On Chained Cryptographic Puzzles

Bogdan Groza, Dorina Petrica

Department of Automation and Applied Informatics, Faculty of Automatics and Computers, Politehnica University of Timisoara, Bd. Vasile Parvan nr. 2, 300223 Timisoara, Romania, E-mail: bogdan.groza@aut.upt.ro

Abstract: Cryptographic puzzles provide an elegant solution in combating denial of services attacks. In this paper we introduce the concept of chained cryptographic puzzle. We define two kinds of chained puzzle constructions: linearly chained puzzles and randomly chained puzzles. The first construction prove to be very useful in some scenarios, from which the more general is in which a client may choose to solve only some of the puzzles that were sent by the server and gain resources from the server according to the amount of puzzles that he solved.

Keywords: cryptography, cryptographic puzzle, denial of services, security

1 Introduction

Cryptographic puzzles have many applications in the field of security. They are most commonly used in eliminating denial of services attacks, a number of papers proposed the use of cryptographic puzzles in combating connection depletion attacks, creating DoS resistant authentication etc. [1], [3], [5], [6]. Also cryptographic puzzles were proposed to be used for constructing time capsules such as in [7] or for partial key escrow techniques [2].

The idea which stands behind the use of cryptographic puzzles in combating various aspects of DoS is simple: if there is no evidence of an attack on the side of the entity which we call sever then resources are allocated to the clients normally, if an evidence of an attack occurs (the number of request drastically increases) then the server sends puzzles to each client that request resources. In this way a potential attacker which has request many resources has to solve many puzzles and his own computational resources are drastically consumed.

This point of view is correct, but for an honest client things may be a bit different. Imagine a honest client that has gained resources on a number of servers and all these servers are under attack, suddenly the client receives a puzzle from each server and solving the puzzle will consume a great amount from his own resources. It will be preferably for the client to continue to use the resources from the severs in a selective order, it is likely that some of the servers are of more interest for the honest client than others. In this paper we introduce the notion of chained cryptographic puzzle. Chained puzzles offer more flexibility in such a scenario since a client may solve only a particular amount of puzzles from the chain and gain resources according to the amount of puzzles that he solved.

In Section 2 we enumerate some common constructions of cryptographic puzzles. In Section 3 we introduce the notion of chained puzzle and in Section 4 we give some practical scenarios in which chained puzzles can be used. In Section 5 we conclude.

2 Some Puzzle Constructions

In this section we consider to enumerate some of the most prominent solutions to construct cryptographic puzzles.

2.1 One Way Function Inversion Puzzles

One way functions and in particular hash functions provide the first solution for constructing cryptographic puzzles. Using hash functions is a good idea since these are the simplest cryptographic primitives and require low computational power for creating the puzzle. The idea which stands behind puzzles constructed on the inversion of a one-way function is quite simple: in order to solve a puzzle an entity must find what value was the input of the one-way function in order to give a particular output. This means that the puzzle can be constructed with the following relation:

$$P = f\left(\sigma\right), \sigma = \rho \,||\, rand_l \qquad (1)$$

Given the value of P and ρ an entity must find the random l bit value, by $rand_l$ we denote a random l bit value, which was concatenated with ρ to give the particular result. This is equivalent to searching a key space of size 2^l and can be achieved on an average of 2^{l-1} hash function computations, l is also called the difficulty level of the puzzle.

2.2 Discrete Logarithm Inversion Puzzles

The idea which stands behind the construction of a discrete logarithm puzzle is of course to invert the discrete logarithm. Discrete logarithm puzzles provide additional properties that make them useful in some scenarios [1], [2].

Such a scenario, in which discrete logarithm puzzles can be used, is in paper [1] were the notion of bastion is introduced. The bastion is a secure entity which creates puzzles in order to protect a number of servers by sending the puzzles to the clients. In order to construct the puzzles a Diffie-Hellman mechanism is used. In brief, the idea is the following: a server publishes the value of a^x and then the bastion publishes a^y. Since the following relation holds:

$$\left(a^x\right)^y = \left(a^y\right)^x = a^{x \cdot y} \tag{2}$$

Both the server and the bastion are aware of the value of a^{xy}, the client receives from the bastion the value of a^y and only a partial information about the value of y (for example all but the last l bits of y). Solving the puzzle means to find the value of a^{xy}, of course in order to compute a^{xy} the client must find the value of y and this will require inverting the discrete logarithm based on the partial information that is known. The great advantage of this construction is that a bastion can protect many servers without involving them in the construction of the puzzles.

2.3 Exponent Reduction Puzzles (Time-Lock Puzzles)

The use of the discrete power function was proposed in [7]. Such a construction can be used in order to construct a time-lock puzzle; this works like a time capsule which consists in a message that can be deciphered only after a period of time. The construction is based on the fact that while working with the power function in Z_n the exponents can be reduced modulo $\phi(n)$. If the following function is used:

$$f(x) = x^2 \bmod n \tag{3}$$

Then computing the composition of the function with herself for η times can be achieved easily with the following relation:

$$f^{\eta}(x) = x^{2^{\eta} \bmod \phi(n)} \bmod n \tag{4}$$

This can be easily done by an entity which knows the factorization of n; otherwise this became a sequential process which requires η compositions. For large values of η the following relation can be used to construct the time-lock puzzle:

$$P_K = K + a^{2^{\eta}} \bmod n \tag{5}$$

Solving the puzzle means finding the value of K. Given the value of a and η an entity which is not in possession of the factorization of n can compute $a^{2^{\eta}} \bmod n$ only with η successive squaring and only then it can recover the key K from P_K in order to decrypt a particular message.

3 Chained Cryptographic Puzzles

A chained puzzle consists in a number of puzzles which may be solved only in a precise order. We will define the puzzle set as $\Pi = \{P_0, P_1, ..., P_{n-1}\}$ and the relation of order as $\Omega : \Pi \times \Pi \to \{0,1\}$. For any $i \neq j$ it holds that $\Omega(P_i, P_j) = 1$ if solving P_i requires the solution of P_j and $\Omega(P_i, P_j) = 0$ otherwise. Therefore a chained puzzle is formed by a set of puzzles Π and a relation of order Ω. Obviously the set of puzzles and the relation of order form a directed acyclic graph and for some i, j if $\Omega(P_i, P_j) = 1$ then $\Omega(P_j, P_i) = 0$ since otherwise it will be impossible to solve the puzzle; therefore in each puzzle chain there is at least one puzzle that does not depend on any other puzzle.

In the following paragraphs we will be concerned with puzzles constructed on hash functions since these are the simplest cryptographic primitives and offer a lot of computational advantages; however the concepts introduced may be extended as well for puzzles constructed with other one-way functions, for example functions over groups of integers.

We define two kinds of chained puzzles:

a) Linearly chained puzzles. A linearly chained puzzle is a set of puzzles in which a puzzle P_i depends on exactly one puzzle P_j, of course there will be exactly one puzzle which does not depend on any other puzzle in order to make it possible to solve the chain.

b) Randomly chained puzzles. Randomly chained puzzles are a generalization of linearly chained puzzle in which the puzzles are chained at random; again there will be exactly one puzzle which does not depend on any other puzzle.

In order to construct a linearly chained puzzle we will use a one way hash function f, by $rand_l$ we always denote a new sequence of l random bits, therefore $rand_l \neq rand_l$ since we always refer to a new sequence of l random bits. We define the first puzzle as:

$$P_0 = f^2(\sigma_0), \sigma_0 = \rho_0 \,||\, rand_l \tag{6}$$

The symbol $\|$ denotes concatenation. The following relation is used to construct the rest of the puzzles:

$$P_i = f^2\left(\sigma_i\right), \sigma_i = \left(\rho_i \,\|\, rand_l\right) \oplus f\left(\sigma_{i-1}\right), 0 < i < n \tag{7}$$

It is necessary to compute the value of $f\left(\sigma_{i-1}\right)$ and only then XOR it with $\rho_i \,\|\, rand_l$ since otherwise if we XOR σ_{i-1} with $\rho_i \,\|\, rand_l$ it follows that $\sigma_i = \left(\rho_i \,\|\, rand_l\right) \oplus \left(\rho_{i-1} \,\|\, rand_l\right) = \left(\rho_i \oplus \rho_{i-1}\right) \| \, rand_l$ which will make the solution of P_i independent on the solution for P_{i-1}. Solving puzzle P_i, $0 \le i < n$ means to find the value of σ_i given ρ_i such that $P_i = f^2\left(\sigma_i\right)$ which is as hard as searching a key space of size 2^l taking on average about 2^{l-1} key verifications; since each key from the key space will require two hash function computations the computational effort to recover a key is about 2^l hash computations. Since the inversion of the one-way function is computational infeasible, the computation of σ_i will require the value of σ_{i-1} and 2^l hash computations. Figure 1 illustrates the construction of such a linearly chained puzzle.

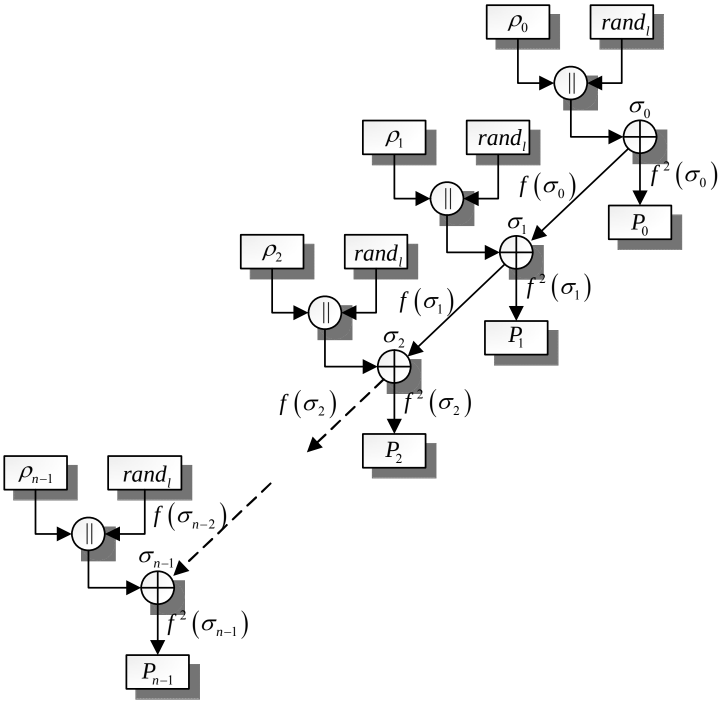

Figure 1
An example of linearly chained puzzle

In order to solve the chained puzzle a client must solve the puzzles in the order that they were constructed; the relation of order Ω is defined as follows:

$$\Omega\left(P_i, P_j\right) = \begin{cases} 1, \ i = j+1 \\ 0, \ i \neq j+1 \end{cases} \quad i, j \in \{0,1,...,n-1\} \tag{8}$$

If the order in which they were constructed is unknown to the client (this can be achieved by sending the puzzles from chain to the client in a random order) then the client can still solve the puzzle but there are $n \cdot \left(n-1\right)/2$ possible combinations. However, we do not see strong reasons to hide the order in which the puzzles were constructed since if one may want to increase the difficulty of solving the chained puzzle he can simply increase the level of the puzzles or their number.

Constructing a randomly chained puzzle can be easily done, in the same way as for the linearly chained puzzles. Relation (6) is used again to define P_0. Puzzle P_i can be made to randomly depend on any puzzle P_j for all $j = 0,...,i-1$ by computing P_i with the following relation:

$$P_i = f^2\left(\sigma_i\right),$$

$$\sigma_i = \left(\rho_i \parallel rand_l\right) \oplus \left(\Omega\left(P_i, P_{i-1}\right) \cdot f\left(\sigma_{i-1}\right) \oplus ... \oplus \Omega\left(P_i, P_0\right) \cdot f\left(\sigma_0\right)\right), \ 0 < i < n \tag{9}$$

Here by Ω we denote the relation of order and is defined as follows:

$$\Omega\left(P_i, P_j\right) = \begin{cases} 0, \ i \geq j \\ r, \ i < j \end{cases} \quad i, j \in \{0,1,...,n-1\} \tag{10}$$

By r we denote a random value from the set $\left\{0,1\right\}$; this will make σ_i depend at random on any of the keys σ_j for all $j = 0,...,i-1$.

If the relation of order Ω is known then solving the chained randomly puzzle is no harder then solving a linearly chained puzzle. Otherwise, if the order in which the puzzles were chained is unknown the client may still attempt to solve the puzzle but the number of possible combinations increases exponentially. A randomly chained puzzle can be used to increase the computational complexity for solving the puzzles by hiding the order in which the puzzles were constructed, however, the computational effort to solve the puzzle can be also increased by increasing the difficulty level of the puzzle and a randomly chained puzzle is not so usefull.

4 Some Scenarios in which Linearly Chained Puzzles can be Used

We propose the use of linearly chained puzzles in the following generic setting: the sever, or its delegate as the bastion from [6], sends a puzzle to each of its clients; this measure can be taken in order to protect the server against DoS attacks or for any purpose that requires the estimation of the clients resources. Each client may decide not to solve the entire puzzle and only some of it and the server will allocate resources to the client according to the amount of puzzle that was solved. Finally, this provides a much more flexible interaction between the client and the server.

The first scenario in which we are interested is the following: a server S will commit resources to a client $C_i, 0 \leq i < n_c$ according to the client computational resources. In the beginning all the clients are waiting in a queue, the server starts by sending a puzzle to each client, and we would like for the server to know after a particular time interval what is the stage of each client in solving the puzzle.

A first solution will be to send a simple one-way function inversion puzzle $P_i, 0 \leq i < n_c$ from the server to each client. However, when the server wants to check what is the stage of some client in solving the puzzle this is likely to be impossible since any client may claim he has searched any amount of keys without being possible for the server to verify this.

Using a puzzle with n_p sub-puzzles may help since the client may report after a particular time interval the number of puzzles that he solved by sending their solutions to the server. However, this will result in sending more than one solution and more than one verification at each step.

A linearly chained puzzle provides an elegant solution to this purpose, since it is obviously that the server may send a linearly chained puzzle to each client and each client can report what is his stage in solving the puzzles at any moment by sending the solution for the last solved puzzle from the chain. All the operations that the server needs to do is to perform a table look-up in order to check that the solution provided by the client is correct.

The second scenario in which we are interested is the use of chained puzzles in order to increase the resistance against denial of services attacks of authentication protocols based on one-way chains such as the Delayed Message Authentication Protocol (DeMA protocol) proposed in [4].

Each sessions of the DeMA protocol consists in two rounds as follows:

Session $k, 1 \leq k \leq \eta$

Round $1 \;\; A \rightarrow B: \; M_{A,k}, MAC(M_{A,k}, \sigma_A(k+1)), \sigma_A(k)$

Round 2 $B \to A: M_{B,k}, MAC(M_{B,k}, \rho_B(k+1)), \rho_B(k)$

The significance of the notations is the following: $M_{A,k}$ denotes the message from session k, MAC is a message authentication code computed on message $M_{A,k}$ with the key $\sigma_A(k+1)$ and respectively on $M_{B,k}$ with the key $\rho_B(k+1)$. The current session keys $\sigma_A(k)$ and $\rho_B(k)$ are elements from a one-way chain computed with the following relations:

$$\sigma_A(k) = f^{\eta-k}(x_A), 0 \le k \le \eta \quad (11)$$

$$\rho_B(k) = f^{\eta-k}(x_B), 0 \le k \le \eta \quad (12)$$

Here f is a one-way function, x_A and x_B are two random values kept secret on each entitie's side, η is the maximum number of communication sessions; more details on the DeMA protocol can be found in [4].

In the case of the DeMA protocol a server may decide to limit the number of requests from a particular client. Sending packages that contain a chained puzzle may be efficient and suitable for the DeMA protocol; therefore the package structure can be the following:

Session $k, 1 \le k \le \eta$

Round 1 $A \to B: (M_{A,k}, P_k), MAC((M_{A,k}, P_k), \sigma_A(k+1)), \sigma_A(k)$

Round 2 $B \to A: (M_{B,k}, S_{k-1}), MAC((M_{B,k}, S_{k-1}), \rho_B(k+1)), \rho_B(k)$

In each session the server will send a new chained puzzle to the client, the authenticity of the puzzle can be checked only in the next session, and then if the puzzle proves to be authentic the client must solve it. Starting from session $k > 0$ the server will expect a solution for the previous puzzle from the client. If such a solution is not received there are two possible reasons for this: a) the message received by the client was not authentic due to the possible intervention of an attacker and the client did not solved the puzzle since it was not authentic (in this case the communication line is corrupt) b) the client did not manage to solve the puzzle due to its limited computational resources; in either cases the server can decide to lower its computational resources allocated for the client. Figure 2 suggests the connection between the keys and the puzzles from each session.

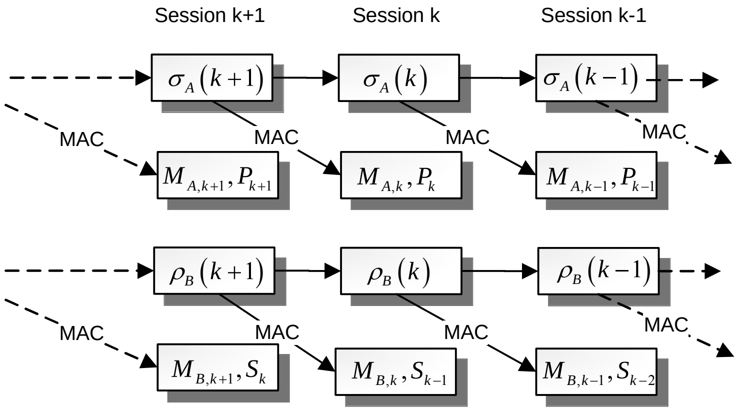

Figure 2
The connection between the keys and the puzzles from each session of the DeMA protocol

Sending a chained puzzle to the client is efficient since the client can solve only some of the puzzles from the chain and the server can allocate its resources according to the amount of solved puzzles.

Conclusions

We have proposed and investigate the construction of chained puzzles. Linearly chained puzzles can be used for providing more flexibility for the client in solving the puzzle. The generic scenario in which they can be used is in which a client may choose to solve only some of the puzzles that were sent by the server and gain resources according to the amount of puzzles that he solved. This solution may be useful in combating DoS attacks and in any other situation which requires the allocation of resources from the server according to the client's computational resources.

References

[1] Aura, T., Nikander, P., Leiwo, J., DOS-Resistant Authentication with Client Puzzles, Lecture Notes in Computer Science, 2001

[2] Bellare, M., Goldwasser, S., Verifiable Partial Key Escrow, ACM Conference on Computer and Communications Security 1997, 1997, pp. 78-91

[3] Dean, D., Using client puzzles to protect TLS. USENIX Security Symposium 2001; Washington, DC. Berkeley, CA: USENIX Association, 2001

[4] Groza, B., Using one-way chains to provide message authentication without shared secrets, accepted at IEEE 2nd International Workshop on

Security, Privacy and Trust in Pervasive and Ubiquitous Computing, Lyon, France, 2006

[5] Juels, A., Brainard, J., Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks. In S. Kent, editor, Proceedings of NDSS '99 (Networks and Distributed Security Systems), pp. 151-165, 1999

[6] Lakshminarayanan, K., Adkins, D., Perrig, A., Stoica, I., Taming IP Packet Flooding Attacks, 2nd ACM Workshop on Hot Topics Networks, Cambridge, MA, Nov. 2003

[7] Rivest L., Shamir, A., Wagner, D. A., Time-lock puzzles and timed-release Crypto, available at http://theory.lcs.mit.edu/~rivest/publications.html

[8] Waters B., Juels A., Halderman, J. A., Felten, E. W., New Client Puzzle Outsourcing Techniques for DoS Resistance, http://citeseer.ist.psu.edu/waters04new.html