SPAM Control Using CPU Timestamps

Sandesh Jagannath and Radhesh Mohandas, *Member, IACSIT*

***Abstract*—Email spam has reached alarming proportions because it costs the sender very little to send; almost all of the costs are paid by the recipient, carrier and the email servers. Email spam's inadvertently affect the performance of email servers which are kept busy in processing huge number of emails sent by the spammers. It also affects the productivity of the recipients who have to skim through lots of irrelevant emails to find the ones that actually require their attention. We propose a solution to control spam at the ingress points on the email servers by throttling the sender's CPU i.e., making an email client pay a stamp fee for every email sent. The stamp fee is collected in terms of resource usage, in our case CPU cycles. The solution is based on the use of Discrete Logarithm Problem (DLP), which is considered to be one of the hardest mathematical problems to solve and is the basis for many cryptosystems. DLP is used to generate CPU stamps as a proof of the CPU cycles expended by the sender's system. A separate stamp has to be calculated for each email and for each recipient which restricts the total number of emails that an email client can send in a time interval. We further claim that a normal user will not notice this cost in his day to day operations and a spammer will not be able to get past this mechanism.**

***Index Terms*—CPU Stamps, CPU Throttling, Discrete Logarithm Problem, SMTP**

I. Introduction

Spam is a growing problem with electronic mail. The key reason for the growth of spam is the fact that it costs nothing to send email. Consequently it costs a spammer nothing to send out millions of emails per day. Anti-spam laws have not helped much because it is difficult to track and convict spammers across international borders. Spam filters require frequent upgrades yet do not work all the time.

Drawing a parallel from the postal system, a simple solution to the spam problem is to make senders pay money to send email. However, the idea of paying money for sending email is not appealing [1].

Computational proof for spam control is an alternative to digital money based payment for email. In this approach the sender pays for email using computational effort (Penny Black Project, Microsoft), like CPU cycles [1] or memory cache hits [2]. Further, a verifiable proof that this payment was made is attached to the email. This proof (also called proof-of-work [3]) convinces the receiver that the sender spent a certain amount of computational effort on the email just for that particular receiver. If the cost of computing the proof is chosen in such a way that it adds about a few seconds to the time taken to send email, it does not have a major impact on the normal sender who sends out only a few emails every day. The spammer however simply cannot afford to spend this additional time as it slows down his throughput by a huge factor.

Our proposed solution makes use of Discrete Logarithm Problem (DLP) to throttle the sender's CPU and in turn generate CPU Stamp. Stamps are generated for every recipient of an email and a verifiable proof for the same is attached to the email. We propose to use DLP because it is not possible to solve it in polynomial time on non quantum computers of the current day; moreover it has been the basis for many cryptosystems.

A. Discrete Logarithm Problem

Our solution uses DLP. In this Section we present the basics of DLP and the definitions that will be used in our solution.

1) *Definition*

Let p be an odd prime, $Zp = \{0, 1,....,p-1\}$ a finite field, Z_p^* the set of integers which are relatively prime to p i.e., $Z_p^* = \{a \in Zp \mid \gcd(a, p) = 1\} \Rightarrow Z_p^* = \{1, ..., p-1\}$. Let α be a generator of Z_p^* i.e., $Z_p^* = \{\alpha^0 \bmod p, \alpha^1 \bmod p, \ldots\ldots, \alpha^{p-1} \bmod p\}$ and β be a non zero integer in Zp such that

$$\beta \equiv \alpha^x \bmod p \tag{1}$$

Given p, α, and β, finding x is called the Discrete Logarithm Problem.

2) *Notations*

We make use of the following notations throughout this paper.

TABLE I: Notations Used

p	An odd prime
selectPrime	Prime p for which $(p-1)/2$ is also prime
Zp	A finite field.
Z_p^*	The set of integers which are relatively prime to p i.e., $Z_p^* = \{1, ..., p-1\}$
α	A generator of Z_p^*
β	Integer in finite field other than zero and one i.e., $\{\beta \in Zp \mid \beta \neq 0 \text{ or } 1\}$
pdigits	Number of digits in prime p
x	Value computed using equation 1
a_h	Hash of message-body
b_h	Hash of email-id
h	Hash function

Rest of the paper is organized as follows. Section II discusses the related work. The proposed solution is discussed in Section III. Section IV presents the algorithms. We give the implementation details and results in Section V and Section VI respectively and finally conclude in Section VII.

Manuscript received August 15, 2012; revised October 12, 2012.

Sandesh Jagannath is with Oracle Corp. in Bangalore, India (e-mail: Sandesh.Jagannath@gmail.com)

Radhesh Mohandas is with the department of Computer Science and Engineering, NITK-Surathkal, Karnataka, Indiathe department of Computer Science and Engineering

II. Related Work

In this section we present some of the previous works which have proposed mechanisms to control outgoing spam.

Reference [4] proposed a solution using Hashcash in May 1997. It was originally proposed as a mechanism to throttle systematic abuse of un-metered internet resources such as email, and anonymous remailers. Back has proposed Hashcash as a CPU cost-function to compute a token which can be used as a proof-of-work.

The computational technique [1] for combating junk mail and controlling access to a shared resource has already been explored. The main idea is that a user is required to compute a moderately hard, but not intractable, function in order to gain access to the resource, thus preventing frivolous use. Dwork and Naor presented different types of pricing functions based on various ideas like extracting square roots modulo a prime, the Fiat-Shamir signature scheme, and the Ong-schnorr-Shamir signature scheme.

Reference [2] proposed that memory-bound pricing functions for computational spam fighting are better than CPU-bound pricing functions. The main idea is that since memory access speeds vary across machines much less than do CPU speeds, memory-bound functions may behave more equitably than CPU-bound functions.

Penny Black Project (Microsoft) is investigating several techniques to reduce spam by making the sender pay. They're considering several currencies for payment: CPU cycles, memory cycles, Turing tests (proof that a human was involved), and plain old cash. Since computer resources are already being spent on email; they propose increasing this cost to about ten seconds for unsolicited email.

The use of ticket servers has been proposed by Abadi and et al [5]. In this proposed technique clients can contact a ticket server and obtain tickets to use for any Network service (such as sending email). The recipient of such a request (such as the email recipient) can use the ticket server to verify that the ticket is valid and that the ticket hasn't been used before.

III. Proposed Solution

The flow of operations in the proposed solution is as follows.

1) After an email sender composes an email and clicks on send, email client sends a 'helo' SMTP command to the email server along with its domain name.
2) On receiving the 'helo' command, the email server sends a prime p from a subset of *selectPrime* for the current session. The p value is injected into the SMTP reply command along with the positive reply code for SMTP 'helo' command.
3) Client extracts the p value injected into SMTP reply command for the current session. Client also computes $\{\alpha, \beta\}$ using the algorithm given in Section IV(B). In order to calculate these values the client needs to compute the hash of the message-body and email-id of the recipient (equation(3) and equation(4)).
4) Client next computes 'x' for the values $\{p, \alpha, \beta\}$ using equation(1). The algorithm to find 'x' given $\{p, \alpha, \beta\}$ is given is Section IV(D).
5) Client then builds a stamp using these computed values and adds the stamp as an extra header of the email. The stamp is of the following format.

$$x\text{-dlp}: a_h : b_h : \alpha : \beta : x : p \quad (2)$$

Client computes $\{a_h, b_h, \alpha, \beta, x\}$ for every recipient of the email, and adds an extra header for each recipient. Multiple recipients are processed sequentially.

6) Client then forwards the email to the email server (ingress server); the email server which acts as an MDA (Mail Delivery Agent).
7) The ingress email server extracts each stamp added as an extra header and verifies it, also checks that p value in the stamp is the same that it had injected for the current session. The stamp verification algorithm is given in Section IV(F).
8) The result of the verification process is added as another extra header of the email, which indicates to the receiving email server that this email has passed some kind of spam control and it can deal with this email with a different priority.

A. How a Spammer Counter-Attacks

Before we start discussion about the various algorithms used in our solution we will look into various scenarios in which a spammer tries to counter our spam control mechanism. We will also present the solutions to deal with these scenarios.

Scenario 1: A spammer computes the stamp only once and appends the same stamp for each email and for each recipient.

Our proposed solution has inherent mechanism to deal with this counter technique. Each stamp is verified at the ingress point by the email server. If the spammer fakes a stamp, that stamp's verification will fail because the values α and β of DLP are calculated as a function of message-body, email-id, and p.

Scenario 2: A spammer guesses the value of p for this session based on its value in the previous session.

On the server side we are selecting a subset of primes from the prime space containing all *selectPrime* of 5 to 8 digits. We use *selectPrime* to avoid Pohlig-Hellman attack [6]. When the SMTP session begins, the server will choose a prime from this subset of *selectPrime* to send to the client for the current session. This subset is updated periodically to avoid guessing attack by a spammer. TABLE II gives details about such primes. This method of choosing a prime p makes it difficult for a spammer to guess the value of p.

TABLE II: Total Number of Primes and Primes whose $(P-1)/2$ Also a Prime.

Digits	5	6	7	8
Total no of primes	8363	68906	586081	5096876
No of *selectPrime*	555	3654	26333	198911

Scenario 3: A spammer bluffs the value of p.

In each of the verification level, given in Section IV(F) the first check that the ingress email server does is to verify that the value of p in the 'x-dlp' header is the same that it had injected for the current session. So, if the spammer

bluff's the value of p, verification of the stamp fails at the ingress email server.

Scenario 4: A spammer fakes hash values.

To elaborate this attack we can say that the spammer uses same hash of message-body and email-id for each recipient. And then when he takes mod p over these hash values as in equation(5) and equation(6) to calculate the values $\{\alpha, \beta\}$ he gets the values in the range [2, p-1] which is the admissible finite space for these values. To overcome this counter attack the ingress email server has to be configured to do a Level II verification of DLP stamp (Section IV(F)). In this level of verification the ingress email server calculates the hash of message-body and email-id and verifies that they match with the values in 'x-dlp' header. So, if a spammer fakes these values his email will fail DLP stamp verification.

IV. Algorithm

A. Method to Calculate Hash

We propose to use MD5 algorithm which produces a 128-bit output. We truncate this output to 32-bit value suitable for our application. Whether cutting the output of some cryptographic hash-function hurts its security with respect to collision resistance is an open research problem ("unnatural" constructed examples exist). But NIST (probably with the approval of the NSA) used the cutting technique to get the SHA-224 from SHA-256 anyway [7].

B. Algorithm: Compute {A, B} Using Message-Body and email-Id

1) The client computes hash of the message-body and email-id

$$a_h = h(\text{message-body}) \quad (3)$$
$$b_h = h(\text{email-id}) \quad (4)$$

where 'h' is the hash function that has been described in the previous Section.

2) Client then computes

$$\alpha = a_h \bmod p \quad (5)$$
$$\beta = b_h \bmod p \quad (6)$$

3) Client checks the value of α computed in step 2. If α = 0 or 1 or if it's not a generator of p, then α = nextgen(p), where 'nextgen' is a function which finds next generator of p (a generator whose value is greater than α) using the algorithm given in Section IV(C).
4) Client then checks the value of β computed in step 2. If $\beta < 2$ or $\beta = \alpha$, select the next 32-bit from the 128-bit hash computed using the hash function given in Section IV(A), take mod p and use it as the value of β and repeat the process until a suitable value for β is found.

C. Algorithm: Efficient Computation of Next Generator

1) Input: $\{\alpha, p\}$. Create a Boolean array isChecked[] of size 'p' and initialize all the values of the array except isChecked[0] to false. Compute limit = (p-1)/2
2) $\alpha = \alpha$+1. Pick g = α as the potential generator of p.
3) If isChecked[g] = false
   Set isChecked[g] = true
   Compute value = glimit mod p
4) If value = 1, go to step 2 else go to step 5.
5) count = 1, prev = g
6) Compute
   - prev = (prev*g) mod p
   - count = count + 1
7) if prev ≠ 1 & count < limit, go to step 6, else go to step 8.
8) if prev = 1, go to step 2, else go to step 9.
9) Return 'g' as the next generator of 'p'.

This is the most efficient algorithm to find the next generator when value in equation(5) is not a generator of 'p'.

D. Algorithm: Solve for 'x' given {p, α, β}

SolveX function

1) Take $\{p, \alpha, \beta\}$ as input.
   Initialize x=0, prev = 1.
2) Calculate val = modpow(α, x, prev)
   prev = val.
   If val = β, go to step 4, else step 3.
   $x = x$+1, if $x < (p$-1), go to step 2.
3) Return 'x' found in step 2 as the solution of DLP.

modpow function
if x = 0, return 1.
if x >1, return (prev* α) mod p.

E. Stamp Generation

A Stamp is generated by the client. Stamp has the format given in equation(2). This generated stamp is added as an extra header of the email. A stamp is a proof that the sender of the email has expended a few CPU cycles before sending the email and it is generated for each recipient of an email.

F. Stamp Verification

Verification of the stamp is done at ingress email server. Result of the verification process is added as an extra header of the email. All those emails which have stamps that have been verified to be correct can be treated with a different priority by the recipient's email server. The verification process at the ingress email server is classified into two levels. The administrator can configure which level to use based on the requirements.

1) *Level I verification*
   In this level of verification the email server extracts the values $\{\alpha, \beta, p, x\}$ from the 'x-dlp' header of each recipient
   a. Email server verifies that p is a value that it had injected for the current session.
   b. It then verifies that the values $\{\alpha, \beta, p, x\}$ satisfy equation(1).
2) *Level II verification*
   In this level of verification the email server extracts the values $\{\alpha, \beta, p, x\}$ from the 'x-dlp' header of each

recipient. It also extracts the message-body and email-id of each recipient.

Email server verifies that p is a value that it had injected for the current session.

Email server then computes α and β values using the same algorithm that is used by the client (Section IV(B)). It verifies that these values match those added in the header.

It then verifies that the values $\{\alpha, \beta, p, x\}$ satisfy equation(1).

3) *Verification at the downstream email server*
This is the verification process that the receiving email server can do. A spam filter at the receiving server can be configured to check for presence of '*x*-dlp' header. Based on the presence or absence of this header the downstream server can deal with that email with a different priority.

V. Implementation

We have implemented our solution in the form of an add-on for Thunderbird email client and a module for *Sendmail* SMTP server. The add-on is used for DLP stamping of emails on the client side and module for *Sendmail* is for verification of the stamps. These modules and client, server system configuration are discussed in the subsequent sections.

A. Server and Client System Configuration

Server configuration: Intel core2 Duo CPU with processor speeds of 3.00 GHz, 2 GB RAM, Fedora 12 operating system.

Client configuration: Intel core2 Duo CPU with processor speeds of 2.00 GHz, 1 GB RAM, Windows XP.

B. Thunderbird Client Add-on

To throttle the client's CPU and to generate CPU stamps we have developed an add-on for Thunderbird email client. This add-on does the job of receiving p value for the current session from the email server, computing hash of message-body and email-id of each recipient, computing α and β values, solving for 'x', building a stamp for each recipient, and adding the stamp as an extra header of the email. We modified an open source add-on "*Penny-Post*" [8] and added our DLP stamping module to it.

C. Sendmail Module to Inject a Prime Value

In order to inject a prime value p for an SMTP session, we had to augment SMTP protocol, for this we have modified the source code of *Sendmail* SMTP server which is available as open source. The modified SMTP server injects p into SMTP 'helo' reply command. The add-on in thunderbird email client extracts this p value to be used for DLP.

D. Sendmail Module to Verify Stamp

For DLP stamp verification to be done on the email server, we added a new module to *Sendmail* SMTP server which verifies the '*x*-dlp' header of each recipient. The verification process begins once the SMTP server receives end of message (eom) SMTP command. This module after verifying the stamp in '*x*-dlp' header adds another extra header which indicates whether a stamp passed or failed in the verification process. This header can be used by the receiving inbox to classify the mail based on heuristics and user rules if it wants to skip verification.

VI. Results

A. Client Delay

Our stamping algorithm is designed to add some delay on the client side before the email is sent to the ingress email server. To see how much delay it adds, we have tested our add-on for large number of recipients and arrived at the graphs given in Fig. 1 and Fig. 2. We can notice from Fig. 1 that given a fixed message-body size, when we increase the number of recipients the delay on the client side increases linearly. But, when we fix the number of recipients and vary the message-body size (plain text used as message-body) as in Fig. 2 we see that the delay on the client side almost remains constant. This is because the hashing algorithm that we are using is fast enough to hash large message-body in relatively constant time. There is hardly any delay on the client side when our spam control mechanism is not used, because the client instantaneously communicates with the email server. Thus, the line indicating delay without DLP spam control mechanism is very close to the x-axis in the graph of Fig. 1. We have also calculated delay with attachments of type text file, image file and video file. This result is shown in Fig. 3.

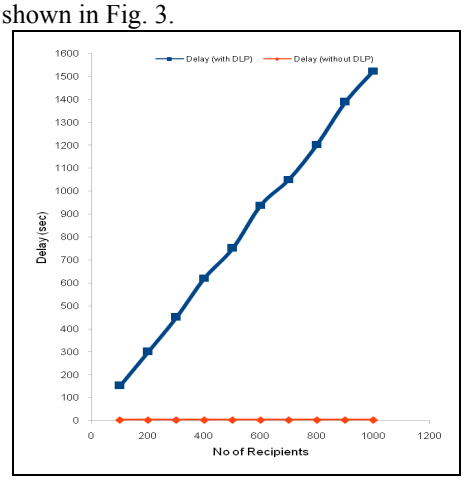

Fig. 1. Client side delay with fixed body size

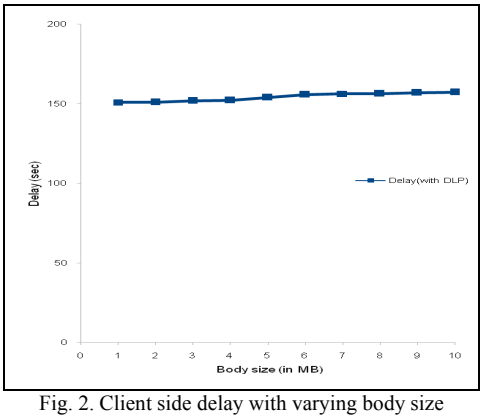

Fig. 2. Client side delay with varying body size

B. Server Throughput

The throughput of an email server is the number of messages that it can process in an interval. It can be measured in terms of messages per second (MPS) that are served. Our solution includes a server side module which

verifies DLP stamps. This adds a bit of overhead to the server which reduces the overall throughput of the email server. To calculate by what factor the throughput reduces we ran a test on a standard *Sendmail* SMTP server and our modified *Sendmail* SMTP server which does DLP stamp verification. We sent constant email traffic for same duration in both the cases and calculated the number of messages served per second. The graphs in Fig. 4 and Fig. 5 show the throughput in a setup with solution and without solution respectively. It is clear from these graphs that the throughput for a setup with solution reduces by a small factor. We computed the factor by which the throughput reduces by using the average throughput for the test duration.

- Average throughput for setup with solution: 2.758 MPS
- Average throughput for setup without solution: 3.012MPS
- Factor by which throughout reduces:

$$100 - ((2.758/3.012)\times 100) = 8.4\ \%$$

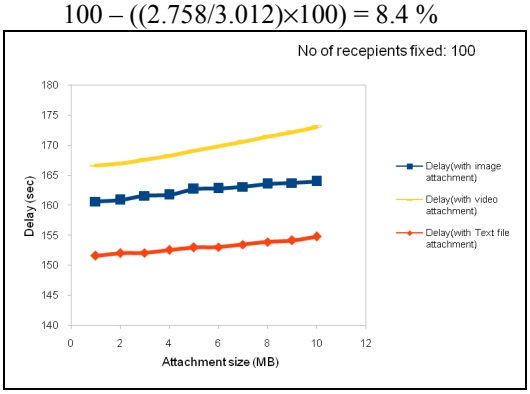

Fig. 3.Client side delay with attachment

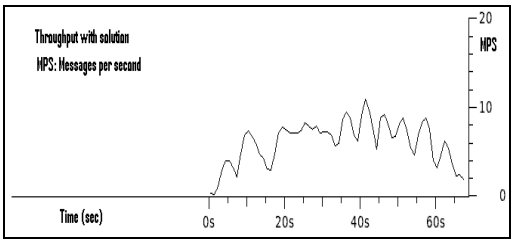

Fig. 4. Email server throughput with solution

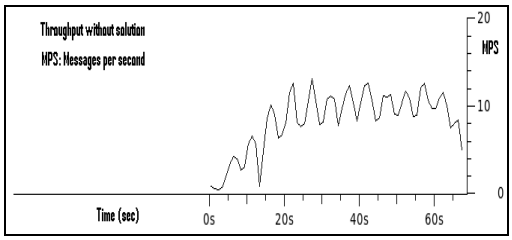

Fig. 5. Email server throughput without solution

VII. Conclusion

In this paper we proposed a solution to control spam rate at the ingress points. The proposed solution uses DLP to throttle the client's CPU and generate CPU stamps as a verifiable proof. Our solution adds a minor delay on the client side. This delay is very negligible as far as a normal user is concerned, but reduces the throughput of a spammer by a huge factor. Our solution adds a very less overhead to the email server which is involved in the verification of DLP stamps. The throughput of the server with our solution is approximately 92% of the throughput without solution. From the graph of Fig. 3 we can say that a spammer is more likely to use text files instead of image or video files to spread malicious content when he is using our DLP module on the client side. We have implemented our DLP stamping algorithm as a module in an open source add-on which makes it a practically usable solution to control spam. Also, the module that we have implemented for verification of DLP stamp is compatible with any version of *Sendmail* SMTP server. We have also presented a fast algorithm to find the next generator of a prime p and also an efficient algorithm to compute x given $\{p, \alpha, \beta\}$ of DLP.

References

[1] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail," in *Proc. of CRYPTO 1992*, Santa Barbara, 1992, pp. 139-147.
[2] C. Dwork, M. Naor, and A. Goldberg, "On Memory-Bound Functions for Fighting Spam," in *Proc. of CRYPTO 2003,* Santa Barbara, 2003, pp. 426-444.
[3] B. Laurie and R. Clayton, "Proof-of-Work proves not to work," in *Proc. of 3rd Workshop on the Economics of Information Security*, Minnesota, May 2004.
[4] A. Back Hashcash - A denial of service counter- measure. [Online]. Available: http://hashcash.org/
[5] M. Abadi, M. Burrows, M. Manasse, and T. Wobber, "Moderately hard, Memory-Bound functions," in *Proc. of 10th Annual Network and Distributed System Security Symposium, San Diego*, February 2003, pp. 25-39.
[6] D. Hankerson, A. Menezes, and S. Vanstone, "Guide to Elliptic Curve Cryptography," New York: Springer-Verlag, 2004, pp. 155-160.
[7] Recommendation for Applications Using Approved Hash Algorithms, NIST 800-107-2009.
[8] A. Lokhandwala, "Penny Post-Computational Proof for Spam fighting," M. *Sc dissertation, Dept. Comp. Science*, Univ. of Birmingham, UK, 2007.

Sandesh Jagannath was working as a Member Technical Staff in Oracle Corp. in Bangalore, India since August 2010 completed his M.Tech in Information Security from NITK-Surathkal, Karnataka, India in June 2010. He carried out the research work presented in this paper, as part of his Master's Dissertation under the guidance of Radhesh Mohandas. Prior to doing Master's he graduated with a B.E degree in Computer Science from PESIT-Bangalore, Karnataka, India in July 2008. While doing his Master's he was privileged to work under Prof. N. Balakrishnan, Associate Director, Indian Institute of Science – Bangalore, Karnataka, India as a summer intern during May-June 2009. He carried out research on Security in Mobile Ad-Hoc Networks during this internship. His area of interest being Cryptography & Information Security, he keeps himself abreast of the latest happenings in this field and is very much inclined towards carrying out more research work in this area.

Radhesh Mohandas was joined his alma-mater NITK-Surathkal, Karnataka, India as an adjunct faculty in the department of Computer Science and Engineering after serving in software industry in the silicon valley for close to six and half years. He worked for companies like Microsoft and Sarvega Inc in the Silicon Valley after completing his Master's in Computer Science from the Washington University. He retired from active software job at Microsoft Corp., Redmond, USA in December 2006 before he moved to his motherland India to work as a faculty. His area of interest being Cryptography & Information Security, he has guided many students in his alma-mater to carry out successful research in this area and has published many technical papers.