Augmenting a webmail application with cryptographic puzzles to deflect spam

Marius Cristea

Faculty of Automatics and Computers
Department of Automation and Applied Informatics
”Politehnica” University of Timisoara, Romania
Email: cristea12@gmail.com

Bogdan Groza

Faculty of Automatics and Computers
Department of Automation and Applied Informatics
”Politehnica” University of Timisoara, Romania
Email: bogdan.groza@aut.upt.ro

***Abstract*—In order to increase the resilience against spam, we design and implement a protocol based on cryptographic puzzles for an open-source web based e-mail client. Our proposal is compatible with existing e-mail infrastructure and does not require modifications on the server side. The only add-on is a stand-alone ticketing server that is used to deliver the current cost to each sender. The puzzles that we use are time-lock puzzles which have the benefit that they cannot be subject to a parallel attack due to their intrinsic sequential nature. Thus an adversary that gains control over multiple hosts cannot use them to solve a puzzle, the risk of a directed attack against some receiver being reduced. Also, the protocol allows the sender of the e-mail to generate the puzzle himself, releasing the e-mail server from an additional computational task. We analyze the efficiency of the proposed solutions in terms of computational costs and the results are satisfactory.**

***Index Terms*—spam, DoS, cryptographic puzzle**

I. Introduction and related work

Cryptographic puzzles, or client puzzles, are a commonly proposed mechanism for combating various resource depletion attacks. This is because they can be used as a proof that a certain amount of computational effort was done and thus resources can be allocated more rationally between clients based on proofs-of-work. Thus, in practice, puzzles encountered various uses against packet flooding [1], TCP SYN flooding [2], protecting the TLS [3], creating DoS resistant authentication [4] or protecting web servers [5]. The idea to trade computational power for the ability to send e-mails was introduced by Dwork and Naor [7]. Later it was independently proposed by Back in the hash cash system [6]. In [7] the initial idea of trading computational power from [8] is improved by using memory-bounded functions, previously addressed by Abadi et al. [9], in order to alleviate differences between cpu speeds (as memory access time does not vary so much between machines).

Still, applications that protect against spam by the use of cryptographic puzzles are not wide spread yet in practice. In this paper we develop a protocol and an implementation for the *Yawebmail* open source webmail client. There are two main goals in the proposed application: first is to use cryptographic puzzles to limit the ability of the spammer to send more spam e-mails by associating computational costs to each e-mail, second is to make no modifications on the server side since we can't depend on the fact that e-mail providers will adopt any modification. For this second issue we will use a retrofit proof-of-work mechanism in which the sender of an e-mail will pay computational time to a ticketing server, his costs for sending a new e-mail is also based on the history of the e-mails he previously send. All these goals should be achieved in such a manner that we can keep the compatibility with existing e-mail protocols such as POP3, IMAP and SMTP.

Practical implementations to combat spam can be found in [10], [11] and [12]. The first paper does not use cryptographic puzzles, although the authors point out that the application could be easily extended with puzzles. The last two papers use cryptographic puzzles but they require modifications on the server side. Also, our solution differs both at the protocol level as well as in the implementation layers that are addressed. In the proposed protocol, the server does not need to spend time to build the puzzles as they are built and solved by the sender itself. Also the SMTP protocol is not modified, the puzzle is just added as an appendix to the original e-mail.

As for the construction of the puzzles, many variants were proposed in the last decade, based on symmetric functions as well as on simple or more sophisticated number theoretic problems. The most commonly used constructions are based on hash functions, namely either by inverting the hash function for some low entropy input or by finding the input for which the image of the function has a particular pattern. In our protocol we will use the discrete squaring function that gives non-parallelizable time-lock puzzles [13], [14]. This kind of puzzle has the advantage that searching for its solution is not a parallelizable procedure, due to the intrinsic recurrent nature of the modular exponentiation. Thus, an adversary that gains control on multiple hosts cannot use all of them to solve a puzzle faster. In this sense, a directed attack on a particular receiver will not be so effective. A second advantage, for which we used it in our protocol, is that it allows the sender of the e-mail to build the puzzle himself, thus releasing the e-mail server from the computational task of generating the puzzles.

The paper is organized as follows. In section 2 we provide details on the protocol and the puzzles that we used. Section

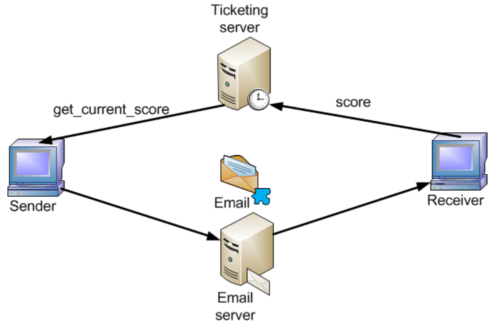

Fig. 1. Protocol setting

3 gives the implementation details and holds the experimental results. Section 4 holds the conclusion and future work.

II. The proposed protocol

A. Protocol description

The participants in the protocol setting are depicted in Figure 1. E-mail senders are requested to perform the computational task of solving a puzzle for each e-mail they send based on the score obtained on previously sent e-mails. This score is continuously updated by the receiver on the ticketing server each time he connects with his webmail browser, details on the filter used to compute scores are given in the next section dedicated to implementation details. Initially, solving the puzzle may take less than a second, but the cost drastically increases if more e-mails from the same sender are classified as spam (in particular, in the implemented application the difficulty doubles each time). Indeed the sender has the choice of not solving the puzzle, but in this case the webmail application will simply mark such an e-mail as spam. The ticketing server TS is responsible for various functions such as providing information from the receiver of the e-mail (puzzle details, score, etc., all the information that is needed in order to solve the puzzle corresponding to each e-mail) and to assure time synchronization between protocol participants. The e-mail server, i.e., MS, has no other function but the usual one: to receive and store e-mails. We assume that all digital certificates of the clients and servers (when needed) are signed by some trusted authority. In the concrete application setting, e-mail senders and receivers need to use the webmail application that we developed while the ticketing server is a stand-alone application, the rest is unchanged.

The puzzle that is appended to the e-mail must depend on a fresh and unpredictable value. This is because we want to avoid a replay attack as well as off-line computations, where some malicious senders can start to solve the puzzles long before the e-mail is send. However, as in our scenario we want to avoid the involvement of the e-mail server in this process while the receiver of the e-mail may be absent at this time, for this purpose we will use as fresh and unpredictable value in each puzzle the signed time value provided by the ticketing-server. Indeed an adversary cannot predict the value of this signature so he cannot start to create puzzles before a particular time-stamp was issued. To avoid the replay of some already used puzzles for e-mails send by the same sender in the same time interval with the same time-stamp (we assume that the ticketing server issues time stamps at regular intervals, for example several seconds), the easiest way will be to ask the sender to insert a distinct nonce in each puzzle and the receivers to store all puzzles received from some sender to check for uniqueness.

Thus, the puzzle construction that we use when A wants to send an e-mail to B is $h(\{t_{TS}\}_{SigTS}, N_i, \mathrm{email}_i, B, A)^{\epsilon} mod n_B$, where n_B is the modulus of receiver B, N_i is the nonce produced for the i^{th} e-mail, $\{t_{TS}\}_{SigTS}$ is the time-stamp signed by the ticketing server and ϵ is an exponent of the form 2^k where k is the difficulty level of the puzzle. Indeed this solution will not require too much space, because the puzzle is small, and identifying whether a puzzle was or not sent before is fast because one can perform a binary search. However, this straight forward solution is not portable if one considers that the inbox can be cleared, or moved from a server to another one. For this purpose, old puzzles can be cleared after a certain amount of time. For example, each receiver chooses a delay, Δ_B in the case of sender B, which denotes the time period for which he stores the nonces for some sender (in particular this can be in the order of hours or days). In particular, for honest senders, the value of Δ_B has no importance as they will honestly generate a new nonce every time an e-mail is sent.

Now, in order to deem a particular e-mail as spam, there are two conditions that can be separately met: i) the e-mail does not have the correct puzzle solution ii) the e-mail is not associated with the correct time interval. The check for the correct time is based on two criteria: first the e-mail must contain a correct time-stamp and second the time-stamp must not point to a time value previous to the most recent sign in of the receiver (to overcome potential delays between sending and receiving a safe margin can be set to minutes, hours, etc.). We underline that each time the client connects to the e-mail server it is desirable to perform a new time synchronization to avoid wrong spam classifications due to clock drifts.

In brief, the protocol works in the following way: a client who wants to send an e-mail to another client has to solve a puzzle according to the specifications from the certificate of the the receiver (using the prescribed modulus, difficulty level and delay) and to his current score, all this information is provided by TS. The solution to the puzzle is appended to the e-mail and the receiver checks for its correctness at the time when he receives it.

We now describe in brief these protocol steps. The two headed arrow in the protocol description implies that the channel is authentic and confidential, this can be easily implemented under SSL/TLS. This channel assumption must hold for all communications with TS first because we must ensure the authenticity of the exchanged values. Also, in the protocol description, the use of alias_X as the alias of principal X was preferred for confidentiality issues in the case when participants do not want TS to be able to monitor principals with which they communicate. For this case, participants can simply choose a random alias and register on TS, in our implementation this is done as default by the webmail application. The only channel in which there are no restrictions is the regular e-mail channel.

Communication with TS when sending an e-mail

1. $A \twoheadrightarrow TS$: alias_B
2. $TS \twoheadrightarrow A$: $Cert_B, \text{score}_{AB}, \{t_{TS}\}_{SigTS}$

Sending the i^{th} e-mail at time interval δ

1. $A \rightarrow MS$: $\text{email}_i, \{t_{TS}\}_{SigTS}, N_i,$

$$\text{puz} = h(\{t_{TS}\}_{SigTS}, N_i, \text{email}_i, B, A)^{\epsilon} mod n_B$$

2. $A \twoheadrightarrow TS$: $\text{alias}_B, \text{puz}$

Communication with TS after receiving an e-mail

1. $B \twoheadrightarrow TS$: $\text{alias}_A, \text{score}_{AB}(\text{puz}), \text{puz}$

The certificate $Cert_B$ includes the values of the modulus n_B and the exponent ϵ_B. After the synchronization procedure the following inequality must be verified: $\Delta_{Err} << \Delta_B$ where Δ_{Err} is the synchronization error, i.e. the round trip time from A to TS. In practice this inequality should always hold, as it is natural to expect that the synchronization error will be smaller than the time-to-live of the timestamps chosen by some client. These values are of different magnitudes, the value of Δ_B can be set to hours, days or weeks, depending on how much the user will want to store the nonces while the synchronization delay should be in the order of milliseconds.

An informal discussion on the security of the protocol may be useful. We considered that the communication channels between participants and TS are confidential and authentic (this is induced by the two headed arrow notation). There are many ways to achieve this in practice and it was not our point here to define how. The only channel that is not secure, is the communication between the sender and the e-mail server, this is in order to cover the usual scenario of sending and receiving e-mails. Since this channel has no security an adversary can take any malicious action on it. However, the adversary cannot reuse the proof-of-work because it is bind to the content of the e-mail and to the identity of the sender and receiver. Of course, if the adversary alters the information in any way the e-mail will be marked as spam, which indeed is.

B. Cost evaluation

We now consider to evaluate the performance of the proposed proof-of-work based protocol. In order to evaluate the profit of a spammer we consider the following parameters:

- n - number of targets for the spammer
- c_{send} - cost for a spammer to send an e-mail
- c_{puz} - cost for a puzzle of difficulty level 0
- α - difficulty increase factor for spam
- r - revenue of a spammer for sending one e-mail
- p - probability that the a spam is correctly classified

The spammer profit should be computed over k send sessions. In the first session, the cost will be $n\cdot(r-(c_{send}+c_{puz}))$. In the second session the adversary will have to pay a double computational price for clients that successfully classified the spam in the first receive session and no computational cost for the others. This follows the binomial distribution and thus for the k^{th} sending session we have:

$$\text{Profit} = \sum_{i=0}^{k} n\cdot(r-(c_{send}+2^i\cdot c_{puz}))\cdot p^i\cdot(1-p)^{k-i}\cdot\frac{k!}{i!\cdot(k-i)!} \tag{1}$$

This is for the particular case of $\alpha = 2$, i.e., when the difficulty doubles each time a potential spam is send. Using precise values for the previous parameters to make a concrete estimation is not easy as they vary from context to context. We will use the values from the seminal work of Laurie and Clayton as a reference [15]. In their estimation the cost for sending one e-mail is around 0.005 cents considering the cost of electricity (8.5 cents), the cost of the computer (50 cents) and the number of e-mails (15,000) that can be send per day. If these results may be outdate, even if we consider Moore's law, the cost per e-mail today is reduced with about one order of magnitude at around 0.0006 cents, i.e., $c_{send} = 0.0006$. In order to estimate the cost of a puzzle we can assume the same price for a computer (75 cents per day) which leads to 3 cents per hour. If some e-mail operator will want to send 10000 non-spam e-mails every hour or so then the cost should be set at $c_{puz} = 0.0003$. As for the value of p we can use the results from [16], according to their work the detection rate is over 60% with all filters and gets around 90% with the best ones, thus it is fair to take $p = 0.6$. Additionally we must also measure the computational waste due to wrong classifications. For this we will denote by p' the false positive rates and considering that in this case the time to solve a puzzle is wasted time we get:

$$\text{Waste} = \sum_{i=0}^{k} n\cdot 2^i\cdot c_{puz}\cdot p'^i\cdot(1-p')^{k-i}\cdot\frac{k!}{i!\cdot(k-i)!} \tag{2}$$

Using these relations we draw the plots in figure 2 for the case of a spammer with $100,000$ target e-mails addresses. As can be seen, after sending an average of 10 e-mails to each user the profit of the spammer falls below 0. If we assume a polynomial increase rate, i.e., $i^{4/3}$, in the puzzle difficulty with an initial cost of the puzzle set to one order of magnitude higher, i.e., $10\times$, the profit of the spammer is less, but the waste due to false positives is higher.

One potential limitation of this protocol resides in the fact that a spammer will have to pay only next time he decides to send an e-mail. To avoid this, one can use an on-line cost procedure in which the e-mail is send for evaluation to the ticketing server which sends back the score for the particular

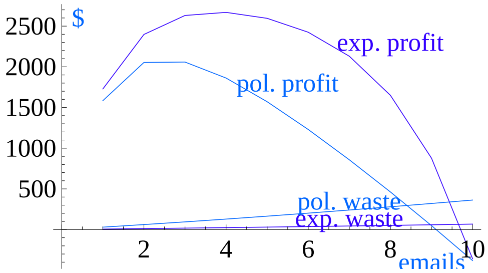

Fig. 2. Spammer profit and computational waste with exponential and polynomial increase factor of puzzle complexity

e-mail, or else the ticketing procedure can be implemented on the e-mail server as well. Nevertheless, we are more interested in spammers that go for a long term income (such as spam companies) which are likely to return.

III. Implementation Details and Results

A. Design of the application

The use of electronic mail (e-mail) is based on two protocols for receiving e-mail (IMAP and POP3) and one protocol for sending e-mail (SMTP). The difference between IMAP and POP3 is that an e-mail client using POP3 protocol will download the e-mails on the local host. If IMAP is used the e-mails are kept on the server and only the selected mail is downloaded on the client host for reading (off-line capabilities are also available in IMAP clients as well). As we used a webmail based application IMAP was the protocol of choice. In order to filter spam e-mail we are enforcing the sender to generate a puzzle based on the receivers requirements. To achieve this we are adding two new e-mail headers: *puzzle* which is the solution to the puzzle generated by the sender and *random_a* which represents a random number used to generate the puzzle. The process of sending and receiving e-mails will look as in Figure 3. The e-mail server doesn't need to be modified as it will only forward the e-mails from the sender to the receiver. The receiver will decide if a particular e-mail should be marked as spam or not, based on the puzzle solution it receives together with the e-mail. The message will be marked as spam if the puzzle is not correct or if there are timing problems as discussed previously.

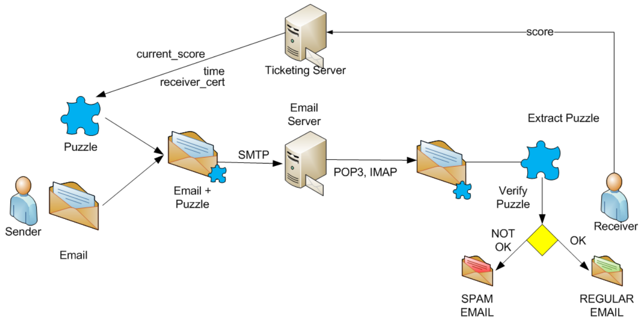

Fig. 3. The sending/receiving process of e-mails with puzzles

As the introduction of time-lock puzzles is transparent for the e-mail server, to implement the proposed solution we need two things:

- An *e-mail client* that is able to compute and verify time lock puzzles. This will be achieved by extending an existing e-mail client, i.e. *Yawebmail*
- A *ticketing server* such that the clients can make their information publicly available. This solutions needs to be distributed so that any client can obtain his score and puzzle details from anyone that he wants to communicate via e-mail.

The client can be any e-mail client that can be extended so that it will support time lock puzzles. We have chosen *Yawebmail*, because it is an open source webmail client and supports a lot of existing e-mail protocols. *Yawebmail* is written in Java as web application and is developed by *Openwebtools*. To be able to support spam filtering by using time lock puzzles, *Yawebmail* was extended with 2 packages *security* and *data*.

The *security* package is responsible for generating and verifying time lock puzzles. This is basically done in the *PuzzleUtils* class, which is included in applet. The generation and solving of the puzzles is done in the *PuzzleUtils.generatePuzzle()* method and it works in the usual way: generates the random value a; computes $puzzle$ = $hash(t_\delta{+}a{+}mess{+}adr_{rec}{+}adr_{send})^{2^k}$ by using k iterations, where: k represents the number of iterations required to compute the puzzle, n represents the receivers modulus, $mess$ represents the message content, t_δ represents the timestamp received from the ticketing server, adr_{rec}/adr_{send} is the receiver's/sender's e-mail address.

After the solution of the puzzle is computed the values a and $puzzle$ are appended to the e-mail message as e-mail headers. The verification of the puzzle is done in *PuzzleUtils.verifyPuzzle()* method and it works as follows. First it computes r = $2^k mod\phi(n)$, and then b=$hash(t_\delta + a + mess + adr_{rec} + adr_{send})^r modn$ where $\phi(n)$ = $(p-1)(q-1)$, p and q being two large prime numbers and t_δ represents the timestamp issued by the ticketing server. Then, if $puzzle$ = b then the received puzzle is correct otherwise the e-mail is marked as spam.

A spam filter runs on the receiver side and it gives a score for every incoming mail based on an Java spam filter (jASEN). If the e-mail is identified as spam then the cost for the sender will be increased with a factor of two.

The *data* package is responsible to hold the puzzle information of all the clients that have an account on the webmail server.

The ticketing server is used by clients to obtain the digital certificate that stores the information needed to compute the puzzles required by the clients they want to communicate with as well as the their current cost. For testing purpose we have created a Ticketing Servlet. This is a servlet that based on the client's request it responds with a digital certificate containing the receiver's puzzle information, which is stored in an xml file (clients.xml), together with the client's score and the timing data. The servlet's main source code can be found in the *servlet* package and it works based on the values of the URL parameters.

B. Experimental results

We now analyze the computational cost of the time lock puzzles. The tests were made on four different machines: (1) notebook with an 1.73 GHz Intel Centrino Processor having 1 GB of RAM and running Windows XP SP3 Professional 32 bit, (2) PC with an 2.4 GHz Intel Core 2 Quad Processor having 4 GB of RAM and running Windows Vista Business SP1 32bit, (3) PC with an 2.66 GHz Intel Core 2 Duo Processor having 2 GB of RAM and running Windows XP SP2 Professional 32 bit, (4) Nokia N95 8GB with a Dual ARM 11 333 MHz processor giving 128 MB of RAM running Symbian OS 9.2, S60 rel. 3.1.

The time required for one modular multiplication, i.e., for a puzzle of difficulty level 0, is displayed in table 1. For the mobile version a modified version of Bouncy Castle's BigInteger was used. The values represent the average time of 1000 runs for each of the 4 different modulus sizes: 512, 1024, 2048 and 4096 bits. The 512 bit modulus is small and is relevant just for comparison because the verification of the 1024 bit puzzle can be done on one 512 bit factor in order to save computational time.

$\lfloor n \rfloor$	512 BITS	1024 BITS	2048 BITS	4096 BITS
(1)	$141 * 10^{-6}s$	$422 * 10^{-6}s$	$1573 * 10^{-6}s$	$6000 * 10^{-6}s$
(2)	$76 * 10^{-6}s$	$225 * 10^{-6}s$	$763 * 10^{-6}s$	$2950 * 10^{-6}s$
(3)	$78 * 10^{-6}s$	$208 * 10^{-6}s$	$703 * 10^{-6}s$	$2672 * 10^{-6}s$
(4)	$26 * 10^{-3}s$	$69 * 10^{-3}s$	$168 * 10^{-3}s$	$641 * 10^{-3}s$

TABLE I
TIME SPENT FOR ONE MODULAR MULTIPLICATION

To correlate these results with the cost evaluation from section 2 we could set the default difficulty of the puzzle at a modulus of 1024 bit with 30 iterations. As we can see from Table 1 sending an e-mail will require only modest amounts of time for honest users. But a spammer who usually sends thousand of e-mail at once will be strongly delayed by solving many puzzles since the puzzle size increases by a factor of 2 for each e-mail detected as spam. If the spammer refuses to solve the puzzles than his e-mails will be marked as spam by default.

IV. CONCLUSIONS AND FUTURE WORK

We proposed a solution that remains compatible with the existing e-mail protocols and only a small change in the existing parts of the e-mail services is needed: the introduction of a ticketing server for distributing the information needed to construct the puzzle. Our cost measurements showed that the introduction of puzzles affects the profit of the spammer much more severe than it affects honest senders. Of course, further improvements could be added to the proposed protocol. Our solution helps the client to determine spam e-mail, but it doesn't stop the traffic of unsolicited e-mail through the Internet. Thus, solutions based on puzzles are only complementary and can be used together with other, existing, anti-spam solutions to improve the performance of the e-mail system in general. The use of cryptographic puzzles to combat spam is not yet wide spread and is still debatable, but the results obtained so far leave us optimistic over this matter.

ACKNOWLEDGEMENT

First author was partially supported by the strategic grant POSDRU/88/1.5/S/50783, Project ID50783 (2009), co-financed by the European Social Fund Investing in People, within the Sectoral Operational Program Human Resources Development 2007/2013. Second author was partially supported by national research grants POSDRU/21/1.5/G/13798 and PNCDI PN II 940/2009.

REFERENCES

[1] K. Lakshminarayanan, D. Adkins, and I. S. A. Perrig, “Taming ip packet flooding attacks,” *SIGCOMM Comput. Commun. Rev.*, vol. 34, pp. 45–50, January 2004.

[2] A. Juels and J. Brainard, “Client puzzles: A cryptographic countermeasure against connection depletion attacks,” in *Proceedings of NDSS '99 (Networks and Distributed Security Systems)*, 1999, pp. 151–165.

[3] D. Dean and A. Stubblefield, “Using client puzzles to protect tls,” in *Proceedings of the 10th conference on USENIX Security Symposium - Volume 10*, ser. SSYM'01. Berkeley, CA, USA: USENIX Association, 2001, pp. 1–1.

[4] T. Aura, P. Nikander, and J. Leiwo, “Dos-resistant authentication with client puzzles,” in *Revised Papers from the 8th International Workshop on Security Protocols*. London, UK: Springer-Verlag, 2001, pp. 170–177.

[5] E. Kaiser and W. Feng, “mod kapow: Protecting the web with transparent proof-of-work,” in *In Proceedings of INFOCOM 2008*, 2008, pp. 1–6.

[6] A. Back, “Hashcash - a denial of service counter-measure,” Tech. Rep., 2002.

[7] C. Dwork, A. Goldberg, and M. Naor, “On memory-bound functions for fighting spam,” in *Proceedings of the 23rd Annual International Cryptology Conference*. Springer-Verlag, 2003, pp. 426–444.

[8] C. Dwork and M. Naor., “Pricing via processing or combatting junk mail,” in *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*. London, UK: Springer-Verlag, 1993, pp. 139–147.

[9] M. Abadi, M. Burrows, M. Manasse, and T. Wobber, “Moderately hard, memory-bound functions,” *ACM Transactions on Internet Technology*, vol. 5, pp. 299–327, May 2005.

[10] Z. Duan, Y. Dong, and K. Gopalan, “Dmtp: Controlling spam through message delivery differentiation,” *Computer Networks: The International Journal of Computer and Telecommunications Networking*, vol. 51, pp. 2616–2630, July 2007.

[11] A. Jeckmans, “Computational puzzles for spam reduction in sip,” draft, July 2007.

[12] Z. Zhong, “Throttling outgoing spam for webmail services,” in *In Conference on Email and Anti-Spam*, 2005.

[13] R. Rivest, A. Shamir, and D. Wagner, “Time-lock puzzles and timed-release crypto,” Cambridge, MA, USA, Tech. Rep., 1996.

[14] A. Jeckmans, “Practical client puzzle from repeated squaring,” Tech. Rep., August 2009. [Online]. Available: http://essay.utwente.nl/59133/

[15] B. Laurie and R. Clayton, “”proof-of-work” proves not to work,” in *In Proceedings of the The Workshop on Economics and Information Security*, May 2004.

[16] R. Segala, J. Crawford, and B. L. J. Kephart, “Spamguru: An enterprise anti-spam filtering system,” in *In Proceedings of the First Conference on E-mail and Anti-Spam (CEAS)*, 2004.