Advanced Materials Research *Online: 2013-11-21*
ISSN: 1662-8985, Vols. 846-847, pp 1615-1619
doi:10.4028/www.scientific.net/AMR.846-847.1615

C2Puzzle: A Novel Computational Client Puzzle for Network Security

Ming Wan, Wenli Shang, Jianming Zhao and Shengshan Zhang

Lab. of Networked Control Systems, Shenyang Institute of Automation Chinese Academy of Sciences, Shenyang, Liaoning 110016, China

ming305.bjtu@gmail.com, shangwl@sia.cn, zhaojianming@sia.cn, zhangshengshan@sia.cn

Keywords: computational client puzzle; prime factorization; discrete logarithm problem; network security.

Abstract. Client puzzles, a type of cryptographic puzzle, have been proposed and developed by many scholars to guarantee network security. In this paper, we proposed C2Puzzle, a novel computational client puzzle. By using the iterative algorithm, C2Puzzle successfully combined the basic principles of prime factorization and discrete logarithm problem. We show that, C2Puzzle has the visible security advantages, and the adjustability of puzzle difficulty is in fine granularity.

Introduction

Client puzzles, originating from cryptographic puzzles [1] used to hide the secret which can merely be uncovered after some computational effort, are extensively explored and used by many scholars to guarantee network security. For instance, various client puzzles have been proposed as a countermeasure to DoS (Denial of Service) attacks [2,3] or Sybil attacks [4]. In general, client puzzles are designed to supply the legitimate clients with service guarantees. For each service request, the client is forced to solve a client puzzle before the server has to commit its valuable resources. Of course, the computational effort for each puzzle is very moderate, whereas a large computational task would be imposed on the adversaries if they want to generate traffic in large quantities. In a word, the main idea of client puzzles is that by asking the client to allocate some of its own resources (CPU time or memory) the server establishes that the client is willing to invest in the valuable service.

Up still now, there are two main categories of client puzzles [5]. The first category which is larger is that of the CPU-bound puzzles [6,7]. Aiming at solving these puzzles, the moderate amount of CPU time has to be consumed by the clients. The second category is that of the memory-bound puzzles [8,9]. In this category, a number of memory look-ups have to be done in order to solve these puzzles. In this paper, we propose C2Puzzle, a novel computational client puzzle belonging to the category of CPU-bound puzzles. What is more, C2Puzzle is based on the basic principles of prime factorization [4] and discrete logarithm problem [10]. For one thing, prime factorization is hard and adjustable, but it is possible to pre-compute the solutions to the puzzles by the clients. For another, discrete logarithm problem based on One-way Trapdoor Function is one-way and irreversible, but it is no defense against parallel computation as the clients are given a range in which to search [11]. By using the iterative algorithm, C2Puzzle efficiently combines prime factorization with discrete logarithm problem to resolve the problems of pre-computation and parallel computation.

Design Details of C2Puzzle

Puzzle Generation Algorithm. Let $a_1 \cdots a_n$ be a series of primes, and the large number K is the product of these primes. The equation is as follows:

$$K = \prod_{i=1}^{n} a_i \ , \ i \in [1, n] \tag{1}$$

Let q be a large prime which may be published by the server, and select a random number r between 0 and $q-1$. x_1 which is the puzzle solution is then randomly chosen over the range

$[r,(r+l)\bmod(q-1)]$ or $[(r+l)\bmod(q-1),r]$, where l is a changeable number indicated by the server, and l can adjust the difficulty of the puzzle which will be depicted in next section.

Then, compute y_n by using the following equations:

$$\begin{cases} y_i = f(x_i) = i * a_i^{x_i} \bmod q \\ x_{i+1} = f(x_i) \end{cases}, \ i \in [1,n] \tag{2}$$

Here, we successfully adopt the iterative algorithm, and the base a_i of each iteration is a prime factor of K, and all of the prime factors of K are in ascending order, for instance, if $e \le f$, then $a_e \le a_f$, where $e, f \in [1,n]$. The iteration time i in Eq. 2 can guarantee that there is no periodic cycle in the sequence from x_1 to x_n. The reasons are as follows: if we only adopt the discrete logarithm equation $y_i = a_i^{x_i} \bmod q$, owning to the modular arithmetic it would probably occur the periodic cycle in the sequence from x_1 to x_n, namely $x_j = x_{j+c}$. Therefore, when a malicious attacker finds the law of periodic cycle in the iterative process, he or she can acquire a shortcut to avoid the massive computational expenses.

In addition, by virtue of introducing the modular arithmetic, several different numbers which can get the same y_n after Eq. 2 may exist in the range $[r,(r+l)\bmod(q-1)]$ or $[(r+l)\bmod(q-1),r]$. In order to resolve this problem, the server needs to compute the sum Sum as follows:

$$Sum = \sum_{i=1}^{n} x_i, \ i \in [1,n], \ x_n = y_n \tag{3}$$

At this stage, the puzzle generation algorithm is accomplished successfully, and the corresponding puzzle is the set $\{K, q, r, l, y_n, Sum\}$.

Puzzle Verification. When a client receives the puzzle challenge from the server, he or she firstly computes the prime factorization of K and gets all of the prime factors $a_1 \cdots a_n$. After that, according to Eq. 2 and Eq. 3, the client implements a brute force to search for a candidate solution x' from $[r,(r+l)\bmod(q-1)]$ or $[(r+l)\bmod(q-1),r]$, and computes y_n' and Sum'. If $y_n' = y_n$ and $Sum' = Sum$, he or she finds that x' is the puzzle solution. And then the client answers the puzzle solution x' to the server.

After the server receives the puzzle solution x', it only needs to compare x' with its own x_1, if consistent, then the server will provide the service for the client; and if inconsistent, then server will reject the client.

Discussion

Advantages of C2Puzzle. Compared with other countermeasures which not only need some modifications to the network protocols or infrastructures but also are nonviable to be deployed for economic or other factors, C2Puzzle is very practical and easy to accomplish. As was discussed before, it merely needs to install some small software on most types of client hardware for the puzzle computation. The advantages of C2Puzzle can be summarized in Table 1.

Table 1 The advantages of C2Puzzle

Advantage	Analytical Description
Stateless	There is no client's information in C2Puzzle, and all of the parameters in C2Puzzle are generated only by the server.
Pre-constructed	The server can pre-construct a lot of puzzles with different difficulty and store these puzzles in its database.
Computation Guarantee	The client firstly must compute prime factorization to acquire each prime a_i, and then carries out the iterations of modular exponentiation with the base a_i to search for the solution x_1.
Against Pre-computation	The malicious attackers cannot dope out the discrete logarithm problem

	because of the random number r and the changeable number l.
Against Parallel computation	The client must first accomplish the prime factorization of K, and then deals with discrete logarithm problem.
Easy to Verity	There is only one comparison to verify the solution.
Correlation-free	The large number K, the random number r and the changeable number l in each puzzle are different and unrelated.

Adjustability of Difficulty. This characteristic is referred to as puzzle granularity [11]. Adjustability of puzzle difficulty means the cost of solving the puzzle can be increased or decreased in fine granularity, and the difficulty of the puzzles should be adjusted flexibly according to the strength of an attack. The adjustability of difficulty in C2Puzzle can be profoundly manifested in the following four aspects: prime factorization, iteration time, search range and modulus.

We implement C2Puzzle in VC++ 6.0, and then we test the adjustability of difficulty by the four aspects mentioned above. The test computer configuration is as follows: the CPU is AMD Athlon 64 X2 Dual Core Processor 4000+ whose frequency is 2.1GHz, and the physical memory is 1GB.

Prime Factorization: In order to improve the efficiency of prime factorization, we introduce Rabin-Miller strong pseudo-prime test [12] in Pollard ρ prime factorization algorithm [13]. In our rough estimation we test the time required for solving the prime factorization, and the test results are shown in Fig. 1. The numbers whom we estimate ranges from 1 bit to 64 bits, and the results are obtained running randomly about 1000 different numbers between two bits. From Fig. 1 we can see that along with the increase of bit the spending time of prime factorization takes on a nonlinear growth, and an approximately 64-bit integer which is about 20-digit in the decimal system is factorized in about 6.5ms. Therefore, we can adjust the difficulty of the puzzle by constructing different large integers.

Iteration Time: In this part, we depict how iteration time can adjust the difficulty of the puzzle. Suppose that for any $i \in [1,n]$ the base a_i is equal to 2, 101 and 1021 respectively, $q = 385575647$, $r = 1000$, $l = 2000$ and $n = 100N$ in our estimation. Fig. 3 shows the test results of iteration time. For example, when $N = 10$, the iteration time n is 1000. It can be seen from Fig. 3 that with iteration time going up the spending time to solve a puzzle approximates a linear growth. At the same time, when the base a_i becomes larger, the spending time increases a little more.

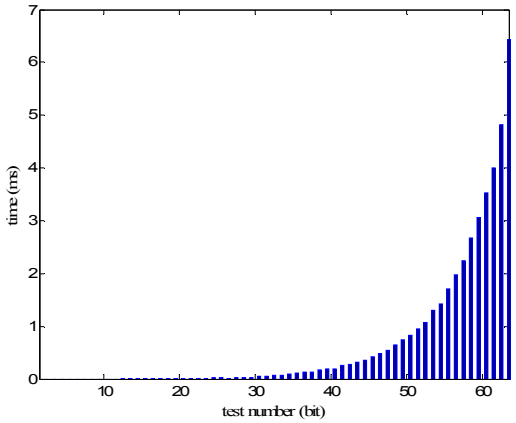

Fig.1 Time of prime factorization

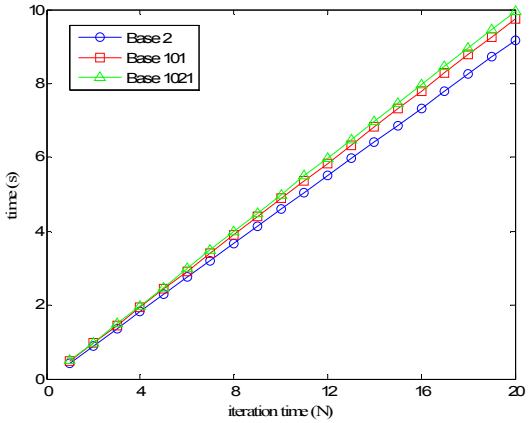

Fig.2 Time of different iteration times

Search Range: In C2Puzzle, the non-negative integer l can control the length of a search range to adjust the difficulty of a puzzle. Similarly, we also suppose that for any $i \in [1,n]$ the base a_i is equal to 2, 101 and 1021 respectively, $q = 385575647$, $r = 1000$, $n = 500$ and $l = 100L$ in our estimation. For example, when $L = 10$, the search range l is 1000. Fig. 4 shows the test results of search range. From this figure we can come to the conclusion that along with the increase of l the spending time to solve a puzzle approximates a linear growth. At the same time, when the base a_i becomes larger, the spending time also increases a little more. In C2Puzzle, the average number of solving a puzzle requires $l/2$ modular exponentiations.

Modulus: Although the modulus q may be published beforehand by the server, we still analyze its influence on the difficulty of a puzzle. The same to the above, in this estimation we also assume that for any $i \in [1,n]$ the base a_i is equal to 2, 101 and 1021 respectively, $r = 1000$, $l = 2000$, $n = 500$ and $q = m_j$ which is an element in the prime set {7, 41, 233, 4229, 60257, 532951, 2295121, 15485863, 385575647}, here $j \in [1,9]$. For example, $m_1 = 7$ and $m_2 = 41$. The prime m_j is randomly selected in different digit integer. The test results are depicted in Fig. 5. When the modulus q becomes larger, the spending time to solve a puzzle increases correspondingly. At the same time, with a_i going up the spending time also increases a little more.

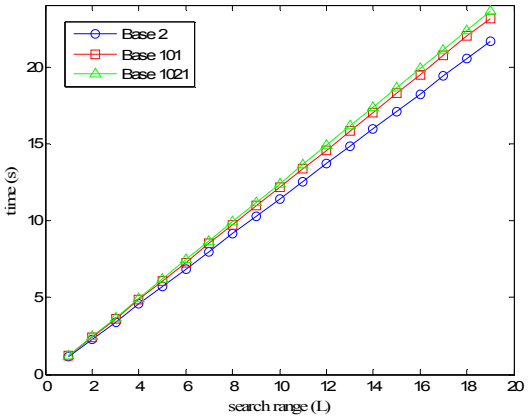

Fig. 3 Time of search range

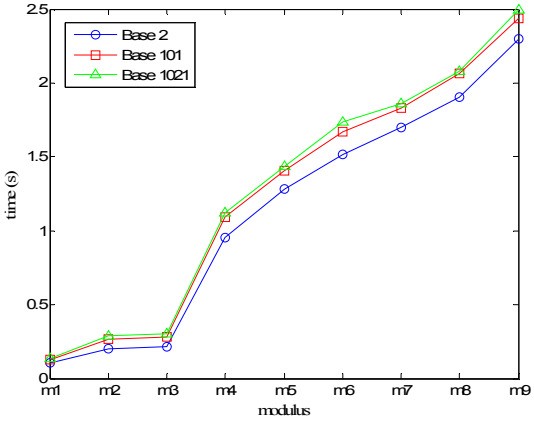

Fig.4 Time of different modulus

To sum up the above arguments, each aspect stated above can efficiently adjust the difficulty of a puzzle. The server can choose different aspects to change the difficulty of a puzzle according to the strength of the attack, and along with worsening of the attack, a client has to invest more cost to search a solution. Meanwhile, it is obvious that the more the adjustable aspects are, the smaller the possibility for a client to pre-compute a solution is.

Summary

In this paper, we propose C2Puzzle, a novel computational client puzzle for network security. By using the iterative algorithm, C2Puzzle successfully combines prime factorization with discrete logarithm problem, and is advantageous to withstand pre-computation and parallel computation to a certain extent. After we describe the puzzle generation and verification in detail, we give the readable discussions about the advantages of C2Puzzle, and show the adjustability of puzzle difficulty by four special aspects.

Acknowledgments

This work is supported in part by Hi-Tech Research and Development Program of China (No.2012AA041102-03) and in part by National Natural Science Foundation of China No.61164012. The authors are grateful to the anonymous referees for their insightful comments and suggestions.

References

[1] R. C. Merkle: *Secure Communications over Insecure Channels*, Communications of the ACM, Vol.21, No.4 (1978), p.294-299.

[2] M. S. Fallah: *A Puzzle-based Defense Strategy against Flooding Attacks Using Game Theory*, IEEE Transactions on Dependable and Secure Computing, Vol.7, No.1 (2010), p.5-19.

[3] Y. K. Jing, J. T. C. Ming, D. Niyato: *Rate Limiting Client Puzzle Schemes for Denial-of-Service Mitigation*, in: Proceedings of 2013 IEEE Wireless Communications and Networking Conference, Shanghai, China (2013), p.1848-1853.

[4] F. Tegeler, X. M. Fu: *SybilConf: Computational Puzzles for Confining Sybil Attacks*, in: Proceedings of 2010 INFOCOM on Computer Communications Workshops, San Diego, USA (2010), p.1-2.

[5] A. Jeckmans: *Practical Client Puzzle from Repeated Squaring*, Technical Report, Centre for Telematics and Information Technology, University of Twente (2009), http://eprints.eemcs.utwente.nl/15951/.

[6] B. Groza, D. Petrica: *On Chained Cryptographic Puzzles*, in: Proceedings of the 3rd Romanian-Hungarian Joint Symposium on Applied Computational Intelligence, Timisoara, Romania (2006), p.182-191.

[7] Y. Gao, W. Susilo, Y. WU, et al: *Efficient Trapdoor Based Client Puzzle against DoS Attacks*, Book Chapter in Network Security (2010), p.229-249.

[8] M. Abadi, M. Burrows, M. Manasse, et al: *Moderately hard, Memory-bound Functions*, ACM Transactions on Internet Technology (2005), Vol.5, No.2, p.299-327.

[9] S. Doshi, F. Monrose, A. D. Rubin: *Efficient Memory Bound Puzzles Using Pattern Databases*, in: Proceedings of Applied Cryptography and Network Security: 4th International Conference, Singapore (2006), p.98-113.

[10] A. K. Lenstra, J. Lenstra: *Algorithms in Number Theory*, in: Handbook of Theoretical Computer Science, Volume A, MIT Press/Elsevier, Amsterdam (1990), p.673-715.

[11] S. Tritilanunt, C. Boyd, E. Foo, et al: *Toward Non-parallelizable Client Puzzles*, in: Proceedings of Cryptology and Network Security: 6th International Conference, Singapore (2007), p.247-264.

[12] R. C. C. Cheung, A. Brown, W. Luk, et al: *A Scalable Hardware Architecture for Prime Number Validation*, in: Proceedings of IEEE International Conference on Field-programmable Technology, Queensland, Australia (2004), p.177-184.

[13] J. Buchmann, V. Muller: *Algorithms for Factoring Integers,* http://www.cdc.informatik.tu-darmstadt.de/~buchmann/Lecture%20Notes/Algorithms%20for%20factoring%20integers.pdf.

Advances in Mechatronics, Automation and Applied Information Technologies
10.4028/www.scientific.net/AMR.846-847

C2Puzzle: A Novel Computational Client Puzzle for Network Security
10.4028/www.scientific.net/AMR.846-847.1615