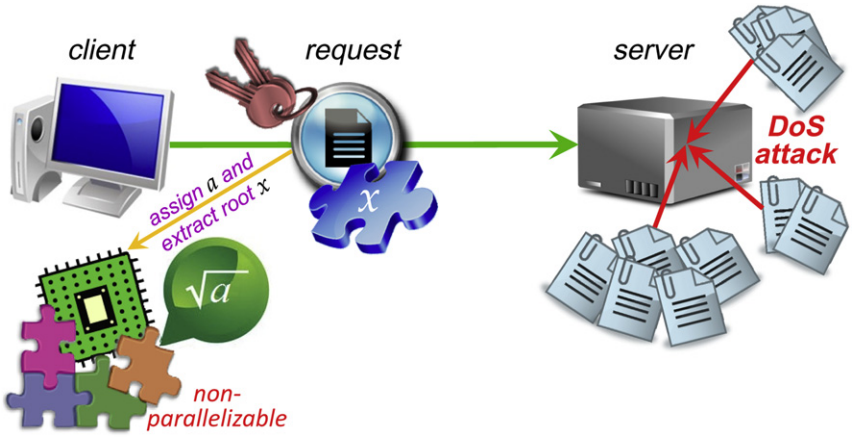

Fig. 2 – Modular square root puzzle (solve $x^2 \equiv a \pmod p$) employed non-interactively.

TCP connection. The idea of CPU-bound client puzzles has been applied to authentication protocols in general by Aura et al. (2001). An implementation of client puzzles to protect the TLS handshake against DoS is described in Dean and Stubblefield (2001).

Hash-reversal puzzles can be used both interactively and non-interactively. They are simple to construct and verify but have the disadvantage of being highly parallelizable and provide only exponential granularity. The task of reversing a one-way hash function by brute force can be easily distributed across many machines. To make them fine-grained Feng et al. (2005) proposed *hint-based hash reversal puzzles* where the server gives the client a hint about the range within which the solution lies. Thus, the granularity becomes linear. The drawback is that hint-based puzzles can be employed only interactively. The authors designed a puzzle architecture, called *network puzzles*, which relies on hint-based hash reversal puzzles and operates at the "weakest link"—the IP layer—to make client puzzles universally usable. The feasibility of the puzzle protocol has been demonstrated through an implementation on Linux with *iptables*. The authors use ICMP source quench messages to deliver puzzles and IP options to transmit client cookies and puzzle answers.

Waters et al. (2004) suggested a client puzzle scheme based on the Diffie–Hellman key exchange where puzzle construction and distribution are outsourced to a secure entity called *bastion*. The bastion periodically issues puzzles for a specific number of virtual channels that are valid during the next time slot. Puzzle construction is quite expensive since it requires a modular exponentiation, but many servers can rely on puzzles distributed by the same bastion. A client solves a puzzle by computing the discrete logarithm through brute force testing—a task that is highly parallelizable. The granularity of the puzzle is linear. On the server side, verifying a puzzle involves a table lookup and another costly modular exponentiation, which, however, is performed in advance during the previous time slot.

Tritilanunt et al. (2007) introduced a non-parallelizable client puzzle scheme that is based on the *subset sum problem*. The client solves the puzzle by applying Lenstra's lattice reduction algorithm LLL. However, the authors point out that the memory requirements for LLL are quite high, which results in some implementation issues. Puzzle verification is quite cheap. It takes one hash operation and about 25–100 additions. Subset sum puzzles are interactive and provide polynomial granularity. In contrast, our puzzle scheme can be also employed non-interactively, has a small memory footprint, and is easy to implement.

Non-parallelizable puzzles based on repeated squaring are well-known in timed-release cryptography. Rivest et al. (1996), introduced interactive *time-lock puzzles* to encrypt messages that can be decrypted by others only after a pre-determined amount of time has passed. Like the RSA cryptosystem time-lock puzzles rely on the intractability of factoring large integers. Constructing a time-lock puzzle requires the server to perform an expensive modular exponentiation. In detail, to encrypt a message m for a period of T seconds Alice:

- generates the RSA modulus $n = pq$ and computes $\varphi(n) = (p-1)(q-1)$.
- determines the number of squaring operations modulo n per second, denoted by S, that can be performed by the solver Bob, and computes $t = T \cdot S$.
- encrypts m with a symmetric cipher using the key K.
- picks a random a, $1 < a < n$, and encrypts K as:

$$C_K = K + a^{2^t} \bmod n. \tag{1}$$

To make the exponentiation efficient, Alice reduces the exponent modulo $\varphi(n)$ by computing:

$$r = 2^t \bmod \varphi(n) \tag{2}$$

and obtains $a^{2^t} \bmod n$ from $a^r \bmod n$.

- outputs the time-lock puzzle (n, a, t, C_K).

To reveal K from C_K, Bob needs to compute $a^{2^t} \bmod n$ and in contrast to Alice cannot take the shortcut via $\varphi(n)$, since determining $\varphi(n)$ is provably as hard as factoring n. Instead, Bob must do the computation step by step by repeatedly performing modular squarings—altogether t times, which is a non-parallelizable task and takes T seconds.

Seeking for a non-parallelizable (but still interactive) client puzzle scheme Karame and Čapkun (2010) adapted Rivest's puzzle by employing an RSA key pair with small private exponent to reduce the costs for puzzle verification. The server

must still perform a modular exponentiation but the number of multiplications is decreased by some factor, e.g., factor 12.8 for a 1024-bit modulus resulting in 120 modular multiplications instead of 1536. We find that these verification costs are nevertheless too high to provide a viable DoS protection for high-speed links. In contrast, verifying our modular square root puzzle takes only a single modular squaring operation.

With the discussed RSA based puzzle schemes we share the idea of a non-parallelizable solution function that relies on modular exponentiation. Apart from that, our approach is different and does not use any trapdoor information. Dwork and Naor (1992) mentioned the extraction of modular square roots as one of three candidate families of pricing functions to combat spam. Our main contribution here to counteract DoS attacks is the computation of modular square roots from so-called "hard" primes resulting in a novel scheme for non-parallelizable client puzzles. Moreover, we point out a counterattack on interactive client puzzles, argue for a non-interactive scheme, and discuss the pros and cons.

Wang and Reiter (2008) proposed a multi-layer framework for puzzle-based DoS protection, which embeds puzzle techniques into both IP-layer and end-to-end services. The authors have presented two mechanisms: *Congestion puzzles* address bandwidth-exhaustion attacks in routers by cooperatively imposing puzzles to clients whose traffic is traversing a congested link. A traffic flow must be accompanied by a corresponding computation flow of puzzle solutions. The second mechanism called *puzzle auctions* protects an end-to-end service like TCP against protocol-specific DoS attacks. Clients bid for server resources by tuning the difficulty of the hash-reversal puzzle that they solve and the server allocates its limited resources to the highest bidder first.

Martinovic et al. (2008) addressed DoS attacks in IEEE 802.11 networks aiming to exhaust the access point's (AP) resources by flooding it with faked authentication requests. The authors introduced *wireless client puzzles* that are distributed by a defending AP to joining stations. To support highly heterogeneous stations these puzzles are not CPU-bound. Instead of inverting a one-way function, a station has to measure the signal strength of the links to its neighbors and to find out those neighbors, whose link reaches a certain *Neighborhood Signal Threshold (NST)*. The NST is randomly chosen and frequently changed by the AP. A station replying with a wrong solution is detected by its neighbors, which thereupon issue a warning to the AP.

Further client puzzle architectures are, e.g., Hlavacs et al. (2008), Schaller et al. (2007), Tang and Jeckmans (2010). Puzzle-based DoS defense mechanisms can also rely on other payment schemes than CPU cycles, for example on memory (Abadi et al., 2005; Dwork et al., 2003; Doshi et al., 2006), bandwidth (Walfish et al., 2006; Jerschow et al., 2009), or human interaction where so-called *CAPTCHAs* (von Ahn et al., 2003) have become the most common technique. Besides DoS protection various other applications for computational puzzles have been proposed, e.g., mitigating spam (Dwork and Naor, 1992; Back, 2002), uncheatable benchmarks (Cai et al., 1993), a zero-knowledge protocol for timed-release encryption and signatures (Mao, 2001), a timed commitment scheme for contract signing (Boneh and Naor, 2000), or offline submission of documents (Jerschow and Mauve, 2010).

3. Modular square roots

3.1. Extracting square roots modulo a prime

Let p be an odd prime and $a \in \mathbb{Z}_p^*$ an integer, i.e., $1 \leq a \leq p-1$. The solution of the congruence $x^2 \equiv a \pmod{p}$ is called a *square root modulo p*. There exist either two solutions x and $-x$ or no solution. In the first case, a is named a *quadratic residue*, and in the latter case a *quadratic non-residue* modulo p. Half of the elements in $\mathbb{Z}_p^*$ are quadratic residues and the other half are quadratic non-residues. To express whether a is a quadratic residue or not the *Legendre symbol* $\left(\frac{a}{p}\right)$ is used. It is defined as being 1 if a is quadratic residue, -1 if a is a quadratic non-residue and 0 if operating in $\mathbb{Z}_p$ and $a = 0$. The Legendre symbol can be efficiently computed in $\mathcal{O}((\log p)^2)$ bit operations (Cohen, 1996; Menezes et al., 1996).

Finding a square root modulo p is quite easy for half of the primes p, namely if $p \equiv 3 \pmod 4$. In this case the solution is given by

$$x = a^{(p+1)/4} \bmod p. \tag{3}$$

For half of the remaining primes where $p \equiv 5 \pmod 8$ a less trivial, but also straightforward solution exists:

$$x = \begin{cases} a^{(p+3)/8} \bmod p & \text{if } a^{(p-1)/4} \bmod p = 1 \\ 2a(4a)^{(p-5)/8} \bmod p & \text{otherwise.} \end{cases} \tag{4}$$

The remaining case $p \equiv 1 \pmod 8$ is the most difficult one. However, there exist two well-known algorithms (Bach and Shallit, 1996; Nishihara et al., 2009) to compute square roots modulo p for all primes p, namely the *Tonelli–Shanks method* (Tonelli, 1891; Shanks, 1972) (see Algorithm 1 (Menezes et al., 1996)) and the *Cipolla–Lehmer method* (Cipolla, 1903; Lehmer, 1969) (see Algorithm 2 (Menezes et al., 1996)). The group-theoretic Tonelli–Shanks method has a running time of $\mathcal{O}((\log p)^4)$ bit operations if $p-1$ contains a large power of two in its prime factorization. But for small s (see line 3) it runs in $\mathcal{O}((\log p)^3)$ since in this case the for loop is executed only a small number of times. The Cipolla–Lehmer method is based on the theory of finite fields and works with polynomials over the field $\mathbb{Z}_p$. In contrast to the algorithm of Tonelli–Shanks its running time does not depend on the decomposition of $p-1$ and is always in $\mathcal{O}((\log p)^3)$. Note that for primes p where s is very small the Tonelli–Shanks algorithm will outperform the Cipolla–Lehmer method, because an exponentiation in the polynomial ring $\mathbb{Z}_p[x]$ is more expensive than in $\mathbb{Z}_p$. Both algorithms have a probabilistic component, namely finding a quadratic non-residue modulo p. For the Tonelli–Shanks method this quadratic non-residue does not depend on a and can be precomputed if p is fixed. A random integer $b \in \mathbb{Z}_p$ is a quadratic non-residue with probability 0.5. In case of the Cipolla–Lehmer method we need to know a to find a suitable quadratic non-residue and the probability for succeeding with a random integer b is $0.5 - 1/2p$ (Bach and Shallit, 1996), which converges to 0.5 for large primes p. On average, two trials should suffice for both methods to find a quadratic non-residue. The time required for this test is negligible compared to the total computation of

the square root. It is an open question whether randomization can be eliminated, although this will be possible if the extended Riemann hypothesis turns out to be true. So far modular square roots can be computed only in random polynomial time by a Las Vegas algorithm (Bach and Shallit, 1996).

3.2. *Modular exponentiation*

Extracting a modular square root requires to perform modular exponentiations. This task can be accomplished by the basic *binary exponentiation method* (commonly referred to as square-and-multiply) or a more sophisticated algorithm like the *k-ary method* or the *sliding-window method* (Menezes et al., 1996). In case $p \equiv 3 \pmod 4$ only one modular exponentiation is needed. If $p \equiv 5 \pmod 8$ then two modular exponentiations have to be performed. Finally, if $p \equiv 1 \pmod 8$ the Tonelli–-Shanks or Cipolla–Lehmer algorithm has to be applied. In the worst case, namely if s is large, the Tonelli–Shanks method carries out up to $\mathcal{O}(\log p)$ modular exponentiations in the for loop and becomes quite inefficient. Primes $p \equiv 1 \pmod 8$ of appropriate size where the prime factorization of $p-1$ contains a large power of two can be easily found. We suggest Algorithm 3 for this purpose. In line 5 the function *IsProbablePrime()* repeatedly performs a randomized primality test, e.g., the Miller–Rabin test, to achieve a given error bound (which is less than 4^{-k} after k rounds in case of the Miller–-Rabin test). Finding such a "hard" prime p with an error probability below 10^{-15} takes less than 50 ms for a 1031-bit prime (input: $l = 1024$) and less than 1 s for a 2058-bit prime (input: $l = 2048$) on a modern 64-bit CPU.

Algorithm 1. Tonelli–Shanks: square roots modulo a prime p.

Input: an odd prime p and an integer a, $1 \le a \le p-1$.
Output: the two square roots of a modulo p, provided a is a quadratic residue modulo p.

1: Compute the Legendre symbol $\left(\frac{a}{p}\right)$. **if** $\left(\frac{a}{p}\right) = -1$ **then print** *"a has no square roots modulo p"* and terminate.
2: Find a quadratic non-residue b modulo p at random, i. e., an integer b, $1 \le b \le p-1$, with $\left(\frac{b}{p}\right) = -1$.
3: Write $p-1 = 2^s t$, where t is odd.
4: Compute a^{-1} *mod* p by the extended Euclidean algorithm.
5: Set $c \leftarrow b^t$ *mod* p and $r \leftarrow a^{(t+1)/2}$ *mod* p.
6: **for** $i = 1$ to $s-1$ **do**
7: Compute $d = (r^2 \cdot a^{-1})^{2^{s-i-1}}$ *mod* p.
8: **if** $d \equiv -1$ $(mod\ p)$ **then** set $r \leftarrow r \cdot c$ *mod* p.
9: Set $c \leftarrow c^2$ *mod* p.
10: **end for**
11: **return** $(r, -r)$

Algorithm 2. Cipolla–Lehmer: square roots modulo a prime p.

Input: an odd prime p and an integer a, $1 \le a \le p-1$.
Output: the two square roots of a modulo p, provided a is a quadratic residue modulo p.

1: Compute the Legendre symbol $\left(\frac{a}{p}\right)$. **if** $\left(\frac{a}{p}\right) = -1$ **then print** *"a has no square roots modulo p"* and terminate.
2: Choose an integer $b \in \mathbb{Z}_p$ at random until $b^2 - 4a$ is a quadratic non-residue modulo p, i. e., $\left(\frac{b^2-4a}{p}\right) = -1$.
3: Let f be the polynomial $x^2 - bx + a$ in $\mathbb{Z}_p[x]$.
Compute $r = x^{(p+1)/2}$ *mod* f. (Note: r will be an integer.)
4: **return** $(r, -r)$

Algorithm 3. Finding a "hard" prime for modular square roots.

Input: minimal bit length l.
Output: the smallest prime p having at least l bits with $p-1=2^s t$ where t is odd and s in $\mathcal{O}(log\ p)$.

1: set $i \leftarrow 1$
2: **repeat**
3: $p = (2^{l-1} \cdot i) + 1$
4: set $i \leftarrow i + 2$
5: **while not** *IsProbablePrime(p)*
6: **return** p

In the following, we thus concentrate on such "hard" primes and the Cipolla–Lehmer method, which ignores the structure of $p-1$. Here the computation consists of a single modular exponentiation $x^{(p+1)/2} \bmod f$, but with polynomials instead of integers. The modulus f is a polynomial of degree 2 with leading coefficient 1. How many modular multiplication/squaring operations on integers are involved in this exponentiation? First, we observe that if p is *a* "hard" prime the exponent $(p+1)/2$ has the form $2^{S-1} \cdot i + 1$ where i is a small integer. Only some of the most significant bits and the least significant bit are set. Hence, the computation actually reduces to an exponentiation with a power-of-two exponent, where repeated squaring—a special case of the binary exponentiation—constitutes the most efficient technique. To compute $g^y \bmod n$ with $y = 2^k$ it takes k modular squarings and no additional multiplications while $\lfloor \log y \rfloor$ is the lower bound for the number of multiplications to carry out a single exponentiation in a general group. Squaring a polynomial $ax + b$ of degree 1 over the field $\mathbb{Z}_p$ requires 3 modular integer multiplications/squarings. Reducing the resulting polynomial of degree 2 modulo f, i.e., performing a polynomial division, involves 2 modular multiplications and 2 modular subtractions on integers. While modular multiplication/squaring of N-bit numbers runs in $\mathcal{O}(N^2)$ (or in $\mathcal{O}(N^{1.585})$ with a sophisticated technique like Karatsuba's algorithm), modular subtraction takes linear time, and thus is negligible. Altogether, the modular exponentiation in $\mathbb{Z}_p[x]$ takes about $5 \cdot \log p$ modular multiplication/squaring operations on integers.

3.3. *Non-parallelizability*

In all exponentiation algorithms the main workload accounts to repeatedly performing modular squarings. This is assumed to be an intrinsically sequential, i.e., non-parallelizable process since each next step requires the intermediate result from the previous one (Rivest et al., 1996). Parallelization of the squaring operation itself cannot achieve a significant speedup either. Each squaring requires only trivial computational resources and any non-trivial scale of parallelization inside the squaring operation would be likely penalized by communication overhead among the processors (Mao, 2001). In complexity theory, the class P contains all decision problems that can be solved by a deterministic Turing machine in polynomial time. NC ⊆ P represents the class of problems that can be efficiently solved by a parallel computer. However, it is still an open question whether modular exponentiation is P-complete, i.e., not in NC (Adleman and Kompella, 1988; Sorenson, 1999). Likewise, it is unknown if factoring is really not in P.

We now want to point out those parts of modular square root computation that are parallelizable. If applying the basic binary exponentiation method the $1/2 \cdot \log p$ multiply steps can be performed in parallel to the $\log p$ squaring steps. Thus, only $\log p$ sequential modular squaring operations can be accounted for when extracting a square root modulo $p \equiv 3 \pmod 4$. The same applies to the case $p \equiv 5 \pmod 8$ where two modular exponentiations are performed (see Equation (4)). Instead of evaluating $a^{(p-1)/4} \bmod p$ first and then deciding on which will be the second exponentiation, one could carry out all three modular exponentiations in parallel and then determine the correct square root instantly by checking the result of $a^{(p-1)/4} \bmod p$. When dealing with "hard" primes $p \equiv 1 \pmod 8$ parallelization is also possible to some degree. We can do the 3 modular multiplications/squarings to square the polynomial simultaneously. Afterward the 2 modular multiplications for polynomial division can be also performed in parallel. This results in about $2 \cdot \log p$ sequential modular multiplications to compute a square root modulo a "hard" prime $p \equiv 1 \pmod 8$ and takes more than twice as long as for other primes, since multiplying is somewhat slower than squaring (GMP). Thus we have found a way to increase the time for square root extraction by more than factor 2, which cannot be diminished by raising the number of available processors.

4. Client puzzles from modular square roots

4.1. *Constructing and solving a non-interactive puzzle*

The benign host A having a request (e.g., an authentication handshake) to host B that is under a DoS attack constructs for its request a unique puzzle. We suppose that both parties share a list $L = \{p_1, \ldots, p_j\}$ of "hard" primes $p \equiv 1 \pmod 8$ with different bit lengths which have been generated once and henceforth can be used by all hosts an unlimited number of times. The puzzle must be bound to A's request message m. Depending on the layer the protocol is operating at m may be an Ethernet frame, an IP datagram or a TCP/UDP segment. First, host A selects from the list L a prime p of appropriate bit length n and applies a cryptographic hash function H with digest length k on m recursively $c = \left\lceil \frac{n}{k} \right\rceil$ times to produce the $(n-1)$-bit digest:

$$d = First_{n-1}(H(m) \parallel H(H(m)) \parallel \ldots \parallel H^c(m)\ ). \tag{5}$$

Here $\parallel$ denotes the concatenation of two bit strings and $First_i$ extracts the first i bits from a bit string. Next host A considers d as a $(n-1)$-bit number and computes the Legendre symbol $\left(\frac{d}{p}\right)$ to check whether d is a quadratic residue modulo p. If it turns out to be a quadratic non-residue, d is decremented by one until the quadratic residue a is found:

Algorithm 4. Assigning a unique quadratic residue to the digest d, method 1.

set $a \leftarrow d$

while $\left(\frac{a}{p}\right) = -1$ **do**
  set $a \leftarrow a - 1$
end while

return a

Since half of the elements in $\mathbb{Z}_p^*$ are quadratic residues, a few trials will usually suffice. A more efficient and deterministic approach for the puzzle solver A to generate a quadratic residue from the digest d is the following method:

Algorithm 5. Assigning a unique quadratic residue to the digest d, method 2.

Precondition: A (small) quadratic non-residue b modulo p has been found.
  if $\left(\frac{d}{p}\right) = 1$ **then** set $a \leftarrow d$
  else set $a \leftarrow b \cdot d \ mod \ p$

return a

According to the properties of the Legendre symbol, the product of two quadratic non-residues is a quadratic residue. Unfortunately, $\left(\frac{1}{p}\right) = 1$ for all p and two other simple candidates for b, namely -1 and 2, also are quadratic residues if $p \equiv 1 \pmod 8$. Thus, some other (small) number has to be found for b. This can be done in advance for each prime from the list L. Method 2 requires one evaluation of the Legendre symbol and at most one modular multiplication. However, as we will point out in the next subsection, applying the second method makes the verification of the puzzle more expensive compared to the first method.

Now, a unique quadratic residue a has been assigned to A's request. The puzzle to solve is the computation of the square root of a modulo p by applying the Cipolla–Lehmer method, which takes about $2 \cdot \log p$ sequential modular multiplications. Without parallelization, about $5 \cdot \log p$ modular multiplications/squarings have to be performed. Having extracted the square root x, host A attaches this n-bit number to its request and sends it to host B. The other square root $-x$ is of no importance for the protocol. There is no need to transmit the prime p. Host A can simply indicate the modulus by stating its position in the list L. Usually, all primes in the list will differ in size so that the corresponding prime may even be deduced from the size of x.

4.2. *Puzzle verification*

The victimized host B verifies the puzzle solution x prior to allocating resources and processing host A's request, which may require to perform a public or even private key operation or an expensive database lookup. Puzzle verification is quite cheap—besides a few hash operations (c times, depends on the hash size and the length of the prime) to compute the digest d from the request only a single modular squaring operation $x^2 \bmod p$ has to be carried out.

If the first method (Algorithm 4) has been applied for assigning a quadratic residue to the digest d, then host B does not need to rerun the algorithm to verify the quadratic residue $a = x^2 \bmod p$ presented by the puzzle solver A. With probability 0.5 we have $a = d$, with probability 0.25 we have $a = d - 1$ and so on. Thus, if $d - (x^2 \bmod p) < \delta$ where δ is a small constant, e.g., $\delta = 20$, the verification can be considered as successful, otherwise A's request is dropped. This check requires only a single modular subtraction and a comparison. Host A cannot take any advantage of extracting the modular square root from $a' = a - \beta$ instead of from a if β is bounded by the small constant δ. Even if host A cheats in this manner for some reason, host B can be certain that A has indeed computed a modular square root specially for its request m. A drawback of the second method (Algorithm 5) is that the verifier B has to rerun it to ensure that the puzzle solver A has actually extracted the modular square root from the quadratic residue that belongs to the digest d.

Host B's decision whether to allocate resources for processing A's request or not can, of course, also depend on the puzzle difficulty (that is, on the size of the chosen prime) and on the strength of the ongoing DoS attack. The rate of accepted requests with correct puzzle solutions shall not exceed host B's processing capacity, i.e., the rate at which B can actually complete these requests. Being rejected, host A may then retry by taking a larger prime from the list L and solving a more difficult puzzle.

4.3. *Puzzle granularity and public auditability*

The ability to finely adjust the puzzle difficulty to different levels represents an important criterion for the practical applicability of a puzzle. Solving a modular square root puzzle with an N-bit prime takes $\mathcal{O}(N^3)$ time while the verification runs in $\mathcal{O}(N^2)$. Thus, having polynomial granularity, our puzzle is quite fine-grained. In contrast, a non-interactive puzzle scheme based on hash-reversal has exponential granularity and is highly parallelizable. Since a third party can efficiently verify the solution of the square root puzzle without access to any trapdoor information, its cost-function is called *publicly auditable* (Back, 2002). Time-lock (Rivest et al., 1996) and Diffie–Hellman based (Waters et al., 2004) puzzles are, by contrast, not publicly auditable.

4.4. *Interactive client puzzles*

Our modular square root puzzles can be also employed in an interactive way, where the victimized server (host B) issues a challenge to the client (host A), as is the case with client puzzles proposed by Juels and Brainard (1999) and reworked by Aura et al. (2001). In the interactive setting the prime modulus p and the quadratic residue a are dictated by the server. This can be done in a stateless manner by hashing the client's request along with a secret number to produce the

digest d and sending d back to the client, which derives from it the quadratic residue a for the puzzle. Thus, the server needs to store only the secret number and the prime which are reused across all clients.

The advantages of interactive client puzzles are the prevention of precomputation attacks and the precise choice of the puzzle's level of difficulty since it is prescribed by the defending server. However, a major drawback of interactive client puzzles that we have already indicated in the introduction is the lack of authentication for the packet containing the puzzle parameters, which the server sends to the client. A second DoS attack against prospective clients with faked packets bearing the server's sender address and containing bogus puzzle parameters may thwart the clients' connection attempts. A client receiving a plethora of bogus challenges that were possibly chosen to be even more difficult than the puzzle of the genuine server may easily become overwhelmed. Most likely, it will not be able to solve the authentic challenge and thus its request will not be processed by the server. Depending on the chosen puzzle difficulty, even a modest puzzle packet rate may be sufficient for the attacker to succeed. The feasibility of such a counterattack depends on the network environment and the attacker's location. Forging the sender address and eavesdropping on the traffic is especially easy in wired and wireless LANs while it is more difficult in the Internet. The ability to eavesdrop on the traffic significantly alleviates the puzzle attack since the attacker gets to know the clients that currently issue a request. Hence, only in environments where counterattacks on the clients are very unlikely, our square root puzzles should be used in the interactive manner.

4.5. *Non-interactive client puzzles from a random beacon*

In case of the favored non-interactive puzzle construction an attacker might compute the puzzle solutions in advance. If precomputation is an issue, it can be mitigated by concatenating the message m with an unpredictable, periodically changing number pior to producing the digest d. Lottery results (Back, 2002) or stock market prices are possible sources of randomness which are easily accessible to both parties A and B. In this case host B will accept only requests bearing an up-to-date random number. In Jerschow and Mauve (2012), we have fundamentally solved the precomputation issue of non-interactive client puzzles by deriving the puzzle from a periodically changing, secure random beacon. We now briefly sketch the central ideas of our secure client puzzle architecture.

The beacons are generated in advance for a longer time span and periodically broadcasted in the LAN by a special *beacon server*. All hosts obtain a signed *fingerprint package* consisting of cryptographic digests of these beacons in advance. Verifying a beacon is very easy—it takes only a single hash operation, which can be performed at line speed by all hosts. Thus, DoS attacks on the beacon service are virtually impossible. Since the beacon server does not need to interact with any host, it can even drop all incoming packets without inspecting them to be resistant against network-based attacks of any kind. Broadcasting beacons is its sole task. If a server becomes overloaded due to a DoS attack, it asks all clients to solve and submit a puzzle prior to processing their requests. A client constructs a non-interactive puzzle by taking its request and the current beacon as input for a cost function. This can be, e.g., the computation of a modular square root or the reversal of a one-way hash function by brute force. Having solved the puzzle, the client attaches the puzzle parameters and the solution to the pending request and retransmits it. To provide a robust and secure beacon service, we have addressed time synchronization aspects and especially elaborated the deployment of beacon fingerprints. Even if hosts were not able to obtain the signed fingerprint package in advance using one of the regular distribution channels, they can acquire it on the fly from the beacon server and verify its signature despite of possible DoS flooding attacks. The secure client puzzle architecture is primarily designed for LANs. But the beacon service can be adopted to operate with a single beacon server in Intranets or even in the Internet by employing multicast or unicast transmissions or even by resorting to DNS.

4.6. *Client puzzles from modular cube roots?*

We have investigated whether our non-parallelizable and non-interactive client puzzles can be improved by resorting to modular cube roots instead of modular square roots. Obviously, verifying a modular cube root is about twice as expensive since in $x^3 \bmod p$ a modular squaring and a modular multiplication have to be carried out. What about the computation of modular cube roots? Like with modular square roots, the difficulty of solving the congruence $x^3 \equiv a \pmod p$ depends on the prime p. If $p \equiv 2 \pmod 3$ extracting the cube root modulo p is very easy—it requires a single modular inversion and exponentiation (Bach and Shallit, 1996). The remaining case $p \equiv 1 \pmod 3$, and especially if $p \equiv 1 \pmod 9$, is the difficult one (Nishihara et al., 2009). For $p \equiv 1 \pmod 3$ one third of the elements in $\mathbb{Z}_p$ are cubic residues. Adleman et al. (1977) generalized the Tonelli–Shanks method to compute n-th roots in $\mathbb{Z}_p$. Its running time again depends on the decomposition of $p-1$, in case of cube roots on $p-1 = 3^s t$ where $3 \nmid t$, and is in $\mathcal{O}((\log p)^4)$ in the worst case. Nishihara et al. (2009), proposed two algorithms to extend the Cipolla–Lehmer method for cube root computation. Its running time is always in $\mathcal{O}((\log p)^3)$ since it ignores the structure of $p-1$. To extract a modular cube root an irreducible monic polynomial f in $\mathbb{Z}_p[x]$ of degree 3 has to be constructed first. This step requires randomization and in case of the more efficient algorithm it takes one modular exponentiation per trial to verify f. The success probability is approximately 2/3. The actual cube root computation is very similar to the Cipolla–Lehmer method and consists of a single exponentiation in the polynomial ring $\mathbb{Z}_p[x]$:

$$r = x^{\left(p^2+p+1\right)/3} \bmod f. \quad \text{(Note : } r \text{ will be an integer.)} \tag{6}$$

To perform this exponentiation, at least $2 \cdot \log p$ squarings in $\mathbb{Z}_p[x]$ have to be carried out. Squaring a polynomial of degree 2 over the field $\mathbb{Z}_p$ requires 6 modular integer multiplications/squarings. Note that they can be performed in parallel. Reducing the resulting polynomial of degree 4 modulo f by means of a polynomial division takes two sequential steps

each one involving 4 modular integer multiplications, which are also parallelizable. Assuming maximal parallelization, this results in at least $6 \cdot \log p$ sequential modular multiplications/squarings on integers to carry out the exponentiation. Taking also the construction of f into account, it requires at least $7 \cdot \log p$ sequential modular multiplications/squarings to solve a modular cube root puzzle versus $2 \cdot \log p$ sequential operations in case of modular square roots. Since the verification of modular cube roots is twice as expensive, the complexity gain with respect to non-parallelizability is about 1.75. We observe that constructing client puzzles from modular cube roots is an interesting option, but it also disproportionately increases the workload for benign hosts which probably solve the puzzle without parallelization.

5. Evaluation and protocol enhancements

In this section we evaluate the performance of our puzzle scheme by comparing the puzzle verification throughput on the victimized server with the time it takes the client to solve a puzzle. We aim to show that puzzle difficulty (i.e., the solution time for the client) can be tuned from easy to hard by raising the size of the prime while for the server puzzle verification is still cheap enough to be performed at full link speed. Meanwhile, we enhance our scheme by introducing a bandwidth-based cost factor for the client.

5.1. *Puzzle benchmark*

For "hard" primes of different size ranging from 264 to 8206 bits we measure the number of modular square root puzzles that an off-the-shelf Intel Core 2 Quad Q9400 2.66 GHz CPU can verify per second and the time it takes to solve a single puzzle. Table 1 presents our benchmark results averaged over 10 runs. In all test series the coefficient of variation was below 1.5%, which can be attributed to slightly different CPU scheduling behavior of the operating system across the runs, since the computation itself is deterministic. For the large-integer arithmetic we employ the well-known open source library *GMP* from GNU (GMP), which claims to be faster than any other bignum library by using state-of-the-art algorithms with highly optimized assembly code. Modular square root extraction is done using the Cipolla–Lehmer method, where the exponentiation in $\mathbb{Z}_p[x]$ constitutes the main workload. In our measurements we take only the time to perform the $2 \cdot \log p$ sequential modular multiplications into account, since the remaining $3 \cdot \log p$ modular multiplications/squarings can be computed in parallel by a well-versed attacker (see Section 3.3). All computations are performed using a single CPU core. For full parallelization of a puzzle an attacker would employ three CPU cores while the defending host can verify as many puzzles in parallel as CPU cores are available. Solving a puzzle on a benign host that uses only a single CPU core actually takes about two and a half times longer than stated in Table 1. To accelerate the repeated modular multiplications we make use of Montgomery reduction instead of performing the classical reduction by dividing. This results in a speed-up by a factor of 1.2–2.0, especially for small moduli in the order of 264–2058 bits.

Evaluating the benchmark results, we first observe that a 64-bit implementation outperforms its 32-bit counterpart by a factor of up to 3.7 in verifying and up to 4.0 in solving a puzzle. Since almost all desktop CPUs manufactured during the last five years are 64-bit capable and 64-bit operating systems are widely available, we consider the 64-bit results as reference values. Secondly, the speed gap between the verifier and the solver (when comparing the time to verify and to solve a single puzzle) constitutes factor 236 for a 264-bit puzzle and increases up to factor 18 640 for a 8206-bit puzzle. Now the main question to pose is whether the verification throughput of modular square root puzzles is high enough to cope with a DoS flooding attack of bogus puzzle solutions mounted at full link speed. Of course, the size of a valid-looking request containing a puzzle solution plays a role. Before we can definitely answer this question with "yes" for networks with 100 Mbit, 1 Gbit, and even 10 Gbit links, we extend the puzzle protocol by a small bandwidth-based cost factor for the client.

The victimized host demands that valid puzzle solution packets must be padded with zeros to have full MTU (Maximum Transmission Unit) size. In the Internet, the MTU usually is 1500 bytes (in Gigabit Ethernet even up to 9000 bytes). Hence, besides solving a puzzle the client must additionally pay with bandwidth. Using bandwidth as a currency for DoS protection is a known approach in the literature (Walfish et al., 2006; Jerschow et al., 2009). Now, dealing with 1500 byte packets, the

Table 1 – Benchmark: verifying and solving modular square root puzzles on Intel Core 2 Quad Q9400 2.66 GHz.

Bit length	Modular squarings/sec (one CPU core)		Modular square root: time in msec (assuming full parallelization)	
	32-Bit	64-Bit	32-Bit	64-Bit
264	1,377,000	2,597,000	0.238	0.091
520	593,500	1,354,000	1.35	0.411
776	329,400	698,300	4.15	1.10
1031	201,300	549,400	9.01	2.42
1547	102,500	337,400	27.7	7.09
2058	62,810	199,100	62.9	15.7
3084	33,030	117,100	196	48.1
4106	20,530	71,630	429	109
6155	10,620	39,250	1350	340
8206	6810	24,430	3020	763

victimized host will receive up to 8300 (100 Mbit link), 83 000 (1 Gbit link) or 830 000 (10 Gbit link) valid-looking puzzle solutions per second. We note that it will perfectly cope with 8206-bit puzzles on a 100 Mbit link, with 3084-bit puzzles on a 1 Gbit link and with 520-bit puzzles on a 10 Gbit link assuming a single CPU core engaged in puzzle verification. The time to compute the digest d must also be taken into account. But only the meaningful part of the request and not the whole packet needs to be hashed, while cryptographic hash functions like MD5 or SHA-1 process about 2.8–3.6 Gbit of data per second on our test machine. Furthermore it is conceivable to produce the $(n-1)$-bit digest d by applying a very fast pseudorandom number generator to $H(m)$ instead of executing the hash function c times. On the opposite side it takes an attacker 763 ms to solve a 8206-bit puzzle, 48.1 ms to solve a 3084-bit puzzle, and 0.411 ms to solve a 520-bit puzzle, respectively, assuming full parallelization.

5.2. Increasing the bandwidth-based payment

Besides prescribing that puzzle solution packets must be padded to have full MTU size we may go a step further and increase the bandwidth-based payment requested from the client. The victimized host can demand to receive from the client multiple copies of the puzzle solution packet prior to processing the associated request. By this means we reduce the maximum number of valid-looking puzzle solutions that have to be taken into account and verified per second. This enables us to employ more complex puzzles in high-speed networks and thus to strengthen the DoS protection. For example, by prescribing that clients must send four copies of their puzzle solution packet we can cut down on the number of valid-looking puzzle solutions to process per second by factor four and verify even 8206-bit puzzles on a 1 Gbit link. Sending multiple copies of the puzzle solution packet is feasible for all clients regardless of their link speed, while DoS protection schemes based solely on bandwidth payment penalize clients behind slow links. To implement this protocol extension, the victimized host must maintain a packet counter for each client. An appropriate data structure for this purpose is a hash map with the client's address as the key and the pair <*packet counter*, *timestamp*> as the value. Elements with old timestamps must be purged periodically from the hash map. Storage overhead for maintaining the counters is fairly low: Assuming 10 bytes per client, a 1 Gbit link with 83,000 packets/s, and a maximum lifetime of 5 s for each entry, the size of the hash map will be about 10 MB (depending on implementation and pointer size).

5.3. Discussion

Besides providing non-parallelizability, granularity, and the possibility of non-interactive usage a good client puzzle scheme must enable to adjust the puzzle difficulty from easy to hard, while puzzle verification should remain cheap enough to be performed at full link speed. Though for modular square root puzzles the level of difficulty cannot be chosen arbitrarily high without rendering the verification too expensive (i.e., becoming not verifiable at full link speed any more), we are convinced that the presented solution times for the client in the order of 0.1–1000 ms are fully viable for DoS prevention in practice. For this range verification at full link speed can be ensured for the victimized host. Solution times much greater than 1 s are possible with hash-reversal puzzles, but for benign clients such long delays seem to be hardly reasonable. In a nutshell, we believe that our modular square root puzzles provide all the desirable properties to protect today's networks with links up to 10 Gbit against DoS.

5.4. FPGAs

Fast modular exponentiation has been also successfully implemented in hardware, especially on FPGAs (Ciaran McIvor et al., 2003; Suzuki, 2007), and for modern GPUs (Szerwinski and Güneysu, 2008; Harrison and Waldron, 2009), which are very competitive. A few years ago FPGAs outperformed ordinary software implementations, but a current comparison (Szerwinski and Güneysu, 2008) shows that nowadays FPGAs are about as fast as software implementations on up-to-date CPUs. A GPU implementation pays off when performing a large number of modular exponentiations simultaneously. However, this comes at the expense of high latency. A speed-up of up to 4 times compared to a modern CPU has been reported in Harrison and Waldron (2009). Though an experienced attacker can benefit from such hardware acceleration, his advantage over a regular solver running a software implementation is bounded by a small factor. In general, this is not an issue for the client puzzle protocol.

6. Conclusion

In this paper we have introduced a novel client puzzle scheme based on modular square roots as a countermeasure against DoS attacks. A modular square root puzzle is non-parallelizable, i.e., the solution cannot be obtained faster than scheduled by distributing the puzzle to multiple machines or CPU cores. Our puzzles can be employed non-interactively, which prevents counterattacks on the client mounted by injecting packets with fake puzzle parameters. Providing polynomial granularity and compact solution and verification functions, modular square root puzzles can be easily implemented to safeguard network protocols, especially those performing expensive public key authentication, against DoS. We have shown how to raise the efficiency of our puzzle scheme by introducing a bandwidth-based cost factor for the client and demonstrated its feasibility in 1 and 10 Gigabit networks through benchmarking.

REFERENCES

Abadi M, Burrows M, Manasse M, Wobber T. Moderately hard, memory-bound functions. ACM Transactions on Internet Technology 2005;5:299–327.

Adleman L, Kompella K. Using smoothness to achieve parallelism. In: STOC '88: proceedings of the 20th annual ACM Symposium on Theory of Computing; 1988. p. 528–38.

Adleman L, Manders K, Miller G. On taking roots in finite fields. In: SFCS '77: proceedings of the 18th annual Symposium on Foundations of Computer Science; 1977. p. 175–8.

von Ahn L, Blum M, Hopper NJ, Langford J. CAPTCHA: using hard AI problems for security. In: EUROCRYPT '03: proceedings of the 22nd international conference on theory and applications of cryptographic techniques; 2003. p. 294–311.
Aura T, Nikander P, Leiwo J. DOS-resistant authentication with client puzzles. In: Revised papers from the 8th international workshop on security protocols; 2001. p. 170–7.
Bach E, Shallit J. Algorithmic number theory. In: Efficient algorithms, vol. I. MIT Press; 1996.
Back A. Hashcash – a denial of service counter-measure, http://www.hashcash.org/papers/hashcash.pdf; 2002.
Boneh D, Naor M. Timed commitments. In: CRYPTO '00: proceedings of the 20th annual international cryptology conference on advances in cryptology; 2000. p. 236–54.
Cai J-Y, Lipton RJ, Sedgewickand R, Yao AC-C. Towards uncheatable benchmarks. In: Proceedings of the 8th annual structure in complexity theory conference; 1993. p. 2–11.
Ciaran McIvor MM, McCanny J, Daly A, Marnane W. Fast Montgomery modular multiplication and RSA cryptographic processor architectures. In: Proceedings of the 37th asilomar conference on signals, systems, and computers; 2003. p. 379–84.
Cipolla M. Un metodo per la risolutione della congruenza di secondo grado. Rendiconto dell'Accademia Scienze Fisiche e Matematiche 1903;9(3):154–63.
Cohen H. A course in computational algebraic number theory. Springer; 1996.
Crosby SA, Wallach DS. Denial of service via algorithmic complexity attacks. In: SSYM'03: proceedings of the 12th conference on USENIX security symposium; 2003.
Dean D, Stubblefield A. Using client puzzles to protect TLS. In: SSYM'01: proceedings of the 10th USENIX security symposium; 2001.
Doshi S, Monrose F, Rubin AD. Efficient memory bound puzzles using pattern databases. In: ACNS 2006: proceedings of the 4th international conference on applied cryptography and network security; 2006. p. 98–113.
Douligeris C, Mitrokotsa A. DDoS attacks and defense mechanisms: classification and state-of-the-art. Computer Networks 2004;44(5):643–66.
Dwork C, Naor M. Pricing via processing or combatting junk mail. In: CRYPTO '92: proceedings of the 12th annual international cryptology conference on advances in cryptology; 1992. p. 139–47.
Dwork C, Goldberg A, Naor M. On memory-bound functions for fighting spam. In: CRYPTO '03: proceedings of the 23rd annual international cryptology conference on advances in cryptology; 2003. p. 426–44.
GMP: GNU multiple precision arithmetic library. http://gmplib.org.
Harrison O, Waldron J. Efficient acceleration of asymmetric cryptography on graphics hardware. In: AFRICACRYPT '09: proceedings of the 2nd international conference on cryptology in Africa; 2009. p. 350–67.
Hlavacs H, Gansterer WN, Schabauer H, Zottl J, Petraschek M, Hoeher T, et al. Enhancing ZRTP by using computational puzzles. Journal of Universal Computer Science 2008;14(5): 693–716.
Jerschow YI, Mauve M. Offline submission with RSA time-lock puzzles. In: CIT 2010: proceedings of the 10th IEEE international conference on Computer and Information Technology; 2010. p. 1058–64.
Jerschow YI, Mauve M. Secure client puzzles based on random beacons. In: IFIP networking 2012: proceedings of the 11th international conference on networking; 2012. p. 184–97.
Jerschow YI, Scheuermann B, Mauve M. Counter-flooding: DoS protection for public key handshakes in LANs. In: ICNS 2009: proceedings of the 5th International Conference on Networking and Services; 2009. p. 376–82.
Juels A, Brainard JG. Client puzzles: a cryptographic countermeasure against connection depletion attacks. In: NDSS '99: proceedings of the Network and Distributed System Security Symposium; 1999.
Karame GO, Čapkun S. Low-cost client puzzles based on modular exponentiation. In: ESORICS 2010: proceedings of the 15th European Symposium on Research in Computer Security; 2010. p. 679–97.
Lehmer DH. Computer technology applied to the theory of numbers. In: Studies in number theory. Englewood Cliffs, NJ: Prentice Hall; 1969. p. 117–51.
Mao W. Timed-release cryptography. In: SAC 2001: proceedings of the 8th annual international workshop on Selected Areas in Cryptography; 2001. p. 342–57.
Martinovic I, Zdarsky FA, Wilhelm M, Wegmann C, Schmitt JB. Wireless client puzzles in IEEE 802.11 networks: security by wireless. In: WiSec '08: proceedings of the ACM conference on Wireless Network Security; 2008. p. 36–45.
Menezes AJ, van Oorschot PC, Vanstone SA. Handbook of applied cryptography. CRC Press; 1996.
Mirkovic J, Reiher P. A taxonomy of DDoS attack and DDoS defense mechanisms. ACM SIGCOMM Computer Communication Review 2004;34(2):39–53.
Nishihara N, Harasawa R, Sueyoshi Y, Kudo A. A remark on the computation of cube roots in finite fields. Cryptology ePrint Archive, Report 2009/457, http://eprint.iacr.org/2009/457; 2009.
Peng T, Leckie C, Ramamohanarao K. Survey of network-based defense mechanisms countering the DoS and DDoS problems. ACM Computing Surveys 2007;39(1):3.
Rivest RL, Shamir A, Wagner DA. Time-lock puzzles and timed-release Crypto. Tech. rep.. Cambridge, MA, USA: Massachusetts Institute of Technology; 1996
Schaller P, Čapkun S, Basin D. BAP: broadcast authentication using cryptographic puzzles. In: ACNS '07: proceedings of the 5th international conference on Applied Cryptography and Network Security; 2007. p. 401–19.
Shanks D. Five number-theoretic algorithms. In: Proceedings of the 2nd Manitoba conference on numerical mathematics; 1972. p. 51–70.
Sorenson JP. A sublinear-time parallel algorithm for integer modular exponentiation. In: Proceedings of the conference on the mathematics of public-key cryptography; 1999. p. 528–38.
Suzuki D. How to maximize the potential of FPGA resources for modular exponentiation. In: CHES '07: proceedings of the 9th international workshop on Cryptographic Hardware and Embedded Systems; 2007. p. 272–88.
Szerwinski R, Güneysu T. Exploiting the power of GPUs for asymmetric cryptography. In: CHES '08: proceedings of the 10th international workshop on Cryptographic Hardware and Embedded Systems; 2008. p. 79–99.
Tang Q, Jeckmans A. On non-parallelizable deterministic client puzzle scheme with batch verification modes. Centre for Telematics and Information Technology, University of Twente, http://doc.utwente.nl/69557/; 2010.
Tonelli A. Bemerkung über die Auflösung quadratischer Congruenzen. Göttinger Nachrichten 1891:344–6.
Tritilanunt S, Boyd C, Foo E, Nieto JMG. Toward non-parallelizable client puzzles. In: CANS 2007: proceedings of the 6th international conference on Cryptology & Network Security; 2007. p. 247–64.
Feng W-c, Kaiser E, Feng W-c, Luu A. The design and implementation of network puzzles. In: INFOCOM 2005: proceedings of the 24th IEEE conference on computer communications; 2005. p. 2372–82.
Walfish M, Vutukuru M, Balakrishnan H, Karger D, Shenker S. DDoS defense by offense. In: SIGCOMM '06: proceedings of the 2006 conference on applications, technologies, architectures, and protocols for computer communications; 2006. p. 303–14.

Wang X, Reiter MK. A multi-layer framework for puzzle-based denial-of-service defense. International Journal of Information Security 2008;7:243–63.
Waters B, Juels A, Halderman JA, Felten EW. New client puzzle outsourcing techniques for DoS resistance. In: CCS '04: proceedings of the 11th ACM conference on Computer and Communications Security; 2004. p. 246–56.

Yves Igor Jerschow received the B. Sc., M. Sc., and Ph. D. degrees in Computer Science from the Heinrich Heine University, Düsseldorf, Germany, in 2005, 2007, and 2012 respectively. In late 2012 he joined the computer networking technology group at the University of Duisburg-Essen, Germany, as a postdoc. His current research interests include network security and cryptography with a focus on local area networks and Denial of Service (DoS) attacks.

Martin Mauve received the M. S. and Ph. D. degrees in Computer Science from the University of Mannheim, Germany, in 1997 and 2000 respectively. From 2000 to 2003, he was an Assistant Professor at the University of Mannheim. In 2003, he joined the Heinrich Heine University, Düsseldorf, Germany, as a Full Professor and Head of the research group for computer networks and communication systems. His research interests include distributed multimedia systems, multimedia transport protocols, mobile ad-hoc networks and inter-vehicle communication.