Inexpensive Email Addresses
An Email Spam-Combating System

Aram Yegenian and Tassos Dimitriou

Athens Information Technology,
19002, Athens, Greece
aramyegenian@alumni.cmu.edu,tdim@ait.edu.gr

Abstract. This work proposes an effective method of fighting spam by developing Inexpensive Email Addresses (IEA), a *stateless* system of Disposable Email Addresses (DEAs). IEA can cryptographically generate exclusive email addresses for each sender, with the ability to re-establish a new email address once the old one is compromised. IEA accomplishes proof-of-work by integrating a challenge-response mechanism to be completed before an email is accepted in the recipient's mail system. The system rejects all incoming emails and instead embeds the challenge inside the rejection notice of Standard Mail Transfer Protocol (SMTP) error messages. The system does not create an out-of-band email for the challenge, thus eliminating email backscatter in comparison to other challenge-response email systems. The system is also effective in identifying spammers by exposing the exact channel, i.e. the unique email address that was compromised, so misuse could be traced back to the compromising party. Usability is of utmost concern in building such a system by making it friendly to the end-user and easy to setup and maintain by the system administrator.

Key words: Email spam, disposable email addresses, stateless email system, proof-of-work, email backscatter elimination

1 Introduction

Unsolicited bulk emails, or more commonly known as spam, reached 180 billion emails of total emails sent per day in June 2009 [1]. The cost of handling this amount of spam can be as much as 130 billion dollars [2]. These costs are borne by the email receivers and the enterprises in the form of lost productivity and/or taxing the network for unproductive bandwidth.

Existing anti-spam systems can be successful in combating spam until the moment spammers adapt and find a way around them. Other anti-spam systems add cost to each sent email such that spam would be economically infeasible. However until now these efforts have not seen wide scale adoption and it is believed that they will not be effective in combating spam even if deployed on a large scale. The merits and costs associated with each such method are outlined below:

Email Filtering: Email filters examine emails *after* they have been accepted into the local mail system to find patterns of a spam email. The costs [2] involved with this solution include: *False negatives*, *False positives*, *Help desk running costs* for handling complaints about spam or lost emails, and *Storage/Processing costs* since i) emails have to be accepted into the local queue, and ii) have to be processed by the filters, consuming both CPU and IO resources.

Domain Name System Blacklist: Email servers consult a published list of the IPs of suspected servers and determine if an incoming email is originating from a spam generating server [3]. However, the operators of such systems are criticized of being too aggressive, listing legitimate email servers by mistake or on purpose, or lacking clear guidelines in listing and de-listing a server. Any of these actions would destroy the trust that is required to make such a system useful.

Greylisting: This is a method of fighting spam by temporarily rejecting incoming emails from a server that the mail server does not recognize [4]. Properly functioning mail servers would retry delivery of the email contrary to spammers who can not afford wasting the time to retry delivery of a failed email. The main advantage of greylisting is that the incoming email is never accepted into the local mail system.

Proof-of-Work systems: There have been proposals to attach cost to each email sent. Hashcash [9] and Microsoft's Penny Black [10] are two such systems by which an email sender would expend an amount of CPU time to calculate a computationally-expensive puzzle before an email is accepted. This, however has two implications: i) Computer processing power varies immensely between different machines. ii) Spammers have access to botnets, which can be used as computing clouds, or rather spam generating clouds. In this work, we argue that proof-of-work functions could be very effective in fighting spam; however, having challenges based on image recognition instead of computational effort would make it harder for spammers to automate email delivery.

Our Contribution: Email addresses are expensive in the sense that they are tied to a person's identity. It would be impractical for people to change their email address once it is compromised since that would entail informing all of their contacts, updating their business card and updating their web site in order to reflect the change.

IEA, Inexpensive Email Addresses, is a system that successfully uncouples a person's identity from their email address. An *exclusive* email address is cryptographically generated per sender that is used instead of a regular email address. However this exclusive email address can be easily disposed of and re-established once it has been compromised.

The system extends the use of Disposable Email Addresses (DEAs) by integrating a proof-of-work function into the standard mail protocol. The username part of the IEA system serves as a publicly known token whereby an email sender would query to generate an exclusive email address, *provided* that a proof-of-work function is solved before revealing that exclusive address. Meanwhile incoming emails are *rejected* during the SMTP session header transmission, ensuring that emails are not bounced at a later stage, something which would constitute email backscatter. IEA has been developed as a *proxy* server, relaying messages to the SMTP server once all validity checks has passed, thus making the system easily *pluggable* to existing e-mail infrastructure. Finally, the system is user friendly, since creating new disposable email addresses is easy and transparent, requiring minor intervention from the email recipient.

The rest of the paper is organized as follows. Sections 2 and 3 unravel the design details of IEA. Implementation details and experimental results can be found in Section 4. Section 5 contains a comparative study of our system against existing work. Section 6 offers a discussion and critique of IEA, while Section 7 concludes the paper.

2 Inexpensive Email Addresses: IEA

Before we delve into the details of the system, we give a high level overview of IEA. IEA makes it easy for a person to publish an email address for the entire world to use but still retain control of what is delivered to their inbox. The username of an IEA user is used as a token to be queried. An email sender would send an email to this token to establish a new and a unique email communication channel for themselves, however, the new custom email address is only revealed *after* going through a proof-of-work process.

Incoming emails are rejected with a notice containing a challenge-response function that, when solved, reveals a customized email address per email sender. The challenge is embedded in the description of the SMTP error message, which is parsed by the sending MTA, eliminating email backscatter. The challenge-response process limits the number of incoming unsolicited emails since proof-of-work is integrated into the mail protocol. If an email address is compromised, a user has the option to “Roll” or “Ban” the email address which disposes that email address and forces any incoming mails to that address to go through a new challenge-response process.

Outgoing emails that are generated by a user of the IEA system are processed by the IEA daemon to replace the original user’s email address by a DEA generated specifically to that recipient. The DEA is then committed to the database of DEA-to-email mappings. Incoming emails using this new DEA are not subject to a challenge-response process, since the IEA user was responsible for initiating the communication channel with the other party.[1]

IEA is transparent to the end-users of the system in the sense that it is compatible with any Mail User Agent (MUA), without any changes needed. Essentially any mail client can be used with IEA, because the IEA daemon is a compliant SMTP server and would handle all emails and perform any required processing transparently. The only case that a user might need to interact with

[1] Some critics of challenge-response email systems point out the counter-intuitive process of forcing an email receiver to go through a challenge-response mechanism when the email initiator is the other party, this is why we have opted to allow incoming emails to go through when the email is *initiated* from the local system.

the system is when a user decides to “Roll” or “Ban” an email address. Currently, we have extended an open source webmail client, SquirrelMail [5], to provide that functionality to the user, however, most popular desktop mail clients, like Mozilla Thunderbird or Microsoft Outlook, can also be extended to have the ability to interact with the IEA system.

2.1 How IEA Makes Use of SMTP

SMTP is the standard protocol used by all mail servers to relay emails across the Internet. SMTP is a simple text-based protocol and has preset error codes to handle abnormal circumstances. The IEA system uses the SMTP error messages to embed the challenge inside it. The following is an SMTP session showing an email sent from `alice@example.com` to `bob@iea_system.com`. Alice has not established proof-of-work yet (the sender’s MTA is signified by text in italics and the lines are numbered for demonstration purposes).

```
1. 220 smtp.iea_system.com ESMTP
2. HELO smtp.example.com
3. 250 Hello smtp.example.com, pleased to meet you
4. MAIL FROM:<alice@example.com>
5. 250 Ok
6. RCPT TO:<bob@iea_system.com>
7. 553 <bob@iea_system.com>: Hello, a custom email address has been created
for you, please resubmit your email to this new email address. Please visit
the following URL to access it:
http://mailhide.recaptcha.net/d?k=01G_n_x4ZFpi4A0gYj6phw bg==&c=C2HSZaHRWBC
vz-9zzfqlsDsZ9Ko8NdH5SXgclfm9QQSy4jAYL T6nv0P7UrK8oMRTiS-iBmyF3_RyGBuIzm-cT
w==
8. QUIT
9. 221 BYE
```

It should be emphasized that at step number 7, the IEA system rejects the email with an SMTP error code, 553 which means “Requested action not taken - Mailbox name invalid” [6], and embeds the challenge inside the description notice. The sending MTA would close the connection since it has encountered an error message, and deliver the notice to the sender. Also note that the *sending MTA never reached the data sending part*, which can be used to deliver huge attachments in an effort to waste bandwidth or to be used as an attack vector.

2.2 IEA Step-by-Step Walkthrough

Now we present a typical use case of a person sending/receiving an email to/from an IEA system user. When `alice@example.com` sends an email to an IEA system user with the email address `bob@iea_system.com`, the sender’s MTA would initiate the SMTP session with the IEA daemon. The IEA daemon system would receive the following email headers:

```
...
From: "Alice" <alice@example.com>
To: "Bob" <bob@iea_system.com>
...
```

At this stage the IEA system checks if the recipient exists in its database and the sender is allowed to use this address, otherwise it checks if the alias used is a valid one by decrypting it. In this case "bob" is a token used to generate a custom DEA, so it should be rejected and a challenge is embedded in the rejection notice. The IEA daemon generates a new DEA as previously discussed resulting in an exclusive email address, much like the following:

```
jTYtOowmrE_omtyfMTNSWrT32gyRR-HT@iea_system.com
```

The IEA daemon then creates a URL that would only reveal the DEA after a CAPTCHA is solved, by using the Mailhide API[13] to encrypt the newly created DEA inside the URL:

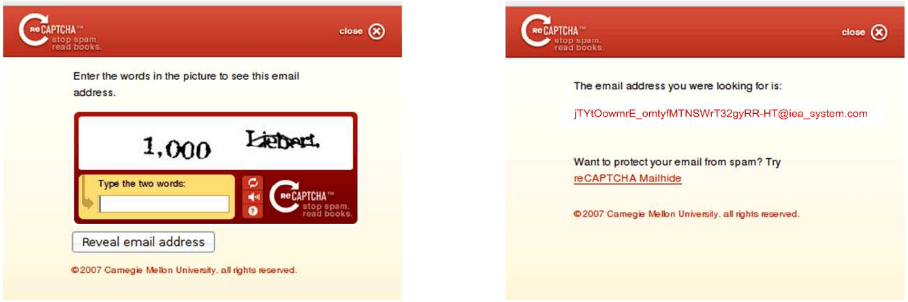
```
http://mailhide.recaptcha.net/d?k=01G_n_x4ZFpi4A0gYj6phwbg==&c=C2HSZaHRWBC
vz-9zzfqlsDsZ9Ko8NdH5SXgclfm9QQSy4jAYLT6nv0P7UrK8oMRTiS-iBmyF3_RyGBuIzm-cT
w==
```

The IEA daemon then embeds this URL in the description part of the rejection notice and delivers it to the sender's MTA and ends the connection. In turn the sender's MTA delivers the rejection notice immediately to the sender, alice@example.com. The sender would receive a message in their inbox from the MTA, the "Mail Delivery Subsystem", stating that the email was not delivered with the subject: "Returned mail: see transcript for details".

Upon opening the email, the following message will be displayed, although the exact message may be different depending on the sender's MTA.

```
...
This is the Postfix program at host iea_system.com. I'm sorry to have
to inform you that your message could not be delivered to one or more
recipients. It's attached below. For further assistance, please send mail
to <postmaster> If you do so, please include this problem report. You can
delete your own text from the attached returned message.
The Postfix program <bob@iea_system.com>: host iea_system.com said:
553 <bob@iea_system.com>: Hello, a custom email address has been created
for you, please resubmit your email to this new email address. Please
visit the following URL to access it:
http://mailhide.recaptcha.net/d?k=01G_n_x4ZFpi4A0gYj6phwbg==&c=C2HSZaHRWB
Cvz-9zzfqlsDsZ9Ko8NdH5SXgclfm9QQSy4jAYLT6nv0P7UrK8oMRTiS-iBmyF3_RyGBuIzm-
cTw==
                   (in reply to RCPT TO command)
...
```

It must be noted that this process is *stateless*, as the DEA is not yet stored in a database. Only *after* the CAPTCHA is solved and an email is resubmitted using this new DEA would the IEA daemon commit the alias into its database. We can clearly see that the email was never accepted in the local mail queue, since the SMTP header transmission was never completed to reach the data transfer stage thus eliminating any possible attacks or email backscatter.



Enter the words in the picture to see this email address.

©2007 Carnegie Mellon University, all rights reserved.

The email address you were looking for is:

jTYtOowmrE_omtyfMTNSWrT32gyRR-HT@iea_system.com

Want to protect your email from spam? Try reCAPTCHA Mailhide

©2007 Carnegie Mellon University, all rights reserved.

(a) (b)

Fig. 1. (a) Mailhide displaying the challenge. (b) Mailhide displaying the DEA.

The sender could now establish proof-of-work by solving the CAPTCHA that is presented to her when she accesses the Mailhide URL (Figure 1(a)). If the sender successfully solves the reCAPTCHA challenge, Mailhide will display the DEA for the sender (Figure 1(b)).

When the sender resubmits the email with the new recipient address, the sender's MTA would reconnect to the IEA daemon to deliver the email, as follows:

```
...
From: "Alice" <alice@example.com>
To: "Bob" <jTYtOowmrE_omtyfMTNSWrT32gyRR-HT@iea_system.com>
...
```

The IEA daemon would check the validity of the alias and decrypt it to reveal the original recipient of the email. The email would only be accepted if the sender information passes the validity checks. The IEA daemon then changes the email headers back to the real recipient's email address, as follows:

```
...
From: "Alice" <alice@example.com>
To: "Bob" <bob@iea_system.com>
...
```

After applying the changes to the email headers then the IEA daemon relays the email to the back-end MTA for delivery to the user's mailbox, `bob`. Since the IEA daemon acts an SMTP proxy and relays emails to the back-end MTA it never stores email locally, therefore extra storage is not required for the proper functioning of the IEA daemon.

If the IEA sender replies to the message it will also go through the IEA daemon, since it is configured as the default SMTP server for outgoing emails.

The IEA daemon processes the headers of the incoming email that is generated from `bob` to `alice`.

```
...
From: "Bob" <bob@iea_system.com>
To: "Alice" <alice@example.com>
...
```

The IEA daemon will first search its database for a DEA associated with `alice@example.com` and use that if available, otherwise it would create a DEA for `alice` and store it in the database. Next the IEA daemon would process the headers by changing the "From" field to the DEA that is associated with `alice`, as follows:

```
...
From: <jTYtOowmrE_omtyfMTNSWrT32gyRR-HT@iea_system.com>
To: "Alice" <alice@example.com>
...
```

This guarantees that when the other party receives the email, the return addresses are correct such that if the receiver is to reply then the correct DEA address is used, and the sender is not subjected to a further challenge-response process.

2.3 Rolling and Banning

An IEA user has a choice of either rolling or banning a DEA once it has been compromised. Rolling a DEA would force only the sender of the email to re-establish a new DEA by going through a proof-of-work process again. A new key (this will become clear in Section 3.2) is generated for that sender and that key is used to generate the new DEA for that sender.

Banning on the other hand would dispose of that DEA for all senders that were allowed to use it and would generate new keys for everyone using that DEA, thereby forcing all the senders of that DEA to go through a proof-of-work process. This would channelize mail communication with all parties, by having the option to only dispose of a DEA per sender, or for everyone that is using it.

3 System Design

3.1 The IEA Sub-Systems

IEA is composed of three sub-systems as illustrated in Figure 2(a). A *daemon* that is responsible for mail reception, mail processing, DEA creation, and submission to a back-end MTA for mail delivery. A *web interface*, which exposes the features of the system to the end-user in a user-friendly manner, and a *database* containing validated DEAs and encryption keys used to generate the DEAs.

The IEA daemon: The IEA daemon acts as a proxy SMTP server which handles incoming and outgoing emails and processes them according to the rules in its database. IEA was designed such that the daemon would be a standalone server, as opposed to be being a mail filter that extends the functionality of the MTA.

This design was chosen for two reasons. First, having a standalone server that uses SMTP to communicate with other MTA servers would make the IEA daemon *inter-operable* with any MTA server running in the back-end. Second, IEA extends the SMTP protocol error message descriptions by embedding the challenge inside the rejection notice that is delivered to a sender's MTA. MTAs do not expose that ability through an API to developers since extending the error descriptions of a mail server is an unorthodox requirement.

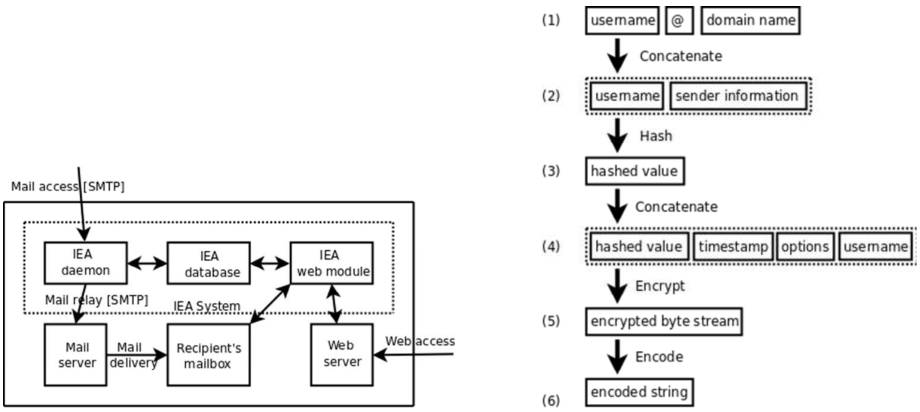
The IEA daemon relays the emails that pass the validity checks to the back-end MTA for final delivery. The validity checks are performed during the SMTP session header transmission, and email destined to invalid DEAs would be rejected before the sending MTA reaches the data transmission stage. This would ensure that no emails are accepted in the local queue before the sender has solved the proof-of-work function.

Regarding incoming emails, the IEA daemon rejects all of them and a challenge is embedded in the rejection notice in response. When the challenge is solved, the new DEA is revealed to the sender and will be stored in a database when the sender resubmits the email using the new DEA. On the other hand, senders who are responding to emails that originated from users of the IEA system *are not subjected* to a challenge-response mechanism and their emails are accepted for delivery.[2] This is accomplished by generating a DEA when an email is originated from the local system that specifies a new external recipient. The new DEA is substituted in place of the user's email address in the "From" field and it is also committed to the database so that further communication with that party does not result in a challenge-response request.

The IEA Database The IEA database contains the DEA-to-sender mapping and their corresponding keys. Since almost 90% of all emails are spam [7], storing a DEA for each incoming email would overwhelm the database in the long run. This can be used as an attack vector by generating a large number of forged email senders and targeting such a system. However, IEA generates a stateless DEA by embedding *inside* the DEA all the data needed to deliver an email. The database is only populated with the new DEA *after* a sender has established proof-of-work.

The database also contains the encryption keys used to generate the DEAs. We distinguish between two keys: i) a *master encryption key* that is used initially per IEA user (say `bob`) to generate all new DEAs for that user, and ii) *roll encryption keys* that get stored in the database only after the user (Bob) has decided to roll an existing DEA.

[2] This is useful when the IEA user wants to generate DEAs on demand, for example to register to a conference or obtain access to a news site. This allows the user to receive emails to that DEA for as long it is desired.



(a) (b)

Fig. 2. (a) IEA sub-systems. (b) DEA generation flow.

The master encryption key is associated with Bob's account and is generated when a new account is *first* created for that user. The key is used to produce the initial DEAs for the conversation between Bob and external correspondents. When `bob` chooses to "Roll" an existing email address, a new encryption key (roll key) is generated for that sender only (say `alice`), a new DEA is generated using that key and `alice` has to re-establish proof-of-work (alternatively, instead of a new key an increasing counter value can be used to generate the new DEA in association with the master key). It is only at this point that this new roll key is committed to the database of DEAs already kept by Bob. Similarly, when a user chooses the "Ban" option, the IEA system generates new keys for all the senders that are using that DEA, and they are required to re-establish proof-of-work.

The IEA Web Module The IEA web module is an interface for users to manage the IEA system. The web module was built as an extension to the popular SquirrelMail webmail client (figure omitted due to space restrictions). However, *any* MUA could be used instead and extended to implement the required features, including desktop email clients like Mozilla Thunderbird or Microsoft Outlook, since the IEA daemon is a fully compliant SMTP server.

The SquirrelMail interface implements the following features:

- *Rolling of a DEA*: A user can choose to tag an incoming email as compromised and "Roll" it. The sender would have to establish a new DEA.
- *Banning of a DEA*: A user can choose to ban a DEA thereby all parties using that DEA would have to re-establish a new DEA.
- *DEA creation*: A user of this system can create a DEA that would be used as an alias to give out to senders or to be used at e-commerce sites. Also the user would have the option to create a DEA that has an *exclusive* user, or a

DEA that allows all senders to deliver emails which is the default (this is used in conjunction with banning above).
- *Specifying an expiration date for a DEA*: This is useful for users who would like to have an email address expire after a certain time has passed.

3.2 The DEAs

A DEA that is generated by the IEA server constitutes an encrypted byte array of the data needed to validate an alias and successfully deliver the email to its intended recipient. The encrypted string is the result of concatenating the username of the recipient with the hashed value of the sender address along with the username of the recipient. The hashed value is used to verify the integrity of the alias at later stages. Each user has a master key that is used to generate the DEAs. The master key is only changed if the user chooses to "Roll" an email address, whereby a new key is generated for that specific sender thus making the old DEA invalid. Figure 2(b) describes how a DEA is generated by the IEA system.

- At first, the domain name part of the recipient email address is stripped away to get the username.
- The hashed value is created by concatenating the sender's email address with the username of the email recipient. MD5 is used to generate the hash value. The hashed value is used to check the integrity of a DEA since we need to verify that an incoming email address is a valid DEA for that specific user.
- The hashed value is then concatenated with the username of the email recipient, a time stamp, and a byte array that contains the option settings for that specific DEA. It is necessary to embed the username of the recipient within the generated DEA because the system is stateless and does not store a DEA-to-username mapping before the challenge is solved; otherwise the intended recipient would be lost. The timestamp is used to determine if the challenge is expired, by default IEA allows a grace period of four days to solve the challenge and resubmit an email using the newly created DEA. The byte array contains option settings (Table 1) that could be used to signify that a DEA is an exclusive one per user, or to verify the domain name of the sender instead of the email address, which is used in the case of correspondence with an e-commerce site.

Table 1. Option settings for a DEA

Options	Values
sender_info_type (default: 0)	0: Sender's e-mail address 1: Sender's e-mail domain 2: mailing list address
dea_type (default: 1)	0: exclusive per sender 1: allow all senders

– The concatenated byte array is encrypted using a master encryption key that was generated *per IEA user* when the user's account was first created. The master encryption key is used to generate all incoming emails per user. Only when the user chooses to "Roll" or "Ban" an email address a new encryption key is generated thus invalidating the old DEA and generating a new DEA for that sender. The encryption scheme used for the IEA system is Blowfish, however any encryption scheme could be used instead. Authenticated encryption, as discussed in [8], was investigated as a faster mode of operation but was not implemented in the current version of the IEA system.
– Finally, the encrypted form is transformed to a string by encoding it using base 64 encoding. It should be noted that we use a safe base 64 encoding that is suitable for the SMTP protocol. Since the characters '+' and '/' are not safe to use with SMTP, they are substituted with '-' and '_' respectively.

4 Implementation

The system was developed and deployed on a virtual machine instance running Linux. The test machine had 64MB of RAM dedicated to it with 128MB of swap space and was powered by an Intel Core 2 CPU running at 2.1GHz.

The prototype IEA system was built using the Python language. The standard Python library `smptd` module was extended to implement the IEA daemon. The IEA system does not require much memory or disk space since it does not accept emails in its local queue. The database to store the generated DEAs and keys was MySQL.

Postfix was used as the back-end MTA to deliver the emails to the user's mailbox and handle outgoing emails. Postfix was setup to run on the same machine as the test machine, although it could be setup to run on a separate machine.

An Apache server was used to run the webmail client. The webmail client, SquirrelMail, is a popular open source client. SquirrelMail is easy to extend and we developed the features required for the IEA system as a plug-in for it.

It should be noted that none of the components used are tightly coupled to the IEA system. Apache, MySQL, Postfix, SquirrelMail and even the operating system, Linux, can be easily replaced with similar functioning components. Although the IEA system was not explicitly designed to be multi-platform or have pluggable architecture, we did rely on standard protocols which make the replacement of components possible.

4.1 Benchmarks

The IEA daemon was benchmarked to measure how many SMTP connections would it be able to handle per unit of time. The IEA daemon was subjected to incoming emails such that it would generate a DEA and create a Mailhide URL for it as we believe that this condition would expose the daemon to the most possible load. The IEA daemon was able to process 252 SMTP sessions *per second*. It should be noted that the running IEA system is a development version, which contains debug code and is *not optimized for performance*.

For comparison reasons we subjected the Postfix server to a similar test. We configured Postfix such that it rejects all incoming emails, as the IEA daemon previously did. The Postfix server was able to process 951 SMTP sessions per second on the same machine. Of course Postfix is a high performance optimized mail server and was configured to reply with just a rejection notice for testing purposes; whereas the IEA daemon is a prototype that had to go through an SQL query, generate a DEA then encrypt the DEA inside a URL. We believe that proper optimization would significantly increase the performance of the IEA system.

While benchmarking the performance of the IEA daemon we noticed that the MySQL server was undergoing increased load. This behavior is understandable since the IEA daemon consults the database to check if the alias already exists as a legitimate alias for a recipient. However, we decided to replace the MySQL server with a lighter database system and to run the benchmarks again to see the difference. SQLite was chosen as a lightweight alternative to MySQL. SQLite is not a standalone client-server SQL server, instead the SQLite library is linked within the Python library thus the overhead of connecting to a back-end SQL does not exist. The benchmarks were performed again, and the IEA daemon was able to process 294 SMTP sessions per second. That constitutes a 16% increase in performance versus using the MySQL server. This further proves that proper optimization could be made to the IEA daemon and better performance would be achieved.

The encryption and decryption speed of DEAs by the IEA daemon was also measured. Test results showed that roughly 6950 encryption and 8720 decryption operations can be performed per second, respectively. This indicates that DEA generation has very little impact on the system as a whole. The validation speed of a DEA was also measured. MD5 was used to create the hash values to verify the integrity of the DEA. Test results showed that roughly 414900 integrity check operations can be performed per second. This also indicates that DEA validation has no impact on the system as a whole.

5 Related Work

In this section, we compare IEA with existing approaches to limit spam. The IEA system borrows some characteristics from these works, however, it refines them and adds some of its own to create a unique and more viable spam-limiting system. What particularly distinguishes IEA is that it does not generate *out-of-band* emails in the challenge-response process, it is not susceptible to a *Denial-of-Service (DoS)* attack and cannot be used as an attack vector to generate spam and/or *email backscatter*. We hope this list of desired properties will prove the viability of IEA and stimulate further research in the area.

Hashcash [9] and the Penny Black Project [10] propose the inclusion of cryptographic puzzles in the SMTP protocol to limit the speed at which spammers deliver emails. However a proof-of-work challenge that relies on computer intensive puzzles will make spammers use botnets to generate and deliver spam.[3] Another problem with this approach is the need for all MUAs and MTAs to be compatible with the protocol. The IEA system uses a human solvable puzzle rather than a cryptographic challenge. Also, the IEA system does not require that the MTA or MUA on the other end to be modified, since it relies on human effort to solve the challenge and it is delivered using the standard SMTP protocol.

Similarly, in [12], the authors propose a system in which a challenge contained inside a URL is distributed alongside one's email address, to be solved prior to sending the email. One critique of this system is that emails that lack the solution to the challenge are silently discarded, which could mean that two users of such a system would get into a deadlock as they do not get any notices about it (the authors, however, propose an alternative solution whereby all MTAs must be modified to cater for discarding emails). In contrast, IEA delivers the challenge through the sending MTA, thereby being compatible with the current mail protocol while also being instantaneous.

Mailhide [13] is a subset of Carnegie Mellon's reCAPTCHA project. Mailhide is very effective in hiding an email address from spambots that scan the web for addresses, however, once the email address has been compromised there is no way back to a safe, no-spam state. The IEA system incorporates Mailhide to deliver the challenge that contains the newly generated DEA. be directly connected to the Mailhide servers. It should be noted, however, that the IEA system could be easily modified to use *any* CAPTCHA generating component.

Rolling Email Address Protocol or REAP [14], is based on a very interesting concept called "Rolling". Rolling basically means that a user has the capability to dispose of an email address and agree upon a new one with the sending party once the current one is compromised and starts receiving spam. The problem with REAP, however, is that it requires too much *manual* intervention from both parties. Therefore what we propose is once a user tags an email address as compromised, all incoming emails to that alias be challenged and a new DEA be created instead. This would channelize the email communication such that each sender would have a unique DEA for itself.

Tagged Message Delivery Agent [15], is an open source mail system that employs a challenge-response function to reduce spam. Three main drawbacks of this system include the generation of an out-of-band email and hence the possibility of email backscatter, the storage of emails into the local queue which

[3] Laurie and Clayton in [11] presented an economic study of the effectiveness of using cryptographic puzzles to fight spam. In their paper they conclude that spammers will not be affected by these systems, instead legitimate email senders would be harmed the most. However, they suggest puzzles based on human effort, CAPTCHAs, as a viable solution to attaching a "cost" to emails.

may be used as DoS attack vector, and finally the possibility of recovering the challenge (simply a random character array) by spambots.

A "Remailer" component has been presented in [16] that automatically creates new aliases for incoming emails and either bounces or relays emails based on acceptance criteria for that alias in order to fight spam. The Remailer system is responsible for storing all incoming emails that are undeliverable in a special mailbox until the Remailer goes through them and bounces invalidated emails. This, however, opens the system for abuse by senders with forged email addresses to send emails to non-existing aliases which in effect would be bounced by the Remailer, generating email backscatter. The IEA system never accepts any incoming emails that could later be bounced by the back-end MTA. Validity checks are performed during the SMTP session header transmission, and emails are rejected *before* the sender is allowed to transmit the actual data part of the message.

MIDEA [17], proposes a method of eliminating lookup tables for DEA management on the server side by embedding the data needed for the DEA management into the email address. The paper mainly considers limited functionality devices, as mobile phones, to send emails, however it does not delve into spam control.

In addition to the characteristics outlined above, IEA achieves a unique set of properties that distinguish it from past work. These are summarized below:

- *Stateless System:* IEA is a stateless system by avoiding storing a sender's email address at the beginning of establishing a DEA so as to conserve resources and avoid DoS attacks. The DEA is effectively used as a cookie to distinguish a sender that has solved the challenge. Only well intentioned users would go through a challenge-response process, and the DEA would be committed to a database *after* the new DEA has been used for the first time.
- *Email Backscatter Elimination:* Email Backscatter [18, 19] is a side-effect of spam emails that get accepted in a mail system having *forged sender* identity then bounced by the email server back to the forged address. This happens since mail filtering checks are done *after* an email is accepted into the queue. IEA was designed to eliminate backscatter by rejecting all incoming emails. The challenge is delivered to the sender by simply embedding it inside the description part of the rejection notice of SMTP error messages.
- *Traceability:* When a DEA starts receiving spam, IEA allows the user to dispose of that particular DEA and tag it as compromised. This would make the system start the challenge-response mechanism on the compromised DEA, ensuring all parties that were using it would establish a new one. If one of these new DEAs is also compromised, that DEA would expose the spamming party since each new party has established a new unique DEA for its own channel.
- *Guaranteed Delivery:* Current challenge-response mail systems generate an out-of-band email that contains the challenge-response function in response to an incoming email. Thus the mail server could be abused to generate backscatter as described before. Also the challenge email is not guaranteed to be delivered since it is effectively an unknown server to the sending mail

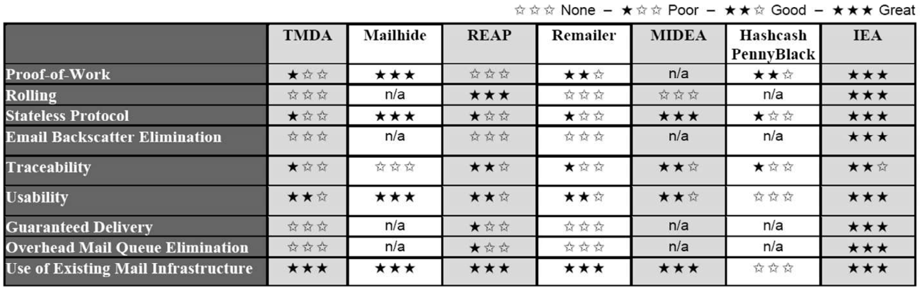
server. More importantly two users of such a system could end up in a deadlock, where each user's Mail Transport Agent (MTA), sends a challenge email in response of the other party's challenge email.
IEA was designed to reject all new emails and instead embed the challenge function inside the SMTP rejection notice during the mail protocol's header transmission. This would guarantee delivery of the challenge-response notice to the sender's inbox since it is handled by the MTA of the sending party and it is the only authority that can successfully deliver emails into a user's inbox without going through spam filters. Also since the email is rejected and the challenge-response function is embedded in the rejection notice, there is no fear of entering a deadlock between two users of this system.

- *Overhead Mail Queue Elimination:* Current challenge-response mail systems accept all incoming emails to the local mail queue and store the email locally until the generated challenge-response is solved. This constitutes wasted storage and could also be used to attack a mail infrastructure by *mail bombing* the mail server with numerous emails having large attachments..
IEA avoids this problem by rejecting any incoming email that is not a valid DEA, *before* the actual data of the message is transmitted.
- *Usability:* The system limits spam in a non-obtrusive way for the end-user. The system is user friendly, and creating new disposable email addresses is easy and transparent. Switching to a new disposable email address, if the original were compromised, requires minor intervention from the email recipient. Most of that burden is shifted to the sender of the email.
- *Ease of Implementation and Deployment:* The IEA system easily integrates with existing email infrastructure. IEA has been developed as a *proxy server* that operates as a front-end and only relays emails back to the SMTP server once all the validity checks have been passed, therefore making this system easily pluggable into an existing mail infrastructure.
Finally, the system was designed such that emails are kept on the sender's side as much as possible, so an email is not accepted in the local mail queue unless the sender has solved the challenge function. This would relieve the local mail system, and the administrator, from the resources needed for storing the mail in the local queue.

A summary of these properties and a comparison with existing systems is shown in Figure 3. It should be noted that some systems, like TMDA and Remailer, generate backscatter and do nothing to eliminate it.

6 Discussion and Critique of the IEA System

As previously discussed, IEA is a challenge-response system where the challenge is delivered to the sender by embedding it into the description part of the SMTP error message. Strictly speaking this does not follow the SMTP specification since the error codes were implemented to return error messages as opposed to be used in a challenge-response mechanism. However, this was chosen to overcome the



☆☆☆ None – ★☆☆ Poor – ★★☆ Good – ★★★ Great

	TMDA	Mailhide	REAP	Remailer	MIDEA	Hashcash PennyBlack	IEA
Proof-of-Work	★☆☆	★★★	☆☆☆	★★☆	n/a	★★☆	★★★
Rolling	☆☆☆	n/a	★★★	☆☆☆	☆☆☆	n/a	★★★
Stateless Protocol	★☆☆	★★★	★☆☆	★☆☆	★★★	★☆☆	★★★
Email Backscatter Elimination	☆☆☆	n/a	☆☆☆	☆☆☆	n/a	n/a	★★★
Traceability	★☆☆	☆☆☆	★★☆	★☆☆	★★☆	★☆☆	★★☆
Usability	★★☆	★★★	★★☆	★★☆	★★☆	☆☆☆	★★★
Guaranteed Delivery	☆☆☆	n/a	★☆☆	☆☆☆	n/a	n/a	★★★
Overhead Mail Queue Elimination	☆☆☆	n/a	★☆☆	☆☆☆	n/a	n/a	★★★
Use of Existing Mail Infrastructure	★★★	★★★	★★★	★★★	★★★	☆☆☆	★★★

Fig. 3. Summary of features per system.

problem of generating an out-of-band email or generating email backscatter in response of an incoming email. It must be noted that some mail providers also use the description part of the error code to deliver information, notably Yahoo! Mail servers [20] specify a URL to be visited in the description part of the error message in an effort to enforce Greylisting [4].

IEA incorporates proof-of-work into the standard SMTP protocol, but unlike Hashcash and the Penny Black Project IEA uses the existing mail infrastructure (MTAs and MUAs) to achieve its goals and does not require the other end to be upgraded or changed to fully utilize the protocol.

IEA uses the sender's email address that is sent during the SMTP header transmission as a token to be embedded in the generated DEA. IEA could be criticized as relying on information that can easily be forged to create this token. However, it must be noted that IEA does *not* generate an email using a forged email address which would constitute email backscatter. Instead, the challenge is immediately delivered to the sender's MTA and does not rely on the return path of the MTA or the sender to be correct: the challenge will be delivered to the sending party whether the sender is forged or not. IEA could also make use of frameworks like [21, 22] to identify a sender. Although it is not strictly necessary for the successful operation of IEA, it could be used to lessen the load on IEA by identifying the sender before the IEA system process the sender information.

A critical note about IEA is the ability to sign up in mailing lists, or automated systems which might not have a human operator to solve the challenge. Users who are using the IEA system can generate a DEA for specific senders that would pass through to their mailbox without having to go through the challenge-response process, as in the case where a sender wants to send a broadcast email (say) to a class of 100 students.

The inclusion of a challenge-response mechanism in the SMTP protocol whereby proof-of-work is established before accepting emails slows down spammers crippling their operation. However, it will also slow down mail delivery for legitimate users, until they solve the CAPTCHA. Users have come to expect a near-instantaneous operation of email, although email was never designed to function like that. Email was designed to be relayed across servers until it reaches its destination without any specification in the protocol for instantaneous delivery. Some users would object to the delay exhibited by the IEA system, but we believe that users would accept the idea of a little delay knowing that their inbox would be clear of spam, and they would not spend time on filtering their legitimate emails from spam.

7 Conclusions and Future Work

In this work we proposed Inexpensive Email Addresses (IEA), a *stateless* system of Disposable Email Addresses (DEAs) that can be used to channelize email communication in order to fight spam. A challenge-response mechanism is applied to the SMTP protocol such that proof-of-work is established *before* emails are accepted into the local queue and delivered to the mail recipient. DEAs are cryptographically generated per sender and can be used to trace compromising parties.

During the design of the IEA system similar systems were studied and their drawbacks revealed in an effort to design a system that does not generate *email backscatter* or that is vulnerable to *Denial-of-Service* attacks. We believe that decoupling a person's identity from their email address would make establishing and disposing email addresses an effective mechanism to fight spam.

The system we developed uses the description notice of SMTP error messages to embed the challenge, thereby using *existing* mail infrastructure to achieve better spam control. We believe that subjecting mail senders to proof-of-work before accepting an email is better than traditional mail filters that could generate false negatives and/or false positives. A challenge-response system would stifle the operation of spammers by increasing the cost and slowing down the effectiveness of sending mass mail.

There are, however, certain possible expansions and works that can be added to the system to enhance its functionality. Currently the IEA prototype is tightly coupled to the email domain of the user; a user has to have a local mailbox on the IEA system server. A more general solution would have the ability to decouple a user's domain from the IEA system; this way any user at any domain could sign up to an IEA service and an IEA username could be created for that user that acts as a proxy and relays valid emails to the subscriber's address. This would make the system more viable for commercial deployment, providing it as a service for any email user.

Desktop mail applications could be extended to have the features necessary for the IEA system that are currently implemented by the webmail client. Having those options inside the desktop mail application would also make it easier for the users to fully utilize the system.

References

1. Messaging Anti-Abuse Working Group (2008). Email Metrics Program: The Network Operators' Perspective. Report #11 First and Second Quar-

ter 2009. Retrieved Oct. 27, 2009 from http://www.maawg.org/sites/maawg/files/news/MAAWG_2009-Q1Q2_Metrics_Report_11.pdf

2. Ferris Research (2009). Industry Statistics. Retrieved Oct. 27, 2009 from http://www.ferris.com/research-library/industry-statistics
3. William K. Cole (2007). DNS-Based Lists: an overview. Retrieved Nov.$\tilde{5}$, 2009 from http://www. scconsult.com/bill/dnsblhelp.html#2-7
4. E. Harris (2003). The Next Step in the Spam Control War: Greylisting. Retrieved Oct.$\tilde{2}$0, 2009 from http://projects.puremagic.com/greylisting/whitepaper. html
5. SquirrelMail. SquirrelMail - Webmail for Nuts!. Retrieved Sept. 15, 2009 from http://www.squirrelmail.org/
6. RFC-1893 (1996). Enhanced Mail System Status Codes. Retrieved Sept. 10, 2009 from http://www.ietf.org/rfc/rfc1893.txt
7. Symantec (2008). The State of Spam A Monthly Report, October 2008. Retrieved Sep. 4, 2009 from http://eval.symantec.com/mktginfo/enterprise/other_resources/b-state_of_spam_report_10-2008.en-us.pdf
8. P. Rogaway, M. Bellare, and J. Black (2001). OCB: A block-cipher mode of operation for efficient authenticated encryption. In *ACM TISSEC*, Nov, 2001.
9. A. Back (2002). Hashcash - a denial of service counter-measure. Retrieved Oct. 10, 2009 from http://hashcash.org/papers/hashcash.pdf
10. Microsoft (2003). The Penny Black Project. Retrieved Oct.$\tilde{1}$0, 2009 from http://research.microsoft.com/research/sv/PennyBlack/
11. B. Laurie and R. Clayton (2004). 'Proof of work' proves not to work. In *Workshop on Economics and Information Security*, Minneapolis, MN, May, 2004.
12. Rodrigo Roman, Jianying Zhou, and Javier Lopez. An Anti-Spam Scheme Using Pre-Challenges. In *Computer Communications*, 29(15):2739–2749, Elsevier, September 2006.
13. Mailhile, Carnegie Mellon University. reCAPTCHA Mailhide: Free Spam Protection. Retrieved Sept. 30, 2009 from http://mailhide.recaptcha.net/
14. J.M. Seigneur and C.D. Jensen (2003). Privacy Recovery with Disposable Email Addresses. In *IEEE Security and Privacy*, vol. 1, no. 6, pp. 35–39, Nov., 2003.
15. J. Mastaler (2003). Tagged Message Delivery Agent (TMDA) Homepage. Retrieved Sep. 30, 2009 from http://www.tmda.net/
16. P. Gburzynski and J. Maitan (2004). Fighting the spam wars: A remailer approach with restrictive aliasing. *In ACM Transactions on Internet Technology (TOIT)*, vol. 4 , Issue 1, pp./ 1–30, Feb, 2004.
17. D. Ochi (2007). MIDEA: Management of Disposable E-Mail Addresses for Mobile Systems. In *International Symposium on Applications and the Internet Workshops (SAINTW'07)*, 2007.
18. Postfix. Postfix Backscatter Howto. Retrieved Sept. 31, 2009 from http://www.postfix.org/BACKSCATTER_ README.html
19. S. Frei, G. Ollmann and I. Silvestri (2004). Mail DDoS Attacks through Non-Delivery Messages. Retrieved Oct. 24, 2009 from http://www.techzoom.net/papers /mail_non_delivery_notice_attacks_2004.pdf
20. Yahoo. 421 Message temporarily deferred - [numeric code]. Retrieved Oct. 17, 2009 from http:// help.yahoo.com/l/us/yahoo/mail/postmaster/errors/
21. Sender Policy Framework (2008). SPF: Project Overview. Retrieved Oct. 17, 2009 from http://www.openspf.org/
22. DKIM.org. DomainKeys Identified Mail (DKIM). Retrieved Oct. 17, 2009 from http://www.dkim.org/