Queensland University of Technology

Stronger Difficulty Notions for Client Puzzles and Denial-of-Service-Resistant Protocols

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

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

Thursday, February 17, 2011

Session ID: CRYP-304
Session Classification: Advanced

Summary

Denial of service attacks aim to overload servers and disrupt access to resources.

Client puzzles aim to defend against denial of service attacks by requiring a client to solve a moderately hard problem before being granted access to a resource.

We describe weaknesses in existing definitions of puzzle difficulty and give improved definitions – in the interactive and non-interactive settings – more appropriate to powerful adversaries.

We also provide a generic way to use puzzles to protect any cryptographic protocol from denial of service attacks.

Background

1. What are denial of service attacks?
2. How can we defend against them?

Cyber attacks

- **Estonia** (April 2007)
- **Georgia** (August 2008)
- **United States and South Korea** (July 2009)
- **Mastercard, Visa** (December 2010)

Cyber attacks

- **Estonia** (April 2007)
- **Georgia** (August 2008)
- **United States and South Korea** (July 2009)
- **Mastercard, Visa** (December 2010)

- **Google** (June 2009): News searches sparked by Michael Jackson's death were initially mistook for an automated DoS attack.

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

Basic idea: as a server builds up more confidence in the client, it is willing to commit more resources.

Cookies
(reachability)
↓
Puzzles
(proof of work)
↓
Digital signatures
(strong authentication)

Listed in order of:

- increasing confidence in client
- increasing cost to server

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 security properties in the literature

- **Difficulty**: it should be moderately hard to solve a puzzle (computation-bound or memory-bound).
- **Unforgeability**: it should not be possible 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.
- **Useful puzzles**: the work done in solving a puzzle can be used for another purpose.

Modelling puzzle difficulty

1. Weaknesses in existing definitions
2. Stronger definitions: interactive and non-interactive
3. Examples

Puzzle definition

Formally, a client puzzle is a tuple of algorithms:

- **Setup**(1^k): Return public parameters and server secret s.
- **GenPuz**(s, d, str): Generate a puzzle of difficulty d 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 very inexpensive.

SPuz: puzzle based on Juels–Brainard[1] construction

Client | **Server**

$\xrightarrow{\texttt{Req}}$

Choose random $x \leftarrow \{0,1\}^k$

$x = \underbrace{x'}_{d} \parallel \underbrace{x''}_{k-d}$

$y = H(x, d, \texttt{str})$

$\xleftarrow{x'', y}$

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

$\xrightarrow{\texttt{str}, x'', y, z}$

$y \stackrel{?}{=} H(z \parallel x'', d, \texttt{str})$

[1] Juels and Brainard. A cryptographic countermeasure against connection depletion attacks. *NDSS 1999*.

Puzzle difficulty: the Bristol definition[2]

- Experiment parameters: puzzle difficulty d, security parameter k, and puzzle scheme P.
- Adversary interacts with a challenger which runs Setup(1^k) and provides access to two oracles:
  - **CreatePuzSoln**(str): Set $puz \leftarrow \text{GenPuz}(s, d, str)$ and find a valid solution *soln* for *puz*. Return (*puz, soln*).
  - **Test**(str^*): Return $puz^* \leftarrow \text{GenPuz}(s, d, str^*)$. Only a single Test query is allowed.
- **Goal:** output $soln^*$ such that $\text{VerSoln}(puz^*, soln^*)$ is true.

[2] Chen, Morrissey, Smart, Warinschi. Security notions and generic constructions for client puzzles. *ASIACRYPT 2009*.

Puzzle difficulty: the Bristol definition

A client puzzle scheme is said to be $\epsilon_{k,d}()$-**difficult** if

$$\Pr(\text{A wins}) \leq \epsilon_{k,d}(t)$$

for all probabilistic algorithms A running in time at most t, where $\epsilon_{k,d}(t)$ is a family of functions monotonically increasing in t.

- Example: might have $\epsilon_{k,d}(t) = t/d + negl(k)$.
- Why monotonically increasing? Should be impossible to solve a puzzle more easily by taking less time.

Limitation in Bristol definition

- Does not address the ability of powerful adversaries to solve multiple puzzles.
- We might choose puzzle difficulty 2^{20} operations, because we want a puzzle that takes a couple of seconds to solve on a modern CPU. There are definitely adversaries that have more power than that, so the puzzle difficulty experiment says nothing about them.
- **Counterexample:** Can construct examples based on signatures using composite modulus. Puzzle solution is a signature forgery.
  - Forging one signature can be easier than factoring the modulus and so Bristol definition is satisfied.
  - Forging 2^{20} signatures may only take the effort of forging, say, 2^{10} signatures by factoring the modulus and then using the trapdoor.

Strong puzzle difficulty

- We introduce new security experiments to address this weakness (and provide additional functionality).
- Quantify the ability of an adversary to return multiple solutions, not just one.
- The adversary can return solutions $(str, puz, soln)$ where it queried (str, puz) to the puzzle solving oracle, provided $soln$ was not the given solution
- Adversary has access to separate oracles for puzzle generation and puzzle solving.

Interactive strong puzzle difficulty

Goal: output a list of n tuples $(str_i, puz_i, soln_i)$ such that

1. $\text{VerSoln}(s, str_i, puz_i, soln_i)$ is true,
2. (str_i, puz_i) was generated by the puzzle generation oracle, and
3. $soln_i$ was not the response of any puzzle solution query for (str_i, puz_i)

A client puzzle scheme is said to be $\epsilon_{k,d,n}()$-**interactive-strongly-difficult** if

$$\Pr(\text{A wins}) \leq \epsilon_{k,d,n}(t)$$

for all probabilistic algorithms A running in time at most t, where

$$\epsilon_{k,d,n}(t) \leq \epsilon_{k,d,1}(t/n)$$

for all t, n such that $\epsilon_{k,d,n}(t) \leq 1$.

Non-interactive strong puzzle difficulty

Goal: output a list of n tuples $(str_i, puz_i, soln_i)$ such that

1. $\text{VerSoln}(s, str_i, puz_i, soln_i)$ is true
2. $soln_i$ was not the response of any puzzle solution query for (str_i, puz_i)

Main difference: don't require puz_i to be generated by the puzzle generation oracle. This accommodates **client-generated puzzles** (useful in asynchronous settings or limited communication rounds).

Non-interactive puzzles are more general than interactive puzzles. But we have two separate definitions because interactive puzzles can be easier to work with in protocols. Interactive puzzles can also protect against puzzle "herding" attacks by changing puzzle validity periods.

Examples

SPuz is a strongly-difficult interactive puzzle:

- **Theorem.** Let H be a random oracle. Let $\epsilon_{k,d,n}(q) = \left(\frac{q+n}{n2^d}\right)^n$. Then SPuz with H is an $\epsilon_{k,d,n}(q)$-interactive-strongly-difficult client puzzle, where q is the number of distinct queries to H.

Hashcash[3] is a strongly-difficult non-interactive puzzle:

- **Theorem.** Let H be a random oracle. Let $\epsilon_{k,d,n}(q) = \frac{q+n}{n2^d}$. Then Hashcash is an $\epsilon_{k,d,n}(q)$-non-interactive-strongly-difficult client puzzle, where q is the number of distinct queries to H.

Proofs are based on counting number of queries to H.

[3] Back. Hashcash. Online, 1997, 2004. http://www.hashcash.org/

DoS-resistant protocols

How should we use puzzles in protocols?

Goal: protect an existing protocol from denial of service attacks by using client puzzles for gradual authentication.

Not linking DoS countermeasure (cookies or puzzles) can cause authentication failure leading to DoS attacks (e.g., in IKEv2[4]).

- Hashcash: primarily for email; use client-generated puzzles with email address as part of puzzle.
- Bristol approach: authenticate generated puzzles; no link between the client puzzle protocol and the subsequent protocol.
- Stebila and Ustaoglu[5]: only for key agreement; use session identifier as part of puzzle.

Previous approaches also don't address adversaries who can solve more than 1 puzzle.

[4] Mao and Paterson. On the plausible deniability feature of Internet protocols. Online, 2002.

[5] Stebila and Ustaoglu. Towards denial-of-service-resilient key agreement protocols. *ACISP 2010*.

Defining DoS-resistant protocols

- Adversary controls communication between all parties.
- Adversary can gain server secret information via Expose query.
- Adversary can get clients to solve puzzles.
- The probability that an efficient adversary can make the server accept n puzzle instances should be bounded by a non-decreasing function $\epsilon_{k,n}(t)$ where $\epsilon_{k,n}(t) \leq \epsilon_{k,1}(t/n)$.
- Server should not perform expensive operations in a protocol run until puzzle is solved.

Generic construction using client puzzles

- Easiest way to protect a cryptographic protocol using client puzzles is to prepend the protocol run with a client puzzle run, and only run the main protocol once the puzzle is accepted.
- Let P be a protocol, Puz be a puzzle, and let $D(P, Puz)$ be the protocol in which each run of P is prepended by a run of Puz, protected by a MAC keyed by a server secret.

Theorem. If Puz is a strongly difficult puzzle, then $D(P, Puz)$ is a denial-of-service-resistant protocol.

If the first round of P involves expensive operations, then this adds an extra round; otherwise, we may be able to combine message flows.

Simple pre-session construction

Client — **Server** (secret ρ)

Choose random N_C

$\xrightarrow{N_C}$

Choose random $x \leftarrow \{0,1\}^k$

$x = \underbrace{x'}_{d} \,||\, \underbrace{x''}_{k-d}$

$\mathtt{str} = (C, S, N_C, N_S)$

$y = H(x, d, \mathtt{str})$

$\boxed{\sigma = MAC_\rho(\mathtt{str}, x'', y)}$

$\xleftarrow{N_S, x'', y, \boxed{\sigma}}$

Find z such that
$H(z \parallel x'', d, \mathtt{str}) = y$

$\xrightarrow{\mathtt{str}, x'', y, z, \boxed{\sigma}}$

Check for replay

Verify MAC σ

$y \stackrel{?}{=} H(z \parallel x'', d, \mathtt{str})$

Queensland University of Technology

Stronger Difficulty Notions for Client Puzzles and Denial-of-Service-Resistant Protocols

Douglas Stebila, Lakshmi Kuppusamy, Jothi Rangasamy, Colin Boyd, and Juan Gonzalez Nieto
Information Security Institute, Queensland University of Technology, Brisbane, Queensland, Australia
Email: stebila@qut.edu.au

- Provided new stronger definitions for puzzle difficulty, in interactive and non-interactive settings.
- Showed existence of efficient puzzles satisfying the new definitions.
- Showed how to apply puzzles in a generic way to achieve DoS-resistant protocols.

Supported by the Australia-India Strategic Research Fund project TA02002.