Queensland University of Technology

Efficient Modular Exponentiation-based Puzzles for Denial-of-Service Protection

Jothi Rangasamy, Douglas Stebila, Lakshmi Kuppusamy, Colin Boyd, and Juan González-Nieto

Information Security Institute
Queensland University of Technology, Brisbane, Queensland, Australia

Friday, December 2, 2011
ICISC 2011

Supported by Australia-India Strategic Research Fund project TA020002.

Summary

- A useful mechanism for protection from denial of service attacks is **client puzzles**, which are somewhat hard problems that require a certain amount of time to solve.
- Important properties include provable difficulty, non-parallelizability, deterministic solving time, and linear granularity.
- Generating puzzles and verifying solutions should be very inexpensive.
- We propose a **new RSA-based non-parallelizable client puzzle** that is up to 30 times faster for verification compared to previous non-parallelizable puzzles and much closer to the speed of hash-based puzzles.

Types of denial of service attacks

- **Brute force attacks**: attacker generates sufficiently many legitimate requests to overload a server's resources. Does not require special knowledge of protocol specification or implementation.
  - Distributed denial of service (DDoS) attacks
  - Ping floods
- **Semantic attacks**: attacker tries to exploit vulnerabilities of particular network protocols or applications. Requires special knowledge of protocol specification and implementation.
  - Buffer overflow attacks
  - TCP SYN flooding / IP spoofing attacks

Prevention techniques

Try to identify malicious traffic:

- address filtering to block false addresses or addresses making too many requests;
- bandwidth management by routers and switches;
- packet inspection: look for patterns of bad requests;
- intrusion-prevention systems: look for signatures of attacks.

Difficult to distinguish real users' legitimate requests from attacker's legitimately-formed requests in brute force attacks.

Gradual authentication

- Principle for denial-of-service resistance proposed by Meadows
- Idea is to use cheap and low-security authentication initially
- Gradually put more effort into authentication if earlier stages succeed
- A typical progression might be to implement cookies first, then puzzles, then strong cryptographic authentication.

- **Cookies** provide proof of reachability
- **Puzzles** provide proof of work
- **Signatures** provide strong cryptographic authentication

Puzzles

The server generates a challenge and the client is required to solve a moderately hard puzzle based on this challenge.

Puzzles should be:

- easy to generate,
- not require stored state,
- provably hard to solve, and
- easy to verify.

Puzzles may be either **computation-bound** or **memory-bound**. We only look at the former.

Puzzle definition

Formally, a client puzzle is a tuple of algorithms:

- Setup(1^k): Return public parameters and server secret *s*.
- GenPuz(*s*, *Q*, *str*): Generate a puzzle of difficulty *Q* for session string *str*.
- FindSoln(*str*, *puz*): Find a solution for session string *str* and the given puzzle *puz*.
- VerSoln(*s*, *str*, *puz*, *soln*): Check if *soln* is a valid solution for puzzle *puz* and session string *str*.

GenPuz and VerSoln should be inexpensive.

Puzzle security properties

- **Difficulty**: it should be moderately hard to solve a puzzle (computation-bound or memory-bound)

Puzzle security properties

- **Difficulty**: it should be moderately hard to solve a puzzle (computation-bound or memory-bound)
- **Unforgeability**: it should not be possible for the adversary to generate valid puzzles

Puzzle security properties

- **Difficulty**: it should be moderately hard to solve a puzzle (computation-bound or memory-bound)
- **Unforgeability**: it should not be possible for the adversary to generate valid puzzles
- **Non-parallelizability**: it should not be possible to have multiple computers solve a puzzle in less time than a single computer could

Puzzle security properties

- **Difficulty**: it should be moderately hard to solve a puzzle (computation-bound or memory-bound)
- **Unforgeability**: it should not be possible for the adversary to generate valid puzzles
- **Non-parallelizability**: it should not be possible to have multiple computers solve a puzzle in less time than a single computer could
- **Tuneable difficulty**: can provide puzzles with different difficulty levels, preferably with linear granularity

Puzzle security properties

- **Difficulty**: it should be moderately hard to solve a puzzle (computation-bound or memory-bound)
- **Unforgeability**: it should not be possible for the adversary to generate valid puzzles
- **Non-parallelizability**: it should not be possible to have multiple computers solve a puzzle in less time than a single computer could
- **Tuneable difficulty**: can provide puzzles with different difficulty levels, preferably with linear granularity
- **Useful puzzles**: the work done in solving a puzzle can be used for another purpose

Hash-based puzzle (Juels–Brainard)

Based on finding partial pre-image of hash function H.
Difficulty parameter is Q.

PuzGen
- Choose random $x \leftarrow \{0,1\}^k$
- Set $x = \underbrace{x'}_{Q} \parallel \underbrace{x''}_{k-Q}$
- Set $z = H(x, Q, \texttt{str})$
- Puzzle is (x'', z)

FindSoln Find y such that $H(y \parallel x'', Q, \texttt{str}) = z$

VerSoln Check that $z \stackrel{?}{=} H(y \parallel x'', Q, \texttt{str})$

Properties of hash-based puzzles

Merits

- Generation and verification very efficient
- Easily tuneable by giving 'hints' (range for solution)

Limitations

- Seem hard to make non-parallelisable
- Proofs of difficulty are only available in the random oracle model

Time-lock puzzles of Rivest–Shamir–Wagner (RSW)

- RSA-based puzzle proposed in 1996
- *Sending information into the future*
- Uses RSA modulus $n = pq$.
- Difficulty parameter is Q.

PuzGen
- Choose random a
- Puzzle consists of (n, a, Q)

FindSoln Compute $y = a^{2^Q} \bmod n$

VerSoln
- Compute $b = 2^Q \bmod \phi(n)$
- Check that $y \stackrel{?}{=} a^b \bmod n$

Properties of RSW puzzle

Merits

- Believed to be non-parallelisable - only known way to find y is to square a repeatedly Q times.
- Simple construction

Limitations

- Verification requires exponentiation
- No proof of difficulty

Karame–Čapkun puzzle (ESORICS 2010)

- RSW puzzle is relatively expensive to verify. VerSoln requires full modular exponentiation.
- Karame and Čapkun use *short RSA private exponent*. Consequently RSA public exponent must be very large.
- Puzzle is essentially to compute RSA encryption of random value.
- Verification is decryption with short exponent and checking.

Karame–Čapkun construction

n is RSA modulus, d is short RSA private exponent of length k (such as $k = 80$), public exponent is $e > n^2$.
Difficulty parameter is Q.

PuzGen
- Choose random X
- $K = e - (Q \bmod \phi(n))$
- Puzzle is (n, X, Q, K)

FindSoln Compute $y_1 = X^Q \bmod n$; $y_2 = X^K \bmod n$

VerSoln Check that $(y_1 y_2)^d \bmod n \stackrel{?}{=} X$

Properties of Karame–Čapkun construction

Merits

- Verification much improved over RSW puzzle, by about $|n|/2k$ times
- Has proof of difficulty (relative to RSW puzzle)

Limitations

- Verification still requires exponentiation
- Parallelisability not so tight

BPV Generator

- Boyko, Peinado, Venkatesan, Eurocrypt'98
- Method for efficiently computing random RSA encryptions efficiently with pre-computation.

Let k, ℓ, and N, with $N \geq \ell \geq 1$, be parameters. Let n be an RSA modulus and u an exponent.

- **Pre-processing** run once. Generate N random integers $\alpha_1, \alpha_2, \ldots, \alpha_N \leftarrow \mathbb{Z}_n^*$ and compute $\beta_i \leftarrow {\alpha_i}^u$ mod n for each i. Return a table $\tau \leftarrow ((\alpha_i, \beta_i))_{i=1}^N$.
- **Whenever a pair** $(x, x^u \bmod n)$ **is needed**: choose a random set $S \subseteq \{1, \ldots, N\}$ of size ℓ. Compute $x \leftarrow \prod_{j \in S} \alpha_j$ mod n and $X \leftarrow \prod_{j \in S} \beta_j$ mod n and return (x, X).

Statistical distance between this distribution and random is $2^{-\frac{1}{2}\left(\log \binom{N}{\ell} + 1\right)}$.

A new non-parallelisable puzzle (RSA Puz)

n is RSA modulus, public exponent is $e = 3$.
Difficulty parameter is Q.

Setup
- Set $d = 3^{-1} \bmod \phi(n)$
- Set $u = d - (2^Q \bmod \phi(n))$
- Compute BPV pre-processing to obtain table with $N = 2500$ and $\ell = 4$ (gives distance 2^{-20}).

PuzGen
- Use BPV algorithm to computer new $(x, X = x^u)$ pair
- Puzzle is (n, x, Q)

FindSoln Compute $y = x^{2^Q} \bmod n$

VerSoln Check that $(X \cdot y)^3 \bmod n \stackrel{?}{=} x$

Properties of RSA Puz

Merits

- Verification only requires a few multiplications
- Non-parallelisable
- Has proof of difficulty (relative to RSW puzzle) in Chen et al. model (ASIACRYPT 2009)

Limitations

- Preprocessing can be somewhat costly

Sample timings

Puzzle	512-bit modulus, $k = 56$			
	Setup (ms)	GenPuz (μs)	FindSoln (s)	VerSoln (μs)
Difficulty: $Q = 1$ million				
RSW puz	13.92	4.80	1.54	474.68
KC puz	11.52	8.37	1.59	263.35
RSA puz	1401.14	16.66	1.54	**14.75**
Difficulty: $Q = 10$ million				
RSW puz	49.99	4.80	15.17	474.83
KC puz	28.95	8.37	15.18	265.28
RSA puz	1419.78	16.66	15.34	**14.53**
Difficulty: $Q = 100$ million				
RSW puz	416.29	4.81	157.10	470.61
KC puz	218.76	8.35	160.97	259.39
RSA puz	1609.83	16.76	158.22	**14.88**

A typical hash-based puzzle has GenPuz $= 5.92\,\mu$s and VerSoln $= 3.77\,\mu$s.

Efficient Modular Exponentiation-based Puzzles for Denial-of-Service Protection

Jothi Rangasamy, Douglas Stebila,
Lakshmi Kuppusamy, Colin Boyd,
and Juan González-Nieto
stebila@qut.edu.au

Queensland University of Technology

- A useful mechanism for protection from denial of service attacks is **client puzzles**, which are somewhat hard problems that require a certain amount of time to solve.
- Important properties include provable difficulty, non-parallelizability, deterministic solving time, and linear granularity.
- Generating puzzles and verifying solutions should be very inexpensive.
- We propose a **new RSA-based non-parallelizable client puzzle** that is up to 30 times faster for verification compared to previous non-parallelizable puzzles and much closer to the speed of hash-based puzzles.