Toward Non-Parallelizable Client Puzzles
CANS 2007

Colin Boyd

Information Security Institute
Queensland University of Technology

Joint work with Suratose Tritilanunt, Ernest Foo and Juan Manuel González Nieto
Outline

Client puzzles and their properties

Previous puzzles

New puzzle
Client puzzles

- Cryptographic puzzles first introduced by Dwork and Naor (1992)
- A client puzzle is a ‘moderately difficult’ computational problem that the client is forced to solve before a server will provide service
- When a client initially requests service the server will construct a puzzle instance
- The client solves the puzzle and sends the solution to the server
- The server checks the solution and provides service if the solution is correct.
Client puzzle applications

- Proposed by Dwork and Naor for combating junk emails
- Rivest, Shamir and Wagner (1996) used puzzles for ‘time-release cryptography’
- Juels and Brainard (1999) considered puzzles to mitigate Denial-of-Service (DoS) attacks in network protocols
- Moskowitz (2007) developed the host identity protocol (HIP), which employs a client puzzle mechanism for protecting the server against resource exhaustion attacks
Client puzzle applications

- Proposed by Dwork and Naor for combating junk emails
- Rivest, Shamir and Wagner (1996) used puzzles for ‘time-release cryptography’
- Juels and Brainard (1999) considered puzzles to mitigate Denial-of-Service (DoS) attacks in network protocols
- Moskowitz (2007) developed the host identity protocol (HIP), which employs a client puzzle mechanism for protecting the server against resource exhaustion attacks
Client puzzle applications

- Proposed by Dwork and Naor for combating junk emails
- Rivest, Shamir and Wagner (1996) used puzzles for ‘time-release cryptography’
- Juels and Brainard (1999) considered puzzles to mitigate Denial-of-Service (DoS) attacks in network protocols
- Moskowitz (2007) developed the host identity protocol (HIP), which employs a client puzzle mechanism for protecting the server against resource exhaustion attacks
Client puzzle applications

- Proposed by Dwork and Naor for combating junk emails
- Rivest, Shamir and Wagner (1996) used puzzles for ‘time-release cryptography’
- Juels and Brainard (1999) considered puzzles to mitigate Denial-of-Service (DoS) attacks in network protocols
- Moskowitz (2007) developed the host identity protocol (HIP), which employs a client puzzle mechanism for protecting the server against resource exhaustion attacks
Aims of the paper

- Survey client puzzles for DoS-resistance
- Compare strengths and weaknesses of existing client puzzle techniques
- Look for client puzzle satisfying all known desirable properties
### Puzzle properties (Aura et al.)

1. Should be easy and cheap to construct and verify for the server
2. Cost of solving should be easy to adjust
3. Can be solved on most platforms
4. Client should be unable to pre-compute solution
5. Server should not need to store the solution
6. Knowing the solution of one puzzle should not help solve a new puzzle
7. Client can reuse puzzles by creating several instances of them
The importance of non-parallelizability

Many authors ignore non-parallelizability as a significant property. Some potential benefits are:

• it delays initial impact of DDoS attack
• it reduces flexibility of attacker strategies
• it can be useful for other applications such as time-lock puzzles

The motivation for this paper relies on the assumption that non-parallelizability is a useful property.
Aura, Nikander, Leiwo puzzles

Construction

Choose $k$ and $N_R$

Solving

Given $k$ and $N_R$ find $x$ such that $h(x \parallel N_R)$ produces output with first $k$ bits all zero.

Verification

Given $x$ check that first $k$ bits of $h(x \parallel N_R)$ are zero.
Aura, Nikander, Leiwo puzzles

**Construction**

Choose $k$ and $N_R$

**Solving**

Given $k$ and $N_R$ find $x$ such that $h(x \parallel N_R)$ produces output with first $k$ bits all zero.

**Verification**

Given $x$ check that first $k$ bits of $h(x \parallel N_R)$ are zero.
Aura, Nikander, Leiwo puzzles

**Construction**

Choose $k$ and $N_R$

**Solving**

Given $k$ and $N_R$ find $x$ such that $h(x \parallel N_R)$ produces output with first $k$ bits all zero.

**Verification**

Given $x$ check that first $k$ bits of $h(x \parallel N_R)$ are zero.
Some puzzles and their properties

- **Hash-based reversal**
  - Pro: Simple and fast
  - Con: Parallelizable

- **Repeated squaring** (Rivest–Shamir–Wagner 1996)
  - Pro: Non-parallelizable
  - Con: Expensive construction and verification

- **Trapdoor RSA or DH** (Gao et al. 2005)
  - Pro: Non-parallelizable
  - Con: Expensive construction and parallelizable
Hash chain puzzles

- Obvious way to try to make hash-reversal into a non-parallelizable puzzle
- Each link in chain needs to be solved before next one is tackled
- Constraints are difficult to resolve:
  - Must be many links in the chain
  - Each link must be relatively easy to solve
- Two proposals to use hash chain in the literature; both have similar limitations
Ma’s (2005) construction

- **Puzzle generation**: Choose random $r$ and compute
  \[ X = h^k(r) = h(h(h(r))) \]
- **Puzzle solution**: Given $X$ and $k$ compute $h^j(r)$ for
  \[ k - 1 \geq j \geq 0 \]

**Limitations**

1. Requires server to store every value of entire hash chain for verification
2. Each step must be ‘easy’. Ma suggested to use 16-bit outputs of hash, but then adversary can invert $h$ in a look-up table
Subset sum problems

- A given set of items each have specified weight
- The knapsack can carry any number of items totalling to no more than a certain weight
- The solver is required to search for a maximum value by picking as many items as the knapsack can carry in terms of weight
- Various knapsack cryptosystems have been proposed but most have been broken.
• A famous tool used to successfully break subset sum cryptosystems is *lattice reduction*.

• The best method so far for breaking the subset sum problems is the LLL (or $L^3$) algorithm developed by Lenstra et al. in 1982.

• LLL is a polynomial time for finding a reduced basis for a lattice.

• It is non-parallelizable because it requires highly sequential computation on an iterative function
Subset sum solving algorithms

- **Backtracking or Brute Force Searching** Exhaustive search that gathers all possible solutions and then checks for one satisfying the solution
- **Branch and Bounding Technique** Avoiding some unnecessary nodes during the searching process by storing and traveling only to states whose total weight does not exceed the limit
- **LLL Lattice Reduction** Solve the puzzle within polynomial time $O(n^3 \log n)$ rather than exponential time as the two other techniques
Subset sum puzzle: precomputation

- Choose random first weight $w_1$.
- Compute $w_k = h(w_{k-1})$ for $2 \leq i \leq k$
- All weights are stored by server and can be computed by clients given $w_1$.
- Secret value $s$ is chosen. This value can change periodically.
Subset sum puzzle instance

- **Puzzle construction:** (Server)
  1. Nonces $N_i$ chosen by client, $N_R$ chosen by server.
  2. Choose puzzle difficulty $k$; $25 \leq k \leq 100$
  3. Set $C = h(ID_i, N_i, ID_R, N_R, s)$ ($k$ least significant bits)
  4. Compute $W = \sum_{i=1}^{k} C_i w_i$

  **Puzzle** is the tuple $(w_1, W, k)$

- **Puzzle solution:** (Client)
  1. Recompute all weights from $w_1$
  2. Find vector $C'$ with $W = \sum_{i=1}^{k} C'_i w_i$
  3. Solution is returned with nonces and client ID

- **Verification:** (Server)
  1. Re-compute $C$ and $W = \sum_{i=1}^{k} C_i w_i$
  2. Compute $W = \sum_{i=1}^{k} C'_i w_i$
  3. Check $C' = C$
Subset sum puzzle instance

- **Puzzle construction:** (Server)
  1. Nonces $N_I$ chosen by client, $N_R$ chosen by server.
  2. Choose puzzle difficulty $k$; $25 \leq k \leq 100$
  3. Set $C = h(ID_I, N_I, ID_R, N_R, s)$ ($k$ least significant bits)
  4. Compute $W = \sum_{i=1}^{k} C_i w_i$

  **Puzzle** is the tuple $(w_1, W, k)$

- **Puzzle solution:** (Client)
  1. Recompute all weights from $w_1$
  2. Find vector $C'$ with $W = \sum_{i=1}^{k} C'_i w_i$
  3. Solution is returned with nonces and client ID

- **Verification:** (Server)
  1. Re-compute $C$ and $W = \sum_{i=1}^{k} C_i w_i$
  2. Compute $W = \sum_{i=1}^{k} C'_i w_i$
  3. Check $C' = C$
Subset sum puzzle instance

• **Puzzle construction**: (Server)
  1. Nonces $N_I$ chosen by client, $N_R$ chosen by server.
  2. Choose puzzle difficulty $k$; $25 \leq k \leq 100$
  3. Set $C = h(ID_I, N_I, ID_R, N_R, s)$ ($k$ least significant bits)
  4. Compute $W = \sum_{i=1}^{k} C_i w_i$

  **Puzzle** is the tuple $(w_1, W, k)$

• **Puzzle solution**: (Client)
  1. Recompute all weights from $w_1$
  2. Find vector $C'$ with $W = \sum_{i=1}^{k} C'_i w_i$
  3. Solution is returned with nonces and client ID

• **Verification**: (Server)
  1. Re-compute $C$ and $W = \sum_{i=1}^{k} C_i w_i$
  2. Compute $W = \sum_{i=1}^{k} C'_i w_i$
  3. Check $C' = C$
Properties of new construction

- **Construction cost:** 1 hash plus 1 summation
- **Verification cost:** 1 hash plus 2 summations
- **Puzzle size:** No more than 400 bits
- **Non-parallelizable:** LLL is much faster than brute force for reasonable parameter sizes
Experimental results

Practical experiments indicate the practical feasibility of the proposed puzzle.

- When $k \geq 25$ brute force (parallelizable) attacks take at least 10 000 times the computation of LLL.
- Puzzle difficulty between a fraction of a second and several thousand seconds.
- Hard to make puzzle difficulty very precise.
Conclusion

• Have given a plausible construction combining all desirable properties including non-parallelizability

• Some practical limitations:
  1. need to implement LLL in clients
  2. large memory requirement for higher puzzle difficulty

• Is a more practical puzzle possible which satisfies all properties?