Performance Evaluation of Non-parallelizable Client Puzzles for Defeating DoS Attacks in Authentication Protocols

Suratose Tritilanunt

Performance Evaluation of Non-Parallelizable Client Puzzles for Defeating DoS Attacks in Authentication Protocols

Suratose Tritilanunt

Computer Engineering Department, Faculty of Engineering, Mahidol University
25/25, Salaya, Phuttamonthol, Nakornpathom, Thailand, 73170
egstl@mahidol.ac.th

Abstract. We provides an evaluation of non-parallelizable puzzles used to prevent DoS in authentication protocols. With an evaluation based on a simulation and performance analysis, this approach helps a responder to resist against DoS, as well as improves the throughput of services for legitimate clients. Another key strength is that the construction and verification at the responder is simple and fast.

1 Introduction

Client puzzles in computer network was first introduced by Dwork and Naor [1] for combating junk emails. Almost a decade, Juels and Brainard [2] adopted this technique to defeat denial-of-service (DoS) attacks in network protocols. Later on, many techniques have been proposed for constructing client puzzles, for examples, Hash-based Reversal Puzzles [2–4], Time-Lock Puzzles [5], and Diffie-Hellman based Puzzles [6].

Hash-based constructions meet many of the desirable properties of proofs of work [7], but they also have the property that exhaustive searching of a pre-image search space is a parallelizable task. Using such a technique in the presence of an adversary with access to distributed computing resources may leave authentication protocols exposed to DoS. Adopting alternate puzzle constructions, such as time lock puzzles that are inherently sequential and non-parallelizable, may need to be considered for protocols that are to be used in an environment where the adversarial model assumes that significant resources are available to the attacker.

A client puzzle is non-parallelizable if the solution to the puzzle cannot be computed in parallel. Non-parallelizable client puzzles can be used to defend against distributed denial-of-service (DDoS) attacks, where a single adversary can control a large group of compromised machines. This adversary could distribute puzzles to other compromised machines to obtain puzzle solutions faster than the time expected by the server. This kind of attack is identified as strong attacks [8].

Two examples of a puzzle construction which was implemented for preventing strong attacks are a *hash chain* [9,10], and a *modified repeated squaring* technique

[11]. Because a nature of chaining requires a previous value for constructing the next consecutive items, the construction of hash chain can prevent parallel searching. In a repeated squaring puzzles, the developer improves a modular arithmetic calculation of Time-Lock Puzzles [5] to achieves a fast verification and a non-parallizable feature.

To address the problem of parallelizable client puzzles, this paper proposes a construction having characteristics comparable to time-lock and hash chain puzzles but the new scheme requires less computation in the puzzle construction and verification. Our new puzzle scheme including puzzle construction, puzzle solving, and puzzle verification, as well as the experimental results which are examined based on the performance analysis using CPN Tools are provided in this paper.

2 Non-parallelizable Puzzles based on Subset Sum

Apart from a brute-force searching (that requires a running time of order $\mathcal{O}(2^n n)$, where n represents the number of decision variables) used to solve subset sum problems, an alternative technique used to successfully break subset sum problems is called a lattice basis reduction. There are several lattice reduction algorithms but the best method so far for breaking the subset sum problems is the *LLL* or L^3 algorithm developed by Lenstra et al. [12]. LLL algorithm has been widely used in breaking subset sum cryptosystems [13,14] because the algorithm is able to terminate in polynomial time. Moreover, it is highly sequential because the underlying program requires recursive computation. From this perspective, LLL is a promising technique to fulfill our requirement in terms of non-parallelizability and thwart coordinated adversaries from distributing the client puzzle to calculate the solution in a parallel manner. Details of L^3 lattice basis reduction is beyond our scope of this paper, so we encourage the reader interested in more detail to read the papers by Nguyen and Stern [14], and Joux and Stern [13].

2.1 Puzzle Construction, Solving, and Verification

To establish a secure connection to a responder R, I sends a request containing an identity (ID_I) along with a random nonce (N_I). The responder chooses a secret parameter s randomly to make the output unique for each communication, and decides a puzzle difficulty k depending on the workload condition. The value of k should be selected to be at least 25 in order to guarantee that the coordinated adversary requires over a thousand compromised machines to brute-force search or over a hundred compromised machines to run bounding algorithm on the subset sum puzzles at the equivalent time to the legitimate user performing LLL lattice reduction. As a practical choice we suggest to take a value of k between 25 and 100 and then if weights are chosen to be of length 200 bits we can ensure that the generated knapsack has density at most 0.5. Practical experimental tests can be found in [15] which support our proposal.

I		R
		Precomputed parameters set of random weight w_n $w_n = H(w_{n-1})$ ....
1) send *request*	$\xrightarrow{ID_I, N_I}$	choose *secret* $\boldsymbol{s} \in_R \mathcal{Z}_n$ choose *puzzle difficulty* $\boldsymbol{k} \rightarrow 25 \leq \boldsymbol{k} \leq 100$ $C = LSB(H(ID_I, N_I, ID_R, N_R, \boldsymbol{s})), \boldsymbol{k})_2$ $W = \sum_{i=1}^{k} C_i \cdot w_i$ $puzzle = (w_1, W, k)$
2) verify ID_I, N_I generate $w_k = H(w_{k-1})$ form a Basis Set $\boldsymbol{B}$ run *LLL Reduction* $\rightarrow$ get C' check $W \overset{?}{=} \sum_{i=1}^{k} C'_i \cdot w_i$	$ID_I, N_I,$ $\xleftarrow{ID_R, N_R, puzzle}$	
3) return C'	$ID_I, N_I, ID_R,$ $\xrightarrow{N_R, puzzle, C'}$	option 1) check$C' \overset{?}{=} C$ option 2) re-generate$C$ $C = LSB(H(ID_I, N_I, ID_R, N_R, \boldsymbol{s})), \boldsymbol{k})_2$ check$C' \overset{?}{=} C$

Fig. 1. Subset Sum Puzzles

Figure 1 represents the *puzzle construction.* The responder R computes a hash operation $(H(\cdot))$, and computes $(LSB((\cdot), k)_2)$ to obtain k bits from the output of hash function. Finally, R forms a *puzzle* by computing a desired weight (W) that it wants a client to solve from a pre-computed set of random weight (w_n). To save on protocol bandwidth, weights can be generated given the initial random weight w_1 by iterative hashing. Hence, a puzzle contains an initial value of weight of the first item (w_1), a desired weight (W), and puzzle difficulty (k).

Considering the client's job for *solving a puzzle*, it begins to generate a series of random weights, $(w_1, w_2, \ldots, w_k)$, by computing a hash chain on an initial value w_1. Then, the client constructs a basis reduction set B as $b_1 = (1, 0, \ldots, 0, w_1)$, $b_2 = (0, 1, \ldots, 0, w_2)$, $b_k = (0, 0, \ldots, 1, w_k)$; and $b_{k+1} = (0, 0, \ldots, 0, -W)$. Finally, the client runs a LLL Basis Reduction [16] which is the most effective method to find moderately short lattice vectors in polynomial time. It is important to note that, the protocol does not limit the client to use LLL algorithm to solve the puzzles. However, using other techniques, such as brute-force search in traditional puzzles, might take an unreasonable interval to solve our scheme.

In terms of the puzzle granularity, there are two possible options for the responder to adjust the puzzle difficult; 1) adjusting the item size (n), or 2) adjusting the density (B). Both modifications affect the running time by a factor $(n^{\alpha} \cdot \log^{\beta} B)$, where α and β are real numbers dependent on the version of LLL basis reduction. Since the complexity of LLL basis reduction is a polynomial function, we conclude that our subset sum puzzles provide a polynomial granularity.

2.2 Comparison of Client Puzzle Properties

Based on the properties of good puzzles defined by Juels and Brainard [2]), only Repeated-squaring, Hash Chain, and Subset Sum puzzles can provide non-parallelization. Comparing our construction with the others, we find that both of them suffer from high computation at construction time which means that a responder using these puzzles would be susceptible to flooding attacks. Since our scheme has coarser granularity than Repeated-squaring and Hash Chain puzzles, this issue could be an interesting open problem for the research community to explore techniques providing both non-parallelization and linear granularity.

Table 1. Summary of Puzzles in term of Proposed Desirable Properties

Puzzle Type	Properties for Good Puzzles						
	Easy to Construct and Verify	Easy to Adjust	Not Require Specialised Client Hardware	Solution cannot be pre-computed	Server does not store solution	Non-parallelization	Granularity
Hash-based Reversal	√	√	√	√	√	X	Exponential
Hint-Based Hash Reversal	√	√	√	√	√	X	Linear
Repeated-Squaring	X	√	√	√	√	√	Linear
DH-based	X	√	√	√	X	X	Linear
Trapdoor RSA	X	√	√	√	X	X	Linear
Trapdoor DLP	X	√	√	√	X	X	Linear
Hash Chain	X	√	√	√	√	√	Linear
Subset Sum	√	√	√	√	√	√	Polynomial

3 Performance Analysis on Subset Sum Puzzles

By replacing a hash-based reversal scheme with our subset sum puzzles, we set up a formal time-based model using CPN Tools as our formalism.

3.1 Tolerance of a DoS-resistant Protocol

Evaluating tolerance of the server under DoS attacks is the major purpose of this experiment. We set up the experiment to measure tolerance of the server under two different workloads (Z); LOW for the light-load, and $HIGH$ for the heavy-load, from five types of adversaries as following

Type 1 adversary or ad1 computes a valid first message (may be pre-computed in practice), and takes no further action in the protocol.

Type 2 adversary or ad2 completes the protocol normally including searching a correct client puzzle solution C' until the third message is sent and takes no further action after this.

Type 3 adversary or ad3 searches for a correct client puzzle solution C' but randomly chooses the remaining message elements, then takes no further action in the protocol.

Coordinated Type 3 adversary or Co_ad3 is similar to Type 3 adversaries, except that Coordinated Type 3 adversaries are able to control a group of compromised machines to solve puzzles in parallel for obtaining the solution with a certain period.

Type 4 adversary or ad4 is like an adversary type 3, except that the client puzzle solution C' is now also chosen randomly.

Table 2 summarizes experimental results as the percentage of a number of successful legitimate requests that the responder can serve under different adversarys abilities. While the output from Type 2 and Type 3 adversaries shows a slight improvement, the most contrast comes from Coordinated Type 3 adversary. Obviously, this is because hash-based reveral client puzzles have not been designed to tolerate the parallel computation from Coordinated Type 3 adversary.

Table 2. Percentage of Throughput with Hash-based Reversal and Subset Sum Puzzles

Adversaries	**LOW**		**HIGH**	
	Hash-based Reversal	**Subset Sum**	**Hash-based Reversal**	**Subset Sum**
ad1	100	100	100	100
ad2	71.60	80.65	42.05	48.25
ad3	62.95	70.50	31.45	33.20
Co_ad3	18.50	71.50	4.95	35.80
ad4	87.20	99.95	83.20	87.45

3.2 Performance Analysis of Subset Sum Puzzles

To evaluate our mechanism, we apply a performance analysis to investigate our puzzles. By means of statistical analysis, we pay more attention to quantitative information about the performance including user processing time compared to server processing time, queue delay on the server at request messages and puzzle verification, as well as number of rejected packets of legitimate users. Table 3 represents our experimental result.

1) ad_Processing Time: This information represents how much computation is spent in the attack in comparison with the responder to defend such

Table 3. Performance of Adjustable Subset Sum Client Puzzles

Performance Factors	**Adjustable Subset Sum Puzzles**									
	ad1		**ad2**		**ad3**		**Co_ad3**		**ad4**	
	LOW	**HIGH**	**LOW**	**HIGH**	**LOW**	**HIGH**	**LOW**	**HIGH**	**LOW**	**HIGH**
ad_Processing Time	50	500	75200	2645147.06	40638	2006222.03	150	1500	100	1000
responder Processing Time	42835	176260	69017	216540	103991.06	266340.90	107611.54	281906.65	42885	171704.86
Time Out at MSG1	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
Max Delay at MSG1	69	519	69	519	69	519	99.65	523.90	69	519
Max Delay at MSG3	0	0	1481	1494	1858.14	1716.35	2456.12	2265.87	0	1354.50

attacks as quantitative measurement in cost-based analysis of Meadows' proposal [17]. From Table 3, `ad2` takes the longest time because `ad2` computes puzzle solving and signature generation, while `ad1` spends less computation to mount DoS attack since they only create and flood bogus requests at step 1. However if we compare the result to Table 2, both `ad1` and `ad2` do not achieve their DoS attacks. From this point of view, other factors should be combined in the evaluation.

2) responder Processing Time: It has been used as a cost factor to compare with the adversaries' processing time for estimating the effect of DoS attacks. As shown in Table 3, the responder wastes maximum computation to `ad3` and `Co_ad3`, while spends lesser computation for `ad1`, `ad2`, and `ad4` approximately. This is because the responder can detect the attacks from `ad4` and disregard them very quickly. Considering the former case, `ad2` does not cause much destruction in comparison with `ad3` and `Co_ad3` because `ad2` requires to compute both puzzles and digital signature. Unlike `ad2`, `ad3` and `Co_ad3` do not compute the digital signature, so their bogus messages arrive to the responder quickly and those bogus packets have longer period to stay in the queue before puzzles expire. Although the responder is able to detect the attacks at signature verification, it is too late for serving legitimate users since the signature verification is an expensive operation which requires plenty of time.

3) Time Out at MSG1: It provides information regarding to how effective are flooding attacks from `ad1`, which is the most common and easiest DoS technique. Since most authentication protocols nowadays implement stateless connection and cookies to thwart TCP SYN flooding attacks, it is more difficult for adversaries to mount the attacks using this simple techniques. This factor also refers to the efficiency of the puzzle generation of the responder in order to deal with large numbers of flooding attacks. As shown in the table, there are no rejected messages at this state for any attacking strategies because our puzzle generation is very fast. The puzzle can therefore be a powerful defending approach as a first line of defense when we combine with other DoS-resistant mechanisms.

4) Max Delay at MSG1 and MSG3: These two values show the maximum time delay of incoming packets in the queue at protocol step 1 and step 3 on the responder. The delay at step 1 indicates the efficiency of the responder to generate the client puzzles, while the delay at step 3 can be referred to the efficiency of the puzzle validation. Not surprisingly the longer delay in the queue at step 3 is, the more degradation of overall services in the system will be. The reason is because the jobs at step 3, which primarily consists of puzzle and signature verification, requires longer time to execute than the job at state 1. In addition, the delay at state 3 might cause the increment of rejected messages at step 1 if the accumulation on the incoming messages at step 3 is increasing at a high rate and keeps the responder busy processing these packets until requests at state 1 have reached or exceeded the maximum time-out period. From the performance result, only `ad3` and `Co_ad3` are able to boost up the delay on both states in our puzzles.

In summary, our subset sum puzzles function properly at least under five proposed attacking strategies. Particularly, they can prevent users from gaining advantages by searching valid puzzle solutions more quickly by parallel computation. Moreover, the performance of subset sum puzzle construction and puzzle generation functions effectively as shown in the performance analysis. This leads to the improvement of the tolerance under all defined denial-of-service techniques.

4 Conclusion

With regard to lacking of the parallelism characteristic in existing client puzzles, we proposes a new puzzle construction based on the subset sum problem. Undoubtedly, the primary strength over others is non-parallelization. In addition, the puzzle construction and verification requires simple and fast computation on the responder as shown in the performance analysis. Evaluation by using performance analysis under five performance parameters and the percentage of successful service shows that our new approach improves the throughput in comparison with hash-based reversal technique.

References

1. Dwork, C., Naor, M.: Pricing via Processing or Combatting Junk Mail. In: the 12th Annual International Cryptology Conference on Advances in Cryptology, Springer-Verlag (1992) Lecture Notes In Computer Science; Vol. 740.
2. Juels, A., Brainard, J.: Client Puzzles: A Cryptographic Defense Against Connection Depletion Attacks. In: the 1999 Network and Distributed System Security Symposium (NDSS '99), San Diego, California, USA (Feb 1999) 151–165
3. Aura, T., Nikander, P., Leiwo, J.: DoS-resistant authentication with client puzzles. In: Security Protocols Workshop 2000, Cambridge (2000) 170–181
4. Feng, W.: The case for TCP/IP Puzzles. In: ACM SIGCOMM 2003 Workshops, Karlsruhe, Germany, ACM Press (25-27 Aug 2003) 322–327
5. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock Puzzles and Timed-release Crypto. Technical Report TR-684, Massachusetts Institute of Technology, Cambridge, MA, USA (10 Mar 1996)

6. Waters, B., Juels, A., Halderman, J.A., Felten, E.W.: New Client Puzzle Outsourcing Techniques for DoS Resistance. In: the 11th ACM Conference on Computer and Communications Security (CCS 2004), USA, ACM Press (2004)
7. Jakobsson, M., Juels, A.: Proofs of work and bread pudding protocols. In: the IFIP TC6 and TC11 Joint Working Conference on Communications and Multimedia Security (CMS 99). (Sep 1999)
8. Bocan, V., Cosma, M.F.: Adaptive Threshold Puzzles. In: EUROCON 2005 - The International Conference on Computer as a tool, Belgrade, Serbia and Montenegro (Nov, 22-24 2005)
9. Ma, M.: Mitigating denial of service attacks with password puzzles. In: International Conference on Information Technology: Coding and Computing, 2005. (ITCC 2005). Volume 2. (2005) 621–626
10. Groza, B., Petrica, D.: On Chained Cryptographic Puzzles. In: 3rd Romanian-Hungarian Joint Symposium on Applied Computational Intelligence (SACI), Timisoara, Romania (May 25-26 2006)
11. Jeckmans, A.J.P.: Practical client puzzle from repeated squaring. Master's thesis (September 2009)
12. Lenstra, A.K., Jr., H.W.L., Lovász, L.: Factoring Polynomials with Rational Coefficients. Mathematische Annalen **261(4)** (Dec 1982) 515–534
13. Joux, A., Stern, J.: Lattice Reduction: A Toolbox for the Cryptanalyst. Journal of Cryptology: the journal of the International Association for Cryptologic Research **11**(3) (1998) 161–185
14. Nguyen, P.Q., Stern, J.: Lattice Reduction in Cryptology: An Update. In: ANTS-IV: Proceedings of the 4th International Symposium on Algorithmic Number Theory, London, UK, Springer-Verlag (2000) 85–112
15. Tritilanunt, S., Boyd, C., Foo, E., Nieto, J.M.G.: Toward Non-Parallelizable Client Puzzles. In: CANS'07: 6th International Conference on Cryptology & Network Security, Singapore (Dec 8 - 10 2007)
16. Coster, M.J., Joux, A., LaMacchia, B.A., Odlyzko, A.M., Schnorr, C., Stern, J.: Improved low-density subset sum algorithms. Computational Complexity **2**(2) (1992) 111–128
17. Meadows, C.: A Cost-Based Framework for Analysis of DoS in Networks. Journal of Computer Security **9(1/2)** (Jan 2001) 143–164