New Client Puzzle Approach for DoS Resistance in Ad hoc Networks

Antonis Michalas
Athens Information Technology
GR-19002 Peania (Athens), Greece
Email: amic@ait.edu.gr

Nikos Komninos
Athens Information Technology
GR-19002 Peania (Athens), Greece
Email: nkom@ait.edu.gr

Neeli R. Prasad
Aalborg University
Fredrik Bajers Vej 7,
DK - 9220 Aalborg, Denmark
Email: np@es.aau.dk

Vladimir A. Oleshchuk
University of Agder
Department of Engineering and Science
Postbox 509, N-4898, Grimstad, Norway
vladimir.oleshchuk@uia.no

***Abstract*—In this paper we propose a new client puzzle approach to prevent Denial of Service (DoS) attacks in ad hoc networks. Each node in the network first solves a computational problem and with the solution has to create and solve a client puzzle. By combining computational problems with puzzles, we improve the efficiency and latency of the communicating nodes and resistance in DoS attacks. Experimental results show the effectiveness of our approach.**

***Keywords*—Cryptographic Puzzles; Client Puzzles; Denial of Service (DoS); Distributed Denial of Service (DDoS)**

I. Introduction

DENIAL OF SERVICE ATTACKS (DoS) is considered to be one of the most important threats as well as one of the hardest problems in computer security nowadays. The main aim of a DoS attack is the interruption of services by attempting to limit access to a machine or service instead of subverting the service itself. This kind of attack aims at rendering a network incapable of providing normal service by targeting either the networks bandwidth or its connectivity. These attacks achieve their goal by sending at a victim a stream of packets that swamps his network or processing capacity denying access to his regular clients. In the not so distant past, there have been some large - scale attacks targeting high profile Internet sites [2]. In general, we can distinguish two different types of DoS attacks: logic attacks and flooding attacks. Until nowadays, there are many security vulnerabilities which an adversary can exploit to launch a DoS attack.

Unlike common wireless networks, ad-hoc networks are characterized by the absence of any existing network infrastructure or centralized administration (decentralized wireless network) as well as the ease and speed of deployment. Such networks are highly dynamic and each node participates in the basic functions of the network like packet forwarding and routing, since there are no routers or access points. Ad-hoc networks can operate in a stand-alone way or be attached to a larger network.

Protection against DoS attacks is a crucial component of any security system. Traditional DoS attacks involve overwhelming a particular host. However, in ad hoc networks, mobility, limited bandwidth, routing functionalities associated with each node, etc, open many new opportunities for launching a DoS attack. These attacks might be at the routing layer or at the MAC layer [9].

A. Our Contribution

In this paper we propose a new client puzzle to avoid DoS attacks in ad hoc networks, where nodes do not necessarily belong to a single authority and the topology can be altered dynamically. Our solution has the following four attributes:

- The creation of puzzles is not outsourced to a third entity (decentralized design).
- The connection initiator is responsible to create and solve the puzzle on the fly.
- Verifying a puzzle solution requires very little work for a node. More precise, the time needed is comparable with a simple table lookup.
- The difficulty of the puzzle is easy adjustable.

Lot of previous schemes involve puzzle distribution on central units or virtual channels. Our approach is decentralized since this is a fundamental property of ad hoc networks and especially for mobile ad hoc networks (MANETs). In our approach every node that is trying to contact another node has to solve two puzzles. The first one is a discrete logarithm problem (DLP) and the solution of this puzzle will help the connection initiator to create and solve the second puzzle which is considered to be the most difficult. Apart from that, even if a node does not solve correctly the DLP but just find the solution from a table lookup this will not make any harm to our network since the puzzle that will have to create and solve in the next step will be based not only to the solution of DLP but to another random number as well.

Every node needs to follow the same procedure even if it is legitimate or malicious, while the solution of the puzzle can become more difficult or easy depending on the trust

level and traffic congestion. Furthermore, DLP and our puzzle are combined with the best optimum approach for ad hoc networks.

B. Organization

Following this introduction, the paper is organized as follows. In Section 2, we examine related works to prevent DoS attacks with the use of client puzzles. Section 3 describes our puzzle construction. Section 4 presents experimental results along with the advantages and disadvantages of our technique and Section 5 concludes the paper.

II. Related Work

Client puzzles approaches have been proposed by several researchers so as to avoid denial of service attacks. For example Juels and Brainard in [5] applied client puzzles to TCP SYN flooding, where they mention that also SSL encloses a parallel problem. In this paper, the authors proposed a cryptographically based countermeasure against connection depletion attacks. Connection depletion is a denial-of-service attack in which an attacker seeks to initiate and leave unresolved a large amount of connection requests to a server, exhausting its resources and rendering it unable to serve valid requests. TCP SYN flooding is a well-known example of such an attack. They introduced a countermeasure (a client puzzle protocol) which the basic scheme was as follows: When a server comes under attack, it distributes small cryptographic puzzles to clients making service requests. To complete its request, a client must solve its puzzle correctly.

An alternative method for implementing client puzzles techniques was proposed by Waters *et al.*. [8]. This technique assumes that the client puzzle protocol is a three party protocol and builds a client puzzle based on the discrete logarithm problem for which authenticity and correctness can be verified using a Diffie-Hellman based technique. One of the main differences of this technique is that puzzle generation can be outsourced from the server to another external bastion host, yet verification of solutions can be performed by the server itself [10].

The attack and possible remedies were analyzed in detail by Schuba *et al.* in [6], where they contribute a detailed analysis of the SYN flooding attack and introduced a new solution, which offers protection against these types of attacks for all hosts that are connected to the same local area network and it's independent of their operating system or networking stack implementation.

Narasimhan *et al.* in [11] introduced the notion of hidden puzzle difficulty, where the attacker cannot determine the difficulty of the puzzle without expending a minimal amount of computational resources. Game theory was used to develop defense mechanisms.

Finally, a game based analysis of the client puzzle approach in order to defend DoS attacks have been introduced in [7], where the optimal strategy is derived for the attacked server in order to respond to DoS attacks effectively.

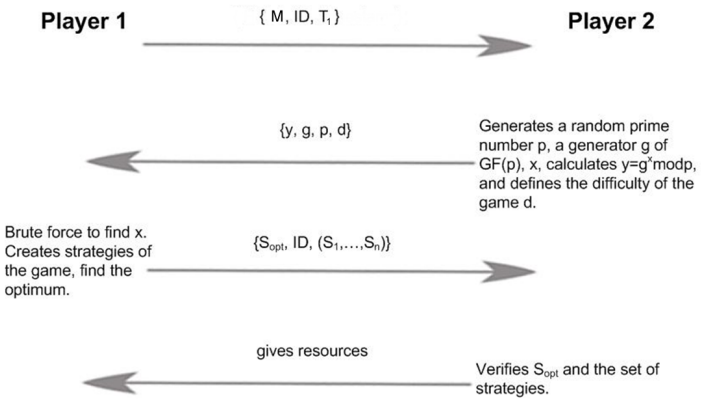

Fig. 1. Client Puzzle

III. Construction of the Puzzle

Our approach follows similar definitions used in game theory, where we have a set of players $P_1, P_2 \in P$, a set of possible strategies $S_1, S_2, ..., S_n \in S$ for every player and a payoff for each combination strategies.

Based on Figure 1 let's suppose that $node_1$ which from now on will be referred as P_1, wants to communicate with $node_2$ (P_2). P_1 sends to P_2 his ID, a hello message M and time stamp T_1. When P_2 receives these three parameters it first generates a random prime number p, a generator g of $\mathbb{Z}_p^*$ and a random integer x. Then sets the difficulty level d of the puzzle that P_1 will have to solve. Next, P_2 calculates value y according to expression (1) as following:

$$y = g^x mod\ p \tag{1}$$

and responds to P_1 by sending g, y, p, T_2, d.

When, P_1 receives g, y and p solves equation (1) over $\mathbb{Z}_p^*$ by using the baby-step giant-step algorithm [12]. Next, P_1 computes the integer division of x and d of the puzzle (i.e. x/d) which is equal to number n that is needed to create the strategies of the puzzle. After finding n, P_1 is responsible for generating and solving the puzzle. This means that P_1 will first need to calculate all strategies S and second to find the optimum strategy S_{opt}.

Each nodes strategy is defined as the sum of three positive integers (n_1, n_2, n_3) in a non-descending order such that:

$$n_1 + n_2 + n_3 = n, \tag{2}$$

where $n = \lfloor \frac{x}{3} \rfloor$.

The number of strategies that each puzzle will have, can be easily calculated from the expression (3) of Theorem 1.

Theorem 1: For all $n \in \mathbb{Z}_+^*$ the number of strategies, is given by expression (3)[1]:

$$\Pi(n) = \frac{1}{2} \cdot \left\lfloor \frac{n}{3} \right\rfloor \cdot n + \frac{1}{8} \cdot (-1)^{n-3\lfloor \frac{n}{3} \rfloor} + \frac{1}{8} \cdot (-1)^{1+n} - \frac{3}{4} \cdot \left\lfloor \frac{n}{3} \right\rfloor^2 \tag{3}$$

[1]The proof is omitted due to space constrains.

Since P_1 can find all possible strategies of the puzzle, it can also find the optimum strategy (S_{opt}). To calculate the optimum strategy, P_1 will 'play' every strategy against all other strategies by using a utility function. However, P_1 compares each strategy by itself and therefore the utility function will take as input two different strategies ($S = (n_1, n_2, s_3), S' = (n'_1, n'_2, n'_3)$), until all possible combinations have been 'played'. The utility function $U(S, S')$ will have the following three possible results:

- S, if strategy S has at least two higher coordinates (n_i, $i \in [1, 3]$) than the corresponding ones of strategy S'.
- $DRAW$, if strategies S, S' have one equal coordinate (for example $n_2 = n'_2$) and two different coordinates, but shared equally (for example $n_1 > n'_1$ and $n_3 < n'_3$)
- S', if strategy S' has at least two higher coordinates (n'_i, $i \in [1, 3]$) than the corresponding ones of strategy S.

When P_1 finds S_{opt}, it sends back to P_2 a set of all strategies S, optimum strategy S_{opt}, value of x and its ID. Upon reception, P_2 verifies the strategies including the optimum and serves P_1. An example of how a puzzle can be solved, follows.

Example 1: Let's assume that $n = 6$ [2]. P_1 needs to calculate all possible combinations in which n can be written as a sum of three positive integers in non-decreasing order, such that $n_1, n_2, n_3 \in [1, n-2]$. From expression (3) the number of strategies when $n = 6$ is $\Pi(n) = 3$. Hence,

$$6 = 1 + 1 + 4, 6 = 1 + 2 + 3, 6 = 2 + 2 + 2$$

which means that for this particular puzzle there are the following three strategies:

$$S_1 = (1, 1, 4), S_2 = (1, 2, 3), S_3 = (2, 2, 2). \quad (4)$$

Now P_1 finds the optimum strategy S_{opt} by using the utility function (if the number of strategies is k, P_1 uses the utility function $\frac{k \cdot (k-1)}{2}$ times). For strategies S_1, S_2, S_3 we have:

$$\begin{aligned} u(S_1, S_2) &= u((1,1,4),(1,2,3)) \\ \Rightarrow u(S_1, S_2) &= (Draw, S_2, S_1) \\ \Rightarrow u(S_1, S_2) &= Draw \\ u(S_1, S_3) &= u((1,1,4),(2,2,2)) \\ \Rightarrow u(S_1, S_3) &= (S_3, S_3, S_1) \\ \Rightarrow u(S_1, S_3) &= S_3 \\ u(S_2, S_3) &= u((1,2,3),(2,2,2)) \\ \Rightarrow u(S_2, S_3) &= (S_3, Draw, S_2) \\ \Rightarrow u(S_2, S_3) &= Draw \end{aligned}$$

From the results above it's clear that the optimum strategy S_{opt} is S_3 since it wins once and has two draws. Therefore, P_1 sends to P_2 the optimum strategy S_{opt}, the set of all strategies S, the value of x and its ID. Upon receiption of the parameters, P_2 first verifies that equation (III) holds for all possible strategies and second uses Algorithm 1 to guarantee that P_1 has correctly found S and S_{opt}.

[2]This is the smallest value that n can have, and the solution is trivial.

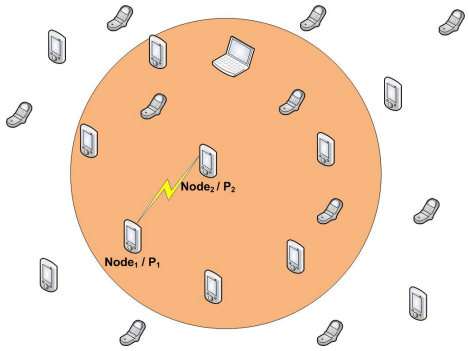

Fig. 2. Small scale ad hoc network.

Algorithm 1: Verification of finding S and S_{opt}

```
Input: V(S, S_opt), x, ID
begin
    if (S_opt[0] > V_i[0] and (S_opt[1] > V_i[1] or
    S_opt[2] > V_i[2])) or (S_opt[1] > V_i[1] and
    (S_opt[0] > V_i[0] or S_opt[2] > V_i[2])) then
        optWins++;
    end
    if V_{i-1}[1] < V_i[1] then
        clientSum+ = V_i[0];
    end
    else
        expectedSum+ = (V_i[1] − j) · V_i[0];
        if expectedSum == clientSum and optWins
        ≥ ⌈Π(n)/2⌉ then
            giveResources;
        end
        else
            Drop Connection;
        end
    end
end
```

IV. Experimental Results & Security Discussion

In order to measure the effectiveness of our solution, we have used the Smart Dust simulator written in Java. Our testbend consists of a desktop computer with Xeon CPU at 2.80GHz, with 2.00 GB of RAM running Windows XP.

Our experiments where implemented in a small scale ad hoc network as seen in Figure 2. Every message of a node is broadcasting to each of its neighbors. When nodes wish to communicate with their neighbors, they broadcast a message. Upon reception of messages, nodes check whether a particular message is intended for them. If so, the node processes the request as we described in Section **??** else it drops the message.

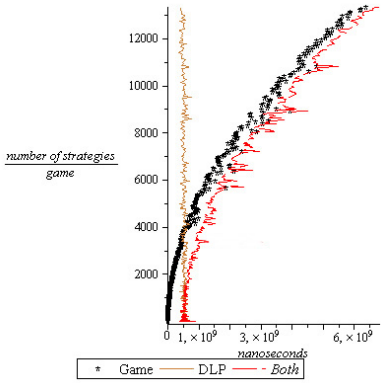

Fig. 3. Number of strategies with relation to time required for the puzzle, the DLP and combination of them.

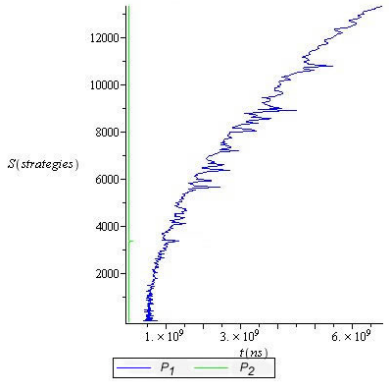

Fig. 4. Time required for P_1 to solve the DLP and the puzzle and for P_2 to verify the solution (Figure 1).

Figure 3 shows how much time it takes to find x from (1), to solve only the puzzle (i.e. find S_i and S_{opt}) and to combine both (i.e. client puzzle of Figure 1). Particularly, we tested our algorithm with different levels of difficulty (low, medium, difficult) and solved puzzles that had from 3 up to 13333 number of strategies. From the results we observe that for a puzzle with 13333 strategies the time needed P_1 to solve the puzzle is 6.38 sec, while the time for both solving equation (1) by x and the puzzle is 6.81 sec, which is an efficient time even for mobile ad hoc nodes even if we consider that there are many puzzles with less number of strategies than 13333 and therefore less time to be solved. Table I, shows some representative values of Figure 3.

TABLE I
RESULTS FROM FIGURE 3

Strategies	**DLP Time(sec)**	**Puzzle Time(sec)**	**Combiation Time(sec)**
444	0.38	0.02	0.40
1976	0.50	0.66	0.89
6769	0.53	1.68	2.21
10034	0.59	3.50	4.10

For solving equation (1) we used the Baby-Step-Giant-Step algorithm which is a generic algorithm that falls into the class of square root algorithms (best possible performance without exploiting properties of a particular group with complexity $O(\sqrt{n})$).

A very important property of client puzzles is the fact that verifying a puzzle solution should require very little work for a server. Considering that, we measured the time that P_2 needs in order to generate the parameters of equation (1) and to verify the solution of the client puzzle. Also we found the time P_1 needs to solve equation (1) and our client puzzle. As seeing in Figure 4, the computational cost of P_2 who generates the DLP and verifies the puzzle, is significantly less than the computational cost of P_1 who solves the puzzle and plays the game. Specifically, the minimum time that a node needs to generate the DLP and to verify the puzzle is 0.0007 secs, while the maximum time is 0.0075 secs (Table II).

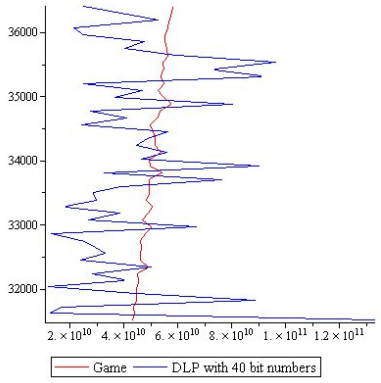

Fig. 5. Discrete Logarithm versus Our Puzzle with more strategies

TABLE II
RESULTS FROM FIGURE 4

Strategies	**Total Time (sec) for P_2**	**Total Time (sec) for P_1**
444	0.0020	0.40
1976	0.0038	0.89
6769	0.0048	2.21
13333	0.0057	6.81

To prove the effectiveness of our approach we consider that P_1 and P_2 will solve only our puzzle. In order to achieve the same level of security as before, we increased the number of strategies from 13333 up to 36410. Furthermore, we increased the security level from 2^{32} to 2^{40} for the DLP and we measured the time that P_1 needed to solve it. In this case, the average time was 44.88 secs, while for our puzzle was 50.53 secs.

We noticed that using DLP with 40 bit numbers is making the problem very difficult and thus it is not suitable for ad hoc networks, since every node would have to spend many computational resources and time until it finds the solution. Figure 5 illustrates equation (1) with 40-bits parameters and our puzzle based only on S and S_{opt}. As it is seen, there are cases where finding S and S_{opt} needs slightly more time than finding x, but in most cases finding x takes much more time.

Moreover, we have calculated the complexity of solving the DLP ($O(\sqrt{n})$), of finding S, S_{opt} ($O(n)$) and of completing the client puzzle of Figure 1 ($O(n\sqrt{n})$).

We have also made some attacks to our algorithm, in order to see how it handles malicious requests. Random nodes send requests to P_2. From the requesting nodes, some of them where malicious and tried to break the security of the system with one of the following ways: Sending strategies from a puzzle that was solved in the past (replay attack), sending the correct optimum strategy but the rest of the strategies was wrong, sending the correct optimum strategy but some of the rest strategies was wrong, sending a totally nonsense answer, sending the correct strategies but a wrong optimum strategy, sending all the strategies correct but did not send the optimum strategy at all and sending wrong x (did not solve the DLP correct) or fake d.

Each of the above attacks, the system realized that some of the expected data were wrong and thus the connection with the malicious node was dropped. Note that in our implementation we do not make use of any database to store information, instead only a vector is used where the node saves the id, the value of x and the difficulty d. Furthermore, the difficulty of the puzzle is increasing analogous to the requests that a node is receiving from its neighbors. If for example all neighbors send a request between a short time interval t, then the puzzle will be difficult for all of them. Apart from that, each node can only have a single open connection with each node. This means that multiple requests to the same node are not allowed.

Our approach can prevent TCP SYN flooding attacks, because as we described in Section 3 the node that is making the request has to perform complex computations than the one who is receiving the request. Our technique is based on the concept that we want to put a possible malicious node of an ad hoc network to perform expensive computations, in order to be able to use some of the resources of another node. This means that the resources of the first node will end before the other ones does. This assumption becomes even stronger in the case of ad hoc networks where nodes have limited resources and short battery life. Furthermore, legitimate users would experience just a negligible computational cost, except for the situation that a node is receiving plenty of requests at a specific time interval t (i.e all of his/her neighbors are sending request at the same time interval t).

Attacks like teardrop, bonk and boink are beyond the scope of this paper and we assume that some of the existing techniques can be used in order to defeat them. Most systems knows how to deal with such attacks now and a firewall can drop fragmented packets in return for more latency on network connections since this makes it disregard all broken packets.

Furthermore, in this paper we do not make use of client puzzle outsourcing techniques since we are referring to ad hoc networks without central authority. The existence of a central unit could lead to situations that malicious nodes could make directly attacks to this machine. Consider a situation where all the nodes in a network are overflowed. Then all the legitimate users will be left without offered services since there would be no one to create puzzles for them neither to verify solved puzzles.

For small smart devices like PDA's and mobile phones, we achieve DoS resistance. We have shown that using only the discrete logarithm problem with a marginary level of security, nodes do not have the necessary resources to solve it. In other words, by increasing the level of security in DLP, the solving time rapidly increases. On the contrary, by adding our puzzle and increasing the number of strategies, with lower level of security in DLP we achieve higher efficiency of computing resources.

V. Conclusion

Unlike traditional networks ad hoc networks use dedicated nodes to support their basic functions like packet forwarding, routing, and network management[1]. In this paper we proposed a new client puzzle to prevent denial of service attacks in ad hoc network nodes. In our approach we do not use any central node that creates client puzzles and verifies the solutions. Instead every node in the network is responsible to generate puzzles and to avoid attacks that could take down the entire network. In our future work we are planning to create a reputation system for every node in a network, that will be based on the solution we presented in this paper. Finally, we are planning to extend our work by creating a distributed DoS defensive mechanism. To do that all nodes will be distributed in groups and they will have to take part to a multiplayer game in order for their requests to be processed.

References

[1] N. Komninos, D. Vergados and C. Douligeris, *Layered security design for mobile ad hoc networks*, In Computer and Security Journal, Elsevier. pages 121-130, September 2005, doi:10.1016/j.cose.2005.09.005

[2] C. Douligeris and A. Mitrokotsa 2004. *DDoS attacks and defense mechanisms: classification and state-of-the-art*. In Comput. Netw. 44, 5 (Apr. 2004), 643-666. DOI= http://dx.doi.org/10.1016/j.comnet.2003.10.003

[3] Deanna Koike, *Client Puzzles as a Defense Against Network Denial of Service*, In ECS 228 (Cryptography for E-Commerce), December 4, 2002.

[4] Dean Drew and Stubblefield Adam, 2001. *Using client puzzles to protect TLS.* In Proceedings of the 10th Conference on USENIX Security Symposium - Volume 10 (Washington, D.C., August 13 - 17, 2001). USENIX Security Symposium. USENIX Association, Berkeley, CA, pages 1-1.

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

[6] Christoph L. Schuba, Ivan V. Krsul, Markus G. Kuhn, Eugene H. Spafford, Aurobindo Sundaram and Diego Zamboni. 1997. *Analysis of a Denial of Service Attack on TCP.* In Proceedings of the 1997 IEEE Symposium on Security and Privacy (May 04 - 07, 1997). SP. IEEE Computer Society, Washington, DC, 2008.

[7] Boldizsar Bencsath, Istvan Vajda and Levente Buttyan, *A Game Based Analysis of the Client Puzzle Approach to Defend Against DoS Attacks.* In Proceedings of the 2003 International Conference on Software, Telecommunications and Computer Networks. pages 763-767, 2003.

[8] Brent Waters, Ari Juels, Alex J. Halderman, and Edward W. Felten, 2004. *New client puzzle outsourcing techniques for DoS resistance.* In Proceedings of the 11th ACM Conference on Computer and Communications Security (Washington DC, USA, October 25 - 29, 2004). CCS '04. ACM, New York, NY, 246-256. DOI= http://doi.acm.org/10.1145/1030083.1030117

[9] V. Gupta, and S. Krishnamurthy, and M. Faloutsos, *Denial of Service Attacks at the MAC Layer in Wireless Ad Hoc Networks.* In MILCOM - Network Security, Anaheim, 2002. citeseer.ist.psu.edu/karpijoki01security.html

[10] P. Morrissey, N. Smart, and B. Warinschi, *Security notions and generic constructions for client puzzles.* In ASIACRYPT, 2009. pages 151-165.

[11] Harikrishna Narasimhan, Venkatanathan Varadarajan and C. Pandu Rangan, *Game Theoretic Resistance to Denial of Service Attacks Using Hidden Difficulty Puzzles.* In ISPEC, 2010. pages 359-376.

[12] Stinson, D. R. 2002. *Some baby-step giant-step algorithms for the low hamming weight discrete logarithm problem.* Math. Comput. 71, 237 (Jan. 2002), 379-391. DOI= http://dx.doi.org/10.1090/S0025-5718-01-01310-2