Curbing Junk E-Mail via Secure Classification

E. Gabber M. Jakobsson Y. Matias* A. Mayer

Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974
{eran, markusj, matias, alain}@research.bell-labs.com

Abstract. We introduce a new method and system to curb junk e-mail by employing *extended e-mail addresses.* It enables a party to use her (core) e-mail address with different extensions and consequently classify incoming e-mail messages according to the extension they were sent to. Our contributions are threefold: First, we identify the components of a system that realizes the concept of extended e-mail addresses and investigate the functionality of these components in a manner which is backwards compatible to current e-mail tools. Secondly, we specify an adversarial model, and give the necessary properties of extended e-mail addresses and of the procedure to obtain them in the presence of the adversary. Finally, we design cryptographic functions that enable realizing extended e-mail addresses which satisfy these properties.

1 Introduction

As more and more people rely on e-mail for daily communication, both for their work and personal use, it becomes increasingly important to sort and classify incoming e-mail. Classification allows users to treat different classes of messages in different ways, e.g., store in different mail folders or delete without inspection. The need for such classification is becoming painstakingly apparent, as e-mail is not only becoming more *used*, but also more *abused*. The most widespread example of abuse is mass mailing of unsolicited e-mail messages based on address lists collected from various sources. The act of sending this "junk e-mail" is called "spamming", and it is now threatening to thwart legitimate e-mail. Since spamming is virtually free, whereas its current methods of prevention are time consuming and expensive, it is becoming alarmingly clear that an inexpensive type of protection must be made available, or both the Internet and individual e-mail accounts will turn into a giant digital traffic jam. However, as backwards compatibility is necessary for a resource as distributed as the Internet, the number of possible solutions are clearly limited.

In this paper, we focus on the problem of how to prevent spamming, which can be viewed as a binary classification problem: a message is either classified

* Also with the Department of Computer Science, Tel-Aviv University, Tel-Aviv 69978 Israel; Email: matias@math.tau.ac.il.

as a legitimate e-mail or as spam. We introduce an appropriate adversarial setting, and provide efficient and inexpensive solutions for secure classification of spamming. Our solution easily extend to more general classifications.

1.1 The Problem of Spamming

Spamming is facilitated by the following facts:

- It is fairly easy and inexpensive to obtain a list of valid e-mail addresses and to use it without the corresponding user's consent. Many people give their e-mail address to Web sites where they open accounts. Newsgroups and home pages are other good sources for collection of a large number of valid e-mail addresses.
- The recipient cannot easily distinguish between a personal e-mail message and an unsolicited spam massage. There is no characteristic envelope or sender's address that facilitates easy recognition of spam. The recipient must open the message and read it before deleting it. Spam messages clog the recipient's mailbox and necessitate time-consuming examination prior to removal.
- The cost of generating a relatively large number of spam messages is very low. Sending one million copies of a message is almost as easy sending one thousand copies.

1.2 Current Anti-Spamming Tools

All current tools use one or a combination of the following three methods for detecting and removing spam:

- *Source Address Filtering*: Collect a blacklist of known spammer e-mail addresses. Recognize these *source addresses* and remove their messages automatically. However, most spammers disguise their true address by various means and frequently "mutate" among various bogus return addresses.
- *Keyword Filtering*: Collect a blacklist of keywords often found in the subject-line or body of junk e-mail messages. Recognize these keywords and remove those messages automatically. This has been a somewhat effective method, since spam messages up to now are often characterized by a limited vocabulary. However, automatic removal of messages based on keywords is heuristic at best. It may eliminate some valid messages. Furthermore, spammers have started to adapt to such filtering by changing the vocabulary of their messages, especially after blacklists of keywords become public knowledge.
- *Address Change:* This is a tedious, last-resort action, appropriate when the user's e-mail address is constantly overloaded with spam messages. The corresponding user has to notify all potential senders, from whom she expects to receive e-mail.

1.3 Our Results

We introduce a method to combat junk e-mail via a new kind of e-mail address, termed an *extended e-mail address*. It enables a user to use her core (current) e-mail address with different extensions and consequently classify incoming e-mail messages according to the extension they were sent to. In particular, a user can classify e-mail sent to a particular extension as *spam*. We show how a party (the initiator) can obtain a valid extension from another party (the receiver) via a *handshake* protocol, whereas junk mailers are deterred from doing the same by introducing a "cost" for each handshake.

Our contributions are threefold. We identify necessary components and their functionality for a system to realize the concept of extended e-mail addresses. We specify an adversarial model and necessary properties of extended e-mail addresses in the presence of such an adversary. We then discuss how to realize the handshake by introducing a "cost" for the initiator of the request. Finally, we show a possible realization of the cryptographic functions generating extended e-mail addresses and the handshake. The system may be implemented on top of existing e-mail tools and services. It does not require public key infrastructure. Moreover, the system may be implemented by "agents" that run on behalf of initiators and receivers, so that the operation of the system (beyond the actual sending and receiving of (non-junk) mail) can be made transparent to users.

Organization: Section 2 presents a high-level overview of our approach. In Section 3, we describe the handshake protocol and introduce the main components of our system. Section 4 describes alternate simplified handshake protocols. Section 5 gives the requirements for the address extensions and a generating function, and Section 6 covers the cost function for handshakes. Finally, Section 7 concludes by showing that our system effectively deals with the facts that currently facilitate spamming (as mentioned in Section 1.1).

2 Overview of our Solution

Currently, most e-mail users have a *very small number* of e-mail addresses. For example, one at the office and one for private use with an ISP at home. In contrast, the principle behind extended e-mail addresses is that each user has *many* e-mail addresses. Possibly, as many as different groups or entities the user is interacting with. Furthermore, a recipient of such an address cannot guess an extended address belonging to the same sender and destined for a different group.

Let Alice be our exemplary e-mail user. Alice wants to communicate with Bob via e-mail, and wants to register at a web-site `www.crook.com`, which requires her to give a valid e-mail address. Finally, Alice wants to post to a newsgroup. Bob receives e-mail from Alice with an extended return address `Alice+xV78Yjklp9@company.com`, whereas the folks at `www.crook.com` will receive the return address `Alice+hdfsjg85nK@company.com`. Alice's address will appear plainly as `Alice@company.com` in her newsgroup posting. We call

Alice@company.com a *core address* and xV78Yjklp9 an *extension*. Subsequently, www.crook.com sells the obtained address to a spammer. As soon as Alice gets her first junk-mail message from that spammer, she can *classify* the address she gave to www.crook.com as "spam". This action is called *address revocation*, and it does not affect Alice's communication with Bob or with any other e-mail user or Web site. Furthermore, www.crook.com only knows a revoked e-mail address and cannot guess any other valid extended e-mail address of Alice. Likewise, an arbitrary newsgroup reader will not know a valid extension of Alice's. We note that Alice does not accept messages to her core address; senders (initiators) are requested to obtain a valid extension first by performing a handshake.

Another possible scenario is that Bob inadvertently leaks Alice's address to a database, which gets into the hands of spammers. Alice has the option of *binding* the address Alice+xV78Yjklp9@company.com to Bob, by classifying all messages sent to the extension xV78Yjklp9 as spam, unless the (recognized) sender is Bob.

In order for extended addresses to be of practical use, Bob (the initiator) must have a way to obtain a valid extended address of Alice's (the receiver). For this *handshake* to be effective, it must involve a procedure acceptable for Bob, but unacceptable to a spammer. One possibility for Alice is to ask Bob for a valid return address where she can send the extension. Many spammers do not reveal their real e-mail address. A more sophisticated way is to force Bob to incur *a computational cost* by performing a CPU intensive computation on his machine. Bob might be perfectly willing to "pay" a small amount, but a spammer intending to send a message to a million users, might not. Once the receiver verified that the initiator has actually incurred the cost, she provides a new *extended e-mail address* to the initiator. The initiator can use this address for further communication with the receiver, as long as the receiver does not revoke this address. Thus, the cost of obtaining the extended address is amortized among multiple messages.

Related Work Extended e-mail addresses are similar in appearance to the e-mail addresses generated by the Andrew Message System (AMS) [BT91], which uses addresses of the form user+text@domain. AMS classifies incoming messages by executing user-supplied programs. Extensions serve a purpose similar to the target revocable email addresses generated by the LPWA proxy [LPWA97]. The system generates such alias e-mail addresses on the user's behalf within the context of Web browsing. The filtering is done via separate tools, such as the filtering functions of popular client mail programs (see http://lpwa.com:8000/filter.html). Extended e-mail addresses are also related to electronic mail channel identifiers (see [Hall98]). Channels, like extended addresses, can be used to block unwanted mail. The channel identifiers contain an indication of the policy for handling incoming mail on this channel. However, individuals who adopt a channelized email system might find themselves receiving significant amounts of spam on public channels but be unwilling to revoke those channels because they are also used for unanticipated, but desired, correspondence. The idea of using computational cycles as "cost", an option how for implementing the handshake in our system, was considered by Dwork and Naor [DN92] and Franklin and Malkhi [FM97] in different settings.

A recent paper by Cranor and LaMacchia [CL98] examines the spam problem and discusses several of the above approaches in more detail.

3 System Operation and Major Components

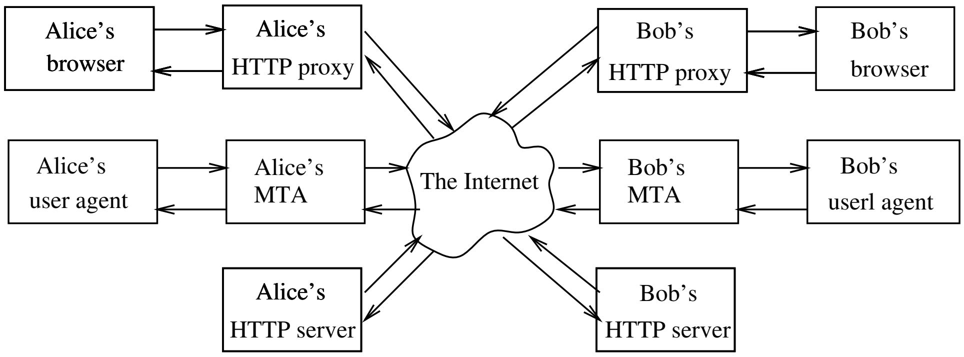

Fig. 1. Basic Communication Structure

In this section we describe the operation of the proposed scheme in some detail. We assume that both senders and receivers use an unmodified *user-agent* for composing and viewing mail messages (e.g., Eudora Pro, Netscape Messenger, etc) and that the actual delivery of the mail is accomplished by a *mail transfer agent* (MTA). The mail agent is typically located on the user's machine, while the MTA might be located on a firewall, Internet access point, etc. A user interacts with a *browser* to visit the Web. That traffic is routed via an *HTTP proxy*, which is typically located at the same place as the MTA; see Figure 1. In addition, the user employs an HTTP server, also often located at the same place as the MTA. The HTTP server allows the user to control the activities of the system and allows external initiators to request an extended address manually. We proceed by presenting in turn the *Extension Generator Module*, the *Destination Lookup Module*, the *Message Receiver Module*, the *Handshake Module*, and the *State Information Database*. We describe the functionality of each logical "module" and point out where a module might physically reside.

The system could be employed either in a fully automatic mode, in which all parties run the entire set of modules, or in a semi-automatic mode, in which some parties use an unmodified MTA without any of the afore-mentioned modules. In the fully automatic mode, the initiator's modules can obtain a valid extended address from the receiver without any interaction with the user. In the semi-automatic mode, the initiator must obtain the extended address from the receiver manually.

The following description should be read in conjunction with Figure 2, which depicts the message flow for obtaining an extended address in the fully automatic mode.

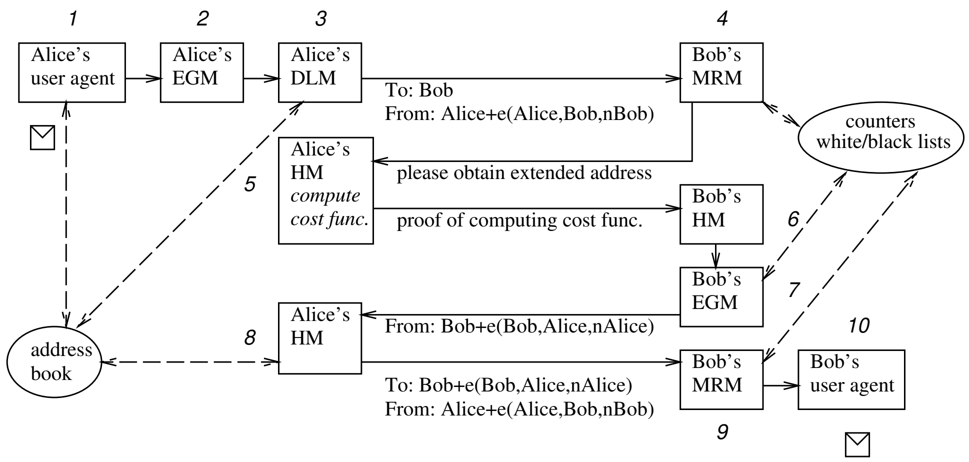

Fig. 2. Message Flow in the Fully Automatic Mode

3.1 Extension Generator Module (EGM)

The EGM computes the appropriate extended address a whenever the e-mail user A (Alice) registers at a Web-site, or sends an e-mail message to another party (Bob), or to a group of parties (Bob and Cathy) or subscribe to a mailing list. The EGM is illustrated in steps 2 and 7 of Figure 2.

When Alice sends a message to Bob, her "From"-address will be calculated as `alice+`$e(Alice, Bob, n_{Bob})$`@company.com`, where e is a function generating the appropriate extension. e takes as input Alice's and Bob's core addresses (e.g, `alice@company.com` and `bob@university.edu`) and a counter (state information) n_{Bob}, whose use is related to the revocation and immediate reissuing of a valid extension, and which will be elaborated on later. The specification of e is given in Section 5. The EGM updates Alice's state information database, as explained in Section 3.7. Bob can forward a message from Alice to a third party, Charlie. Charlie may communicate directly with Alice using her extended address in the forwarded message.

When Alice sends a message to both Bob and Cathy, the mail transfer agent duplicates the message body, and hence both Bob and Cathy will receive the same `From`-address. EGM will generate a single extension $e(Alice, Bob, n_{Bob})$, where Bob is the first recipient in the group. Both Bob and Cathy may use the same reply address (extension) in order to communicate with Alice.

Similarly, the EGM needs to compute extended addresses when Alice registers at a Web-site, which asks her to give an e-mail address. The EGM provides a valid extended address `alice+`$e(Alice, domain(site), n_{domain(site)})$`@company.com`.

When Alice posts to a Usenet newsgroup, it is a prudent choice that only her core address will be visible in her posting. When a reader of a newsgroup wants to send Alice e-mail directly, he will have to get a valid extension via a handshake (as described in Section 3.4). Hence, the EGM will simply forward Alice's core address to the MTA.

3.2 Destination Lookup Module (DLM)

The destination lookup module replaces the destination's core address (e.g. `bob@university.edu`) by its latest known valid extended address using the information stored in the *address book*. The DLM is needed only when Alice's user agent cannot access the address book directly to perform this lookup. The DLM does nothing if the destination address is an extended address or if the address book does not contain a valid extended address for this core address. The DLM is depicted in step 3 of Figure 2.

3.3 Message Receiver Module (MRM)

The MRM is responsible for classifying all incoming mail messages into two categories: *valid*, which is passed to the receiver's user agent, and *all others*, which are returned to the sender, who is asked to apply for a valid extended e-mail address. In the description of the MRM we will use the following definitions:

Definition 1. An extension e' is *genuine* for a user A, if $e' = e(A, B, n_B)$ for some party B and a current counter n_B; and *ingenuine* otherwise.

Definition 2. The subset of genuine extensions, which a user no longer wants to accept is denoted *invalid*, all other genuine extensions are *valid*.

For all parties B, the MRM must deliver each message, addressed with a valid $e(Alice, B, n_B)$ to Alice. The action of Alice removing an extension ϵ from the list of valid extensions is equivalent to classifying e-mail addressed with ϵ as junk e-mail. In this way Alice essentially *revokes* an extension she gave out to a party at some earlier point in time. The MRM must support this operation. Alice can further specify what the (different) actions of the MRM should be, when it receives a message addressed with either a invalid (but genuine) extension, a fake (ingenuine) extension, or no extension at all (i.e., Alice's core address). For example, a possible action might be an automatic reply, advising the sender to obtain a valid extension from Alice (via the handshake described in Section 3.4).

The MRM also allows Alice to *bind* an extension ϵ to a group G of other parties. The effect of binding is that only for parties in G the extension ϵ is valid, for all other parties, the extension ϵ is invalid.

When managing an intranet, it is beneficial to eliminate junk e-mail right at the gateway (MTA). Hence, part of this module should reside close to the MTA. Hence, an appropriate location for the MRM is inside the *gatekeeper*, which is a new component that handles all communication between the user agent and the MTA, as depicted in Figure 3. The gatekeeper maintains a database for each user to make a the decision which incoming mail the intended user classifies as junk. The required state information is also discussed in Section 3.7.

One way to control the operation of the MRM is to use an internal HTTP server, which allows the user to specify the desired actions (i.e. filter-like definitions) via a some web interface. The most important action is, of course, the revocation of a certain extended address.

In order to minimize the amount of data storage in the state information database, we introduce the notion of verifiable addresses:

Definition 3. A recipient-address a of a message intended for user A is *verifiable*, if there is an efficient check that a is genuine. This excludes the trivial check of simply going through the list of parties B, for whom A had generated an e-mail address at any point in the past.

The use of verifiable addresses thus allows that the system only has to store those extensions which are genuine, but no longer valid ("blacklist"), a potentially significant reduction in size. We note that a possible way to implement verifiability is to append the output of a keyed MAC to each extension. The input to the MAC is the extension, keyed with the user's secret. The key is shared with the user's MRM for easy discarding of fake (ingenuine) extensions.

3.4 Handshake Module (HM)

The handshake module is employed by the initiator (sender) and the receiver to implement the protocol for obtaining an extended address, as depicted in steps 5 and 8 of Figure 2.

If Alice would like to start exchanging e-mail with Bob, she might not know a valid extension for Bob. The handshake procedure allows Alice to obtain a valid extension for Bob, which she then can store in the address book of her user agent. We assume that Alice knows either Bob's core address or the URL of Bob's home-page (which is consistent with today's e-mail usage). Bob can publish these items on his business card, resume, and other immutable media. Alice initiates the handshake by sending e-mail to Bob's core address. The handshake method must involve a procedure acceptable to Alice, but not to spammers. We observe that spammers often do not give out returnable sender addresses and typically send to a large number (millions) of recipients. This leads to the following desirable properties for a handshake method:

- The handshake requires Alice to give a valid return address.
- The handshake requires Alice to "pay a cost". For example, Alice might be required to spend a certain amount of computing power in order to complete the handshake. We examine suitable cost functions in Section 6.

If Alice's message is satisfactory to Bob, she obtains a reply with a valid extension for Bob. We note that Bob's part in the handshake can be automated, so that Bob's HM handles all messages addressed to Bob's core address: The HM verifies that Alice has paid her cost and then generates a valid extension (which requires that the EGM is also accessible by the HM). In any other case, the HM either ignores a message sent to a core address or executes a user-specified action. HM might reside on the gatekeeper as well.

3.5 HTTP Proxy

Alice must employ a HTTP proxy to compute her extended address whenever she is registering at Web sites that require a valid e-mail address. The HTTP proxy employs the EGM, that was described in Section 3.1. This configuration is similar to the configuration of the Lucent Personalized Web Assistant (LPWA) [GGMM97], that is used to enhance privacy by creating consistent aliases for the user.

3.6 HTTP Server

Alice should also employ an HTTP server that has access to her state information database. This HTTP server is used for controlling the operation of the system and to allow manual requests of extended addresses. Alice could query the internal state information database as well as specify the desired actions of the system by a Web interface, which is maintained by the HTTP server.

3.7 State Information Database

If Alice revokes the extension $e(Alice, Bob, n_{Bob})$, used for communicating with both Bob and Cathy, she might later want to re-establish communication with Bob (alone). EGM uses the counter n_{Bob}, and increments it at the time when Alice re-establishes an extension. The new extension will be $e(Alice, B, n_{Bob+1})$. Initially, the generated extension is $e(Alice, Bob, 0)$. Hence, the EGM needs to maintain a counter for each party Alice is communication with. The HM needs access to these counters as well. This is depicted in Figure 3. The EGM and the HM on the gatekeeper access the counters in the state information database. The mail agent keeps an address book for storing valid addresses of other parties, like most agents do already today (Figure 3). So Alice only needs to tell the mail agent to which party she wants to send a message, and the system supplies the corresponding recipient address and her own extended sender address transparently.

The MRM must keep track of Alice's genuine and valid extensions. Possible solutions range from keeping a simple database of valid addresses ("whitelist") to use *verifiably* genuine addresses, and storing (the potentially much smaller list of) revoked addresses ("blacklist"). In all cases, user updates in the set of valid addresses have to be reflected in the state information database (Figure 3).

Accountability: When Alice qualifies an incoming message addressed to a currently valid address a as junk e-mail, then she might want to find out for whom a was originally computed. This party is directly or indirectly accountable for the junk e-mail message. This information is easily stored with the corresponding counter.

3.8 The Gatekeeper

The EGM, DLM and HM are best located between the user agent and the MTA, inside a new component called the *gatekeeper*, as depicted in Figure 3. The gatekeeper implements the functionality of our system without requiring any change to user agents or to the MTA. In this way, users need not change their user agents (mail readers), and the operation of the system is mostly transparent to the users.

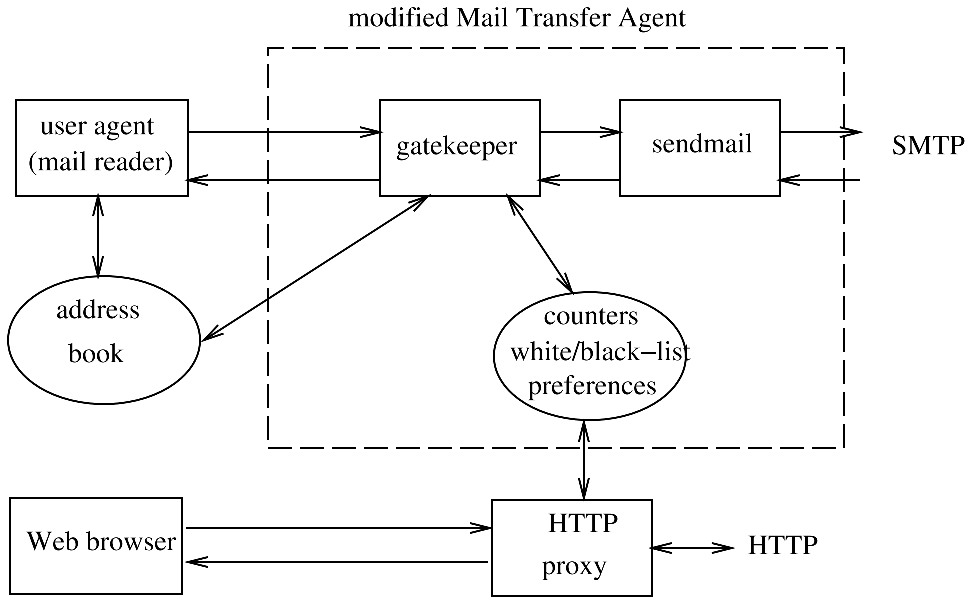

Fig. 3. Enhanced Communication Structure

Other systems, such as the channels system of [Hall98], use a similar component, the Personal Channel Agent (PCA), which also sits between the MTA and the user agents (mail clients).

Deployment

The proposed system may co-exist with other mail tools, and it does not require all parties to use it. The receiver may be protected by the system, while the senders use unmodified mailing tools. The senders have to keep the receiver's extended e-mail address in their address book, which is often maintained by their mailing tool.

The best place for placing the gatekeeper software is at the firewall, so that unsolicited mail can be returned to the sender immediately without the need to store or transfer it.

4 Alternative Protocols

The handshake protocol described in Section 3 involves 5 messages and a considerable internal state, as depicted in Figure 2. In this section we propose a variant that reduces the number of messages by computing the cost function prior to sending the first message to the recipient.

The variant protocol combines steps 3 and 5 in Figure 2 and eliminates step 4. Alice's first message to Bob contains the proof of computing the cost function. In this way, Bob does not have to ask Alice to compute the cost function. In this way, we reduce the number of messages from 5 to 3. If Bob ever revokes the extended address he gave to Alice, he will ask Alice to recompute the cost function.

5 Extension Function

In this section, we give precise requirements for the function e generating a user's extensions and show a possible realization.

5.1 Requirements

We consider the following adversarial model:

Definition 4. The adversary can obtain e-mail addresses from any chosen e-mail "parties" (e.g., Web-site database, Usenet newsgroup, private address book). All addresses stored at such a party are considered *compromised*.

Requirements on Generation of Sender Addresses:

1. *Authenticity:* An adversary cannot do better than guessing an extension of an uncompromised address with negligible probability. Having compromised some addresses of a user does not help an adversary in guessing the user's extension of an uncompromised address with non-negligible probability.
2. *Consistency:* The computed extension for a given party/state-counter is consistent.
3. *Efficiency:* The computation of the extension is efficient.

4. *Acceptability:* The range of generated extensions must be appropriate for e-mail addresses. Furthermore, the length of $e(A, B, n_B)$ is determined to give meaning to "negligible" in the above requirements.

5.2 Realization

The requirements *authenticity*, *consistency*, and *efficiency* would all be satisfied by the use of a perfect pseudo-random function generator. However, in the absence of such a function, we need to consider reasonably inexpensive approximations to it with respect to these requirements and to ensure *acceptability* at the same time.

We note that this was investigated by Bleichenbacher *et al* [BGGMM97] in the context of secure and pseudonymous client-server relationships. The so-called *Janus* function suggested in [BGGMM97] embodies the requirements we put on our module for generation of sender address extensions. We further note that the Lucent Personalized Web Assistant ([GGMM97] and [LPWA97]) computes a different e-mail address on a user's behalf for each Web-site, which requires registration. More generally, each system which derives different e-mail addresses from a core address under the above properties forms the basis for an effective tool against spam.

6 Handshake Function

The handshake can be implemented by simply having the initiator calling up her intended receiver on the phone, asking for an extension, or by sending an e-mail request of a special format. However, we will consider the somewhat more intricate solution of using a *cost function*, since this allows a considerable amount of automation. Intuitively, we want this function to be such that it has some well known and well regulated generation cost, is fast to verify, and can be easily implemented. Thus, a "friendly" user would not be too inconvenienced by its use (since it only has to be invoked once for each pair of communicating parties), but it would substantially hinder a spammer desiring to send to a large number of parties. In this section, we give precise requirements for the function controlling the cost of a handshake and show a possible realization.

Software implementing (1) a function for generating valid handshakes via the cost function and (2) a verifier of valid handshakes would be made publicly available.

We let c be a parameter describing the cost of performing the handshake; this may be a global parameter, but may also be individually set (and published) to customize the level of resistance to spam. Next, there is a target function T that decides whether its input is the result of a valid handshake or not. This function could simply compare a preset portion of its input to some predetermined pattern (of length corresponding to c). The initiator of a handshake repeatedly attempts to construct a potential handshake using a random approach, and verifies her attempt using the target function. She repeats her attempts until a valid handshake is found (the cost being measured in the expected number of trials.) She then sends her request for an extension to the core address of her desired recipient and encloses the handshake, which we note, is specific to this pair of parties. The handshake module at the recipient will sort out the request and evaluate the target function on it. If the result indicates that it is valid (and the initiator is not on some blacklist), then a valid extension will be generated and returned to the address of the initiator of the handshake. Given the correct extension, the initiator of the request is now able to send an e-mail that will reach the recipient.

6.1 Design Goals

The cost function is used to ensure that the party initiating the handshake has performed a certain expected amount of computation. Thus, this function controls the number of valid handshakes a party can initiate per time period. More formally, the cost function should have the following properties:

1. *Known Generation Cost:* Let c be a security parameter controlling the expected cost for generating a valid cost function. This cost measures the number of operations required and can be set either universally or individually. If the computation can be distributed, the cumulative expected cost per cost function evaluation must be at least c.
2. *Low Verification Cost:* We require that it should be inexpensive to verify that a given handshake is valid, where we leave "inexpensive" to mean considerably less expensive than c, but do not specify the exact maximum cost.
3. *Amortization Freeness:* Assume that E is an adversary who wants to generate k valid and different cost function evaluations: each such cost function transcript may be valid for any two protocol participants (who may not even exist at the time of the computation, but may be named and created afterwards) and for any time in the future. We require that the expected cost for E's computation is at least ϵkc, for ϵ very close to 1.
4. *Function Familiarity:* For practical purposes, it is an advantage if the cost function is based on a well known family of one-way primitives, for which computational costs are understood.
5. *Software Based:* In order to make the playground level, and not permit an adversary any significant advantage, we choose a cost function where hardware implementations are not giving any drastic advantage over software implementations.
6. *Transferability:* It may be required that a third party can verify that a given cost function evaluation is valid. In this situation, the third party wants the guarantee that the expected generation cost is c, even if all other handshake participants do not collaborate.

6.2 Realization

We will now briefly consider a possible implementation of the cost function, satisfying the most important requirements listed in the previous section. Recall that c is the cost parameter and T the target function. Herein, we will assume that A is the identity of the initiator of the handshake and B is the receiver. Furthermore, we will let d denote the day of the handshake; this is used to force an initiator to perform a new handshake computation if an old extension gets revoked – other degrees of granularity may be employed. The initiator will construct potentially valid handshakes using a probabilistic approach, and verify, using the target function, whether the result is satisfactory or not. A handshake specifies the identity of the sender, the identity of the recipient, and the date, and contains a randomly chosen string. The handshake is valid if a one-way function of it, evaluated by the target function, makes the latter output 1. The initiator of a request therefore tries different random strings until such a result is achieved. Consider now the following cost function:

Given a one-way hash function $h : \{0,1\}^* \rightarrow \{0,1\}^n$, a cost parameter $c < 2^n$ and a value $v \in \{0,1\}^*$, the cost function is evaluated by picking a random value $x \in \{0,1\}^*$, and evaluating $T(x,v,c)$. If $T(x,v,c) = 1$ then x is a satisfying choice for v. In other words, x is a valid cost function value. If $T(x,v,c) = 0$ we pick another random value x and repeat the above process. Here, we can choose MD5 for h, let $v = A||B||d$, let x be a random string of 128 bits, and let T output 1 when $h(x||h(v)) < 2^n/c$, otherwise 0. We note that the probability of picking a value x that causes T to accept is $1/c$, and so, the expected number of necessary hash function evaluations required to arrive at a satisfactory value x is $c+1$. Note that one cannot reduce the effort of computing the cost function by pre-computing $h(x)$ for many values of x, since MD5 works on 512 bit blocks, and its input $(x||h(v))$ is 256 bit long.

Evaluating the cost function for $c = 2^{20}$ would take about 2 seconds on a 266 MHz Pentium II processor[2]. On the other hand, verifying that a given value x satisfies the target function for a certain sender, receiver and date only takes two hash function evaluations.

Support: We are not able to prove that the suggested function has the desired properties. In particular, we do not know how to prove that the function has a known generation cost and is amortization free, since the area of lower bounds for cryptographic functions still to a large extent is a grey and unknown area. It will therefore have to suffice to explain why the cost function was chosen as it was.

First, the reason why we hash down v first, as opposed to plainly appending it to x, and hash these together, is that this makes the generation cost for the cost function (largely) independent on the length of v, which would not be the case if it were plainly to be appended.

Second, note that it is not possible to reduce the effort of computing the cost function by pre-computing $h(x)$ for many values of x, and then for many values $h(v)$ find a pair that results in a valid output. This is so since MD5 works on

[2] Wei Dai reports in `http://www.eskimo.com/~weidai/` that a 266 MHz Pentium II processor can compute MD5 hash at the speed of 32MB per second. This is equivalent to 0.5M blocks of 512 bits per second.

512 bit blocks, and its input, $x|h(v)$, is 256 bits long.

We believe that our requirement for known generation cost is satisfied: If we treat the hash function as a random oracle, then we can show that it is not possible to find a valid x in less c trials on average. Without the random oracle assumption, the generation cost depends on possible (today unknown) methods of evaluating hash functions such as MD5 on certain inputs. Similarly, the amortization-freeness can be shown in an idealized setting where we treat the hash function as a random oracle, and appears to hold without this assumption, given the current knowledge about hash functions. It is clear that the suggested function satisfies the rather fuzzy requirement of a low verification cost, since it only requires one function evaluation (of the extended function, including the three individual hash function applications). Also, the fuzzy requirement of function familiarity is satisfied given a choice of MD5, which is a function for hardware support does not gain a significant advantage over software implementations (as opposed to, for example, DES.) Finally, we see that the suggested function satisfies receipt availability, viz. that a third party is able to verify that a given valid cost function evaluation indeed is valid.

7 Discussion and Conclusion

We have proposed a design of a new system to effectively help in curbing junk e-mail. We have made typical sources for spammers to obtain valid e-mail addresses (Web site databases, Usenet newsgroups, ISP lists of (core) addresses of subscribers) much less attractive. Upon receiving a first junk e-mail message, a user will declare the corresponding extension invalid. Hence, a list of extended addresses is useless, if another spammer had access to it before. A core address cannot be used directly for a spamming. We have furthermore introduced a non-negligible cost to obtain a new and valid e-mail addresses, so even if junk e-mailers can use a high speed machine with good price/performance ratio, they must reduce the rate of sending junk messages, which will require them to target their messages carefully and avoid mass mailing. Furthermore, a spammer must provide a working e-mail address in the course of this handshake, which serves as another deterrent and prevents spoofing, currently prevalent among spammers.

We have shown that our system is mostly transparent to an e-mail user and easily integrated with today's e-mail tools. Implementations and practical use will guide in showing the appropriate cost for a handshake, the appropriate degree of automation of the functionalities and other possible trade-offs.

Acknowledgments

The authors would like to thank the anonymous referees for their helpful comments, and Daniel Bleichenbacher for very helpful discussions.

References

[BGGMM97] D. Bleichenbacher, E. Gabber, P. Gibbons, Y. Matias, A. Mayer, *On Secure and Pseudonymous Client-Relationships with Multiple Servers*, submitted, also available at URL `http://www.bell-labs.com/projects/lpwa/papers.html`.

[BT91] N.S. Borenstein and C.A. Thyberg, *Power, Ease of Use and Cooperative Work in a Practical Multimedia Message System*, International Journal of Man-Machine Studies, Volume 34, Number 2, February 1991, pp. 229–259.

[CL98] L.F. Cranor and B.A. LaMacchia, *Spam!*, to appear in Communications of the ACM. Also available at URL `http://www.research.att.com/~lorrie/pubs/spam/`.

[DN92] C. Dwork and M. Naor, *Pricing via Processing or Combating Junk Mail*, Crypto'92, pp. 139–147.

[FM97] M. Franklin and D. Malkhi, *Auditable Metering with Lightweight Security*, Proc. of Financial Cryptography'97, Springer-Verlag, LNCS 1318, pp. 151–160.

[GGMM97] E. Gabber, P. Gibbons, Y. Matias, A. Mayer, *How to Make Personalized Web Browsing Simple, Secure, and Anonymous*, Proc. of Financial Cryptography'97, Springer-Verlag, LNCS 1318, pp. 17–31. Also available at URL `http://www.bell-labs.com/projects/lpwa/papers.html`.

[Hall98] R.J. Hall, *Channels: Avoiding Unwanted Electronic Mail*, to appear in Communications of the ACM. Also available at URL `ftp://ftp.research.att.com/dist/hall/papers/agents/channels-long.ps`.

[LPWA97] LPWA: The Lucent Personalized Web Assistant, A Bell Labs Technology Demonstration. Available at URL `http://lpwa.com`.

This article was processed using the LaTeX macro package with LLNCS style