RE: Reliable Email

Scott Garriss[†], Michael Kaminsky[*]
Michael J. Freedman[‡◦], Brad Karp[**], David Mazières[◦], Haifeng Yu[*†]
[†]*Carnegie Mellon University,* [*]*Intel Research Pittsburgh,*
[‡]*New York University,* [**]*University College London,* [◦]*Stanford University*

Abstract

The explosive growth in unwanted email has prompted the development of techniques for the *rejection* of email, intended to shield recipients from the onerous task of identifying the legitimate email in their inboxes amid a sea of spam. Unfortunately, widely used content-based filtering systems have converted the spam problem into a false positive one: email has become *unreliable*. Email *acceptance* techniques complement rejection ones; they can help prevent false positives by filing email into a user's inbox before it is considered for rejection. *Whitelisting*, whereby recipients accept email from some set of authorized senders, is one such acceptance technique.

We present Reliable Email (RE:), a new whitelisting system that incurs zero false positives among socially connected users. Unlike previous whitelisting systems, which require that whitelists be populated manually, RE: exploits *friend-of-friend* relationships among email correspondents to populate whitelists *automatically.* To do so, RE: permits an email's recipient to discover whether other email users have whitelisted the email's sender, while preserving the privacy of users' email contacts with cryptographic private matching techniques. Using real email traces from two sites, we demonstrate that RE: renders a significant fraction of received email reliable. Our evaluation also shows that RE: can prevent up to 88% of the false positives incurred by a widely deployed email rejection system, at modest computational cost.

1 Introduction and Motivation

Written communication is most useful when it is *reliable*; that is, when senders and recipients can both expect that a message sent will be received successfully by the intended recipient. Correspondents who are connected socially, whether directly or indirectly, often have something to lose if a message is not received (e.g., an urgent request from one colleague to another, or an urgent communication to a parent from her child's teacher). Similarly, a sender is most likely to expect a recipient to read and act on communication when the two parties have a preexisting social relationship, direct or indirect.

By the above definition, Internet email has been reliable throughout most of its entire long history, beginning with its 1971 origins on the ARPAnet. Email users have come to rely upon email reaching its destination.[1] Today, however, unsolicited commercial email has rendered Internet email unreliable, as we explain below. *We seek to restore email's reliability among correspondents who are linked to one another socially.*

Spam inconveniences users by forcing them to search for legitimate email in an inbox dominated by chaff. Despite the efforts of legislative and law-enforcement bodies to the contrary, spam remains a pressing problem of large scale. In 2003, corporations spent an estimated $2.5 billion on increased SMTP server capacity needed to process spam [30], and in July 2005, over 65% of email that crossed the Internet was spam [1]. The natural response of researchers and practitioners has been to develop and deploy a broad range of techniques intended to ensure that spam does not reach a user's inbox [2, 4, 6, 8, 19, 23, 29, 32, 33].

Content-based filtering has been particularly widely adopted as a spam defense strategy, perhaps because of its wide availability in free and commercial implementations. These systems are designed to reject email based on the presence of string tokens associated with spam. These tokens may either be selected manually by an administrator, or may be learned, most often by applying Bayesian learning to user-supplied training examples.

Many have reported great success at rejecting spam using content-based techniques, as measured by sub-1% false negative rates [20, 34]. Unfortunately, content-based filtering has replaced the spam problem with a *false positive* one. That is, misclassification of legitimate email as spam by content-based filters has rendered email unreliable. False positives are arguably more severe than spam in one's inbox, in that they are not merely a waste of a user's time—they represent possibly important email the user does not see.

The reasons why content-based filters result in false

[1] SMTP server failures, disk exhaustion, and other hardware or software failures—or a recipient's simply not reading his email—can all of course result in delay or non-delivery. Such cases have not historically led users to view email itself as significantly unreliable, and some are not amenable to eradication by technical means. They are thus beyond the scope of consideration in this work.

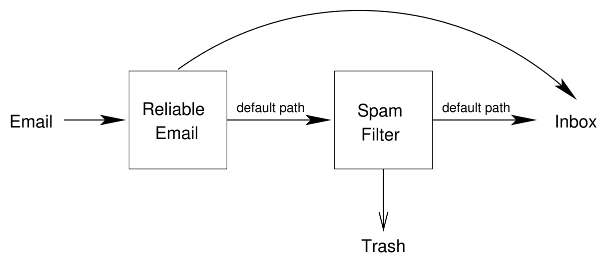

Figure 1: Complementary whitelisting and blacklisting.

positives are manifold.[2] A legitimate communication may contain strings associated with spam (e.g., "mortgage," "offer," "lottery," "Lagos," or any number of more colorful words that figure prominently in the descriptions of items flogged by spammers). Even if one were to build a content-based filter that could avoid false positives in these cases, others remain problematic, such as forwarding an interesting spam to a mailing list used for discussion by spam filter designers. In effect, content-based filters act as "dumb censors," as they prevent legitimate discussion on the basis of keyword matching—users can no longer say what they want in email.

In this paper, we propose Reliable Email (RE:), an *automated email acceptance system* based on whitelisting of email according to its sender. Figure 1 depicts a schematic view of how RE: fits into an email recipient's spam-fighting system. RE: is first to examine inbound email, and it delivers any message it accepts directly to the recipient's inbox. Note that RE: is entirely complementary to a mail rejection system; it cannot increase false positives because it either accepts a message or passes it to whatever rejection system was already in place.

The concept of a mail acceptance system is hardly new. Perhaps the simplest mail acceptance system is a *sender whitelist*, which places mail from senders enumerated on a list directly into the recipient's inbox. Today, whitelisting is rarely used in practice for three chief reasons:

- Whitelisting based on sender is trivially defeated by forging `From:` addresses, which are unauthenticated in SMTP. Even without knowing the contents of a recipient's whitelist, a spammer may trivially generate spam forged to appear to be *from* the recipient, whose address is most likely whitelisted.
- A recipient's whitelist cannot accept mail from a sender previously unknown to the recipient.
- Populating whitelists requires manual effort distributed diffusely in time, as users acquire new contacts.

RE: incorporates a mechanism to defeat forgery of `From:` addresses, as do other proposals that aim to stop spam [8, 33]. More significantly, RE: *automatically* broadens the set of senders whose mail is accepted by recipients' whitelists by explicitly examining the social network among email users. In particular, RE: allows user *A* to *attest* to user *B*. Such an *attestation* indicates that user *A* is willing to have email from user *B* directly filed in his inbox. An attestation thus roughly corresponds to the notion, "User *A* trusts user *B* not to send him spam." We say *B* is a *friend* of *A*. Clearly, attestations are useful for accepting mail in cases where a sender and recipient are friends; a sender may choose to generate an attestation for a recipient, and vice-versa, on the basis of the other party's identity.

We observe further that attestations are useful for accepting mail in cases where the sender and recipient are *not* already friends, but instead share a friend in common, a situation we term *friend-of-friend* (FoF). Suppose *A* and *B* are friends, *B* and *C* are friends, but *A* and *C* are as yet unknown to each other. If *C* sends email to *A*, the FoF relationship between *A* and *C* may give *A* confidence that *C* is not a spammer. That is, *A* may trust *B* not to be a spammer, *B* may trust *C* not to be a spammer, and on this basis, *A* may conclude that *C* is unlikely to be one.

Each email domain that participates in RE: runs a server that stores attestations. Together, the distributed collection of RE: servers allows *FoF queries* over attestations, whereby an email's recipient may determine whether the email's sender is an FoF. RE: thus allows a recipient to accept email from FoFs without requiring all users to trust a central authority.

Because attestations name email correspondents, allowing one user to query another user for attestations raises privacy concerns. RE: employs cryptographic private matching techniques to preserve the privacy of users' contact lists. Section 3.8 details the specific guarantees that RE: provides.

A central question is how useful FoF relationships are in increasing the number of emails RE: accepts into a user's inbox, versus the number of emails accepted by direct friend relationships alone. We consider this question in detail in Section 5. By way of motivation, we note briefly here that when evaluating the utility of social whitelisting using email traces from multiple sites, we find that RE: can accept almost 75% of received email and can prevent up to 88% of the false positives incurred by the existing spam filter. Moreover, augmenting friend relationships with FoF relationships increases the fraction of all received email accepted by RE: by at least 10%—a significant improvement in the fraction of received email rendered reliable.

We proceed in the remainder of this paper as follows. After reviewing related work in Section 2, Section 3 of-

[2] Some argue that spammers will eventually be able to evade Bayesian-trained content filters [10]; we are concerned chiefly with their false positives in this work.

fers design goals for a distributed whitelisting system and describes the design of RE: in detail. Section 4 describes our working RE: prototype, and Section 5 evaluates the system. Section 6 discusses various design decisions and open questions. We conclude in Section 7.

2 Related Work

We now survey the numerous and varied schemes proposed to fight spam, and compare these approaches to that taken in RE:.

Forgery Protection. Today's whitelists are often vulnerable to abuse because sender addresses are unauthenticated in SMTP, and thus may be forged trivially. Current methods for the prevention of mail forgery fall into two categories: digitally signed mail and trusted senders.

Digitally signing mail (e.g., with PGP [35]) allows recipients to authenticate the mail's content, including the sender's address. Clearly, requiring that all mail be digitally signed would solve the address forgery problem, but the use of digital signatures is hampered by the lack of any widely deployed public-key infrastructure.

Under trusted senders, a recipient can determine whether a received mail originated from a mail server in the domain of the sender's address. The Sender Policy Framework (SPF [33]), its derivative, Sender ID [26], and Yahoo! Domain Keys [8] exemplify this approach.

Social Networks. Ebel et al. [16] study the topology of an email social network and show that it exhibits small-world behavior. Kong et al. [23] use this finding to propose a collaborative, content-based email rejection system based on social networks, in which a user manually identifies the spam he receives, and publishes a digest of it to his social network. A user queries these digests to determine if mail he has received was previously classified as spam by others in his social network. This scheme presumes that users who are connected socially have the same definition of what constitutes spam *content*.

Ceglowski and Schachter [12] and Brickley and Miller [11] propose mechanisms for exchanging whitelist information using Bloom filters and SHA-1 hashes, respectively. These data structures do not contain cleartext whitelist entries, but are open to straightforward dictionary attacks. Both schemes assume sender addresses are not forged. Goldbeck and Hendler [19] present an algorithm that infers a trust score for a given sender based on social relationships, but compromises user privacy by requiring that users publish their social relationships.

PGP [35], though not originally intended to fight spam, uses a web-of-trust model for key distribution. The web-of-trust model relies on friend-of-friend trust relationships, as does RE:. RE:, however, uses these FoF relationships only to help identify legitimate senders, not for key distribution. RE: intentionally sidesteps the problem of robust key distribution, as described in Section 3.4.

Mail Rejection Systems. Machine learning techniques for text classification have been adapted to distinguish spam from legitimate email [21, 31]. Systems in this category, as exemplified by SpamAssassin [6], are by far the most widely deployed spam defense. As previously described, such systems reduce the reliability of email, as they inevitably classify some legitimate email as spam. Such false positives are a severe enough problem that some of the most accurate spam classification systems (e.g., [2]) resort to human labor to help with their classification, at greatly increased cost.

Since human-directed classification is accurate, but costly, several efforts have been made to distribute that cost across an email system's users. SpamNet [3], Distributed Checksum Clearinghouse (DCC) [4], and Razor2 [29] are all systems in which email users collaborate to classify spam. In each of these systems, users report spam to a centralized database. It is unclear how resilient these systems are to malicious users, particularly ones who mount a Sybil attack [13].

One may also reject email on the basis of the IP address of the sending host; past spam sources are thus *blacklisted.* Realtime Blackhole Lists (RBLs [5], or more generally, DNSBLs) and SpamCop [7] take this approach. These systems automate the distribution of a list of IP addresses known to have sent spam or run open SMTP relays. Blacklisting the SMTP server for a domain runs the risk of blacklisting many legitimate users. Because these lists are maintained in ad hoc fashion, it can be slow to have a server's reputation "cleared."

Other Mail Acceptance Systems. In proof-of-work schemes [9, 14, 15], a sender "pays" to send an email by solving a computational puzzle. Payment schemes rely on a similar notion, but they require senders to expend money rather than computation, e.g., by directly paying recipients [25]. For both classes of system, the intent is that the resources required to send mail will be prohibitively expensive in volume for a spammer, while affordable in smaller quantities for an average legitimate email sender. Laurie and Clayton [24] argue that spammers may harness the computational resources of large collections of compromised hosts to send spam in volume. Keeping email too expensive for spammers might then entail making it too expensive for heavy legitimate email users.

DQE [32] aims to limit the volume of email any one sender may send, without compromising the reliability of email between legitimate users. To achieve this goal, DQE employs a central authority trusted by all email users, which allocates quotas of unforgeable stamps to email senders. Senders attach a stamp to every email they send. Recipients check the validity of the stamp on mail they receive. DQE eliminates false positives between sender-recipient pairs that adopt the system (and thus trust the same central quota allocator). When DQE is deployed incrementally, however, a recipient that wants protection from spam must fall back on a mail rejection system when it receives unstamped mail; in so doing, the recipient risks a false positive. We believe RE: and DQE complement one another well. RE: incurs no false positives for email between friends and FoFs, but cannot prevent false positives on mail whose sender is not a friend or FoF of the recipient. DQE, in contrast, can prevent false positives between *any* sender-recipient pair, provided they trust the same central quota allocator.

3 Design

We now continue by offering design goals for a robust and secure distributed email social whitelisting system. We then define terms useful in reasoning about RE:, describe RE:'s major components, and explain the system by way of a typical email usage scenario. We conclude this section by presenting the assumptions under which RE: operates and its corresponding security guarantees.

3.1 Design Goals

To be robust and secure, a distributed system that whitelists email and allows users to generate and query for attestations must provide three basic guarantees.

Sender addresses cannot be forged. Because a whitelisting system automatically accepts email based on sender address, protecting against this type of forgery is particularly important. Basic SMTP, however, does not provide any mechanism to prevent a user from sending email with an arbitrary `From:` address. RE: robustly verifies that the `From:` address in a received email has not been forged.

Attestations cannot be forged. An attestation is a statement by one user that another user is trusted to send email. Forging attestations would allow an adversary (spammer) to trick the recipient into accepting the adversary's email, even if the sender's address is not forged. RE: uses digital signatures to guarantee that attestations cannot be forged.

Privacy when exchanging whitelist information. A social whitelisting system can automatically accept email based on FoF relationships. A naive approach to learning about mutual friends is for one party to send his list of friends to the other party, who computes the set intersection. This approach, however, compromises the privacy of the first party. RE: uses a private set-matching protocol to compute the intersection of sets of friends, but still provides provable privacy guarantees for both parties.

Additionally, any spam-fighting system is far more easily deployable if it provides two practical properties:

Incremental deployability. Inevitably, some users will adopt a spam-fighting system before others. A spam-fighting system ideally will offer real benefit to the community of users who have adopted it thus far. Because RE: is a mail acceptance system, it is complementary to pre-existing spam rejection systems, and thus easy to deploy incrementally. Clearly, the deployment of RE: does nothing to worsen the spam problem for those who have not adopted it. Moreover, RE: can confer benefit in the form of reduced false positives in any pairwise deployment, in which the domains of both sender and recipient run it.

Compatibility with today's SMTP. Making changes to the SMTP protocol slows adoption because developers must incorporate the new anti-spam SMTP protocol extensions into their mail software, and site administrators must then adopt one of the resulting new releases of this software. RE: does not change SMTP in any way; the system communicates over its own connections, and it is easily placed in the email processing path of most mail servers.

3.2 Definitions

The two key participants in any email exchange are the sender (S) and the recipient (R). Every user U in RE: possesses a public/secret key pair, denoted by PK_U and SK_U. The user keeps his secret key somewhere where he can access it while sending email. He registers his public key with his *Attestation Server (AS)*, described below.

As described in Section 1, a user can issue an *attestation* to vouch that another user is a legitimate sender. An attestation by A for B, written $A \rightarrow B$, indicates that A trusts B to send email, and that A believes that other people should trust B to send email. Attestations contain the identities of the attester and the attestee, as well as an expiration time. More formally, an attestation is

$$A \rightarrow B \;=\; \{\mathrm{Hash}(A), \mathrm{Hash}(B), start, duration\}_{SK_A}$$

Hash() is a collision-resistant cryptographic hash (e.g., SHA-1) of the attester's or attestee's email address. The *start* and *duration* fields specify when the attestation expires. The entire attestation is cryptographically signed by the attester using his secret key.

Each SMTP domain that participates in RE: runs an Attestation Server (AS) responsible for storing attestations both by and to the users of that domain. The AS

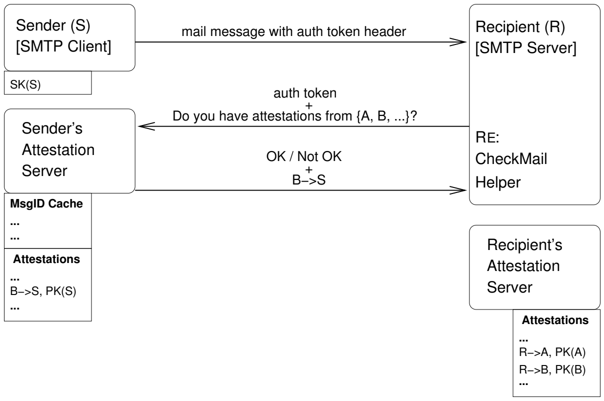

Figure 2: Sending Mail and Finding FoFs with RE:

also stores the public keys of its users as well as the public keys of users attested to by its users (i.e., for each attestation $A \rightarrow B$, where A is in the AS's domain, the AS stores PK_B). When the AS's users receive email from third-party senders, they use these attestee public keys to verify the signatures on the attestations that the third-party presents during an FoF query, as described below. The AS can run on the same machine as the domain's SMTP server, or on a different host, as indicated by DNS SRV records [22].

An AS responds to several types of client requests, such as queries for public keys, attestations, and FoFs. Some queries are restricted to users in the AS's domain (e.g., storing a public key) while others are open to everyone (e.g., getting a public key). Section 4 describes the full list of requests an AS supports.

3.3 Example: Sending Mail with RE:

Figure 2 shows an example of how two users running RE:, S and R, correspond by email. First, S composes a message to R and creates an authentication token (see Section 3.4). S signs the authentication token with his secret key and sends the message to R using standard SMTP. The authentication token is included as a header in the email message to avoid requiring modifications to the SMTP protocol.

R can decide to accept the mail based on the sender in one of two ways, depending on whether the sender is a direct friend or an FoF. These two cases differ in what R sends to S and what S returns. Figure 2 shows the FoF case.

To check for direct friendship, R examines the list of senders to whom he has attested (stored on R's AS); these attestations are of the form $R \rightarrow *$. If the recipient has attested to the sender (i.e., $R \rightarrow S$ exists), R contacts S's AS to verify the authentication token. In this case, the token verification is the only communication between R and S. If the verification succeeds, RE: accepts the message and delivers it directly to the recipient's inbox.

If S is not one of R's direct friends, R will try to determine if S is an FoF. Specifically, R seeks to discover whether any of his direct friends has attested to S: is there an x such that $R \rightarrow x$ and $x \rightarrow S$? To answer this question, R queries the sender's AS. This query effectively contains a list of the recipient's direct friends. The sender's AS responds with a list of any attestations to the sender it holds from those direct friends. In Figure 2, the sender's AS returns $B \rightarrow S$, where B is a mutual friend. In this case, the communication between R and S combines the FoF query and verification of the authentication token.

In practice, R never actually sends its list of friends directly to S. Rather, R performs this FoF query using a private set matching protocol described in Section 3.8, which guarantees that the sender does not learn any of the recipient's direct friends and that the recipient only learns the attestations that are from his friends (i.e., the set intersection).

If R finds a match for this FoF query—e.g., an attestation to the sender by a B—it verifies the signature and freshness of this attestation. Note that R's own AS already has the mutual friend's public key, PK_B, stored alongside $R \rightarrow B$. If the attestation is valid and the authentication token verification succeeds, RE: accepts the message and delivers it to the recipient's inbox. Otherwise, R concludes that it is unable to determine whether or not the message is spam. R can then perform a default action (e.g., running the mail through a spam filter, graylisting it, etc.).

3.4 Sender Authentication

RE: attaches an *authentication token* to each outgoing email to protect against forged sender email addresses. This token is defined as follows:

$$\{Sender, Recipient, Timestamp, MessageID\}_{SK_S}$$

The recipient verifies the authentication token before determining whether an incoming email should be whitelisted: (1) The recipient checks that the sender, recipient, and unique message ID match the values found in the message itself. This check ensures that the token cannot successfully be attached to an email destined to a different address. The recipient then connects to the sender's AS, which (2) checks that the authentication token is unused and (3) verifies the token's signature, as the AS knows the sender's public key PK_S. If an authentication token has been previously redeemed or the signature fails verification, the token check fails.[3]

[3] If after the recipient redeems the authentication token, one party suffers a crash or a network failure, the recipient will be unable to com-

Thus, authentication tokens serve two purposes. First, they allow recipients to reject forged sender addresses. Second, they prevent an adversary from making arbitrary FoF queries to the AS by replaying a previously redeemed token. Restricting FoF queries prevents the adversary from learning who has attested to the sender.

To verify that the authentication token is unused, the AS caches previously used tokens. To bound the size of this cache, the authentication token contains a timestamp. The AS only keeps tokens whose corresponding timestamps (times of issuance) are more recent than t seconds in the past. A reasonable value for t might be one week. Expiring authentication tokens based on timestamps assumes loosely synchronized clocks.

Because the recipient contacts the sender's AS to verify the authentication token, the recipient need not know the sender's public key. This method of verification assumes, however, that an adversary cannot perform a man-in-the-middle attack (see Section 3.7). Performing a man-in-the-middle attack in the wide-area network is non-trivial, and is far more work than a typical spammer would have the resources to perform for each email. Furthermore, successfully tricking the recipient into believing an invalid authentication token is valid results in, at worst, accepting a spam email into a user's inbox (i.e., a false negative).

3.5 Revocation

If a user's secret key is compromised or lost, authentication tokens and attestations signed by that key should be revoked. Once a user discovers that his key has been compromised, he uploads a new public key to his Attestation Server. From that point forward, the AS simply stops accepting authentication tokens signed with the old key.

For attestations, there are two cases that RE: must handle. First, when a recipient R's key is compromised, he should invalidate all attestations $R \rightarrow *$ stored at its own AS; we call these *local* attestations. When R uploads a new key to its AS, the AS can simply remove these local attestations; R can then re-issue the attestations as needed.

Second, R's friends must stop using R's attestations stored at their Attestation Servers; we call these *remote* attestations. Currently, RE: does not have a way to notify every one of R's friends automatically that his attestations are now invalid. RE: handles this case through expiration dates in attestations. Two expiration dates are relevant: (1) The remote attestation itself (signed by R) has an expiration date and after this date, R's friends will stop using it. (2) During an FoF query, R's friends present these remote attestations to users who have attested to R, and these users attempt to verify the signature on the remote attestations using R's public key, which they stored when attesting to R. These users will reject the remote attestation if it is signed with an old, now invalid key. For example, if user U holds $U \rightarrow R$ and PK_R, U will re-fetch the attestation and PK_R once the attestation expires. Until then, however, U might accept mail from one of R's friends, or an adversary that can generate any $R \rightarrow *$ using R's compromised key.

A similar situation exists for revoking a single attestation, e.g., after a recipient R attests to a spammer by accident. To prevent the attestation's direct use (not FoF use), R simply removes the offending local attestation from his AS, thereby eliminating future false negatives from this spammer. This removal, however, does not help if the spammer (the attestee) is already holding a copy of the attestation, as he can subsequently use this remote attestation to send mail to users who have attested to R. Currently, RE: only limits the duration of a bad attestation by its expiration time, but alternatively the user can decide to remove his attestation for the recipient and thereby disregard the remote attestation presented by the spammer, as described in the following section.

3.6 Policy Decisions

RE: leaves several policy decisions to the user and/or system administrator. Most importantly, a user must decide when to create attestations. The most labor-intensive but least error-prone policy is for the user to manually attest to other users after verifying that the sender is trusted to send email. This verification can be out-of-band (the user knows the sender personally) or can be based on the recommendation of a mutual friend learned through the RE: protocols. In the example above, the recipient R might decide to attest to the sender S because his mutual friend B attested to S. In this way, FoF queries help bootstrap a user's attestations to include a previously unknown correspondent.

Users can also specify policies that automatically create attestations. For example, a user might decide that anyone to whom he sends email is a trusted sender—presumably this user does not send email to spammers—and thus automatically create an attestation for each recipient of his outbound email. The user might also tune such a policy decision to include or exclude mail sent to particular domains. Another example of automatic attestation creation might be to attest to anyone that sends a user three non-spam emails that the user does not discard.

plete the SMTP transaction. When the sender retries, RE: will attempt to redeem the same authentication token a second time. This authentication check will fail, however, and RE: will not be able to accept the message automatically. In the current design, the recipient will then simply fall back to its default action. In future versions of the protocol, however, we intend to make RE: robust under such failures.

Users must also decide how to set expiration dates in an attestation. Expiration dates allow users to limit how long their friends will continue using the remote attestation. For example, a user might choose a more distant expiration date for personal acquaintances versus senders attested to automatically because they have sent three legitimate emails.

We presume that a user's trusted friend will only rarely attest to a spammer. If this friend does attest to a spammer, the user will accept mail from the spammer because of the FoF social relationship. The user will know, however, which friend attested to the spammer. In this sense, RE: limits the harm of attesting to a spammer: users can identify a friend's ill-considered remote attestation, and they can choose to ignore that friend's remote attestations when accepting inbound email in the future.

We elected to limit the social networking component of our system to one level (FoF). One might also consider using social relationships of three hops or longer in whitelisting email. We defer a discussion of this choice to Section 6.

3.7 Assumptions

In addition to standard cryptographic hardness assumptions, RE: operates under the following assumptions:

- Clocks are loosely synchronized to within some error bound. This is to ensure that (1) the sender's AS accepts only authentication tokens that are not too old, and that (2) the recipient uses only attestations that have not expired.
- An adversary cannot launch a man-in-the-middle attack. This assumption implies, for example, that the adversary cannot subvert forward DNS queries or intercept and modify IP packets traveling between an email sender and recipient.
- An adversary cannot compromise the sender's AS and/or convince it to lie about the validity of an authentication token or public key.
- An adversary cannot compromise a sender's machine. Section 6 discusses this assumption in more detail.

3.8 Privacy Protection

The FoF query allows the recipient to determine if any of his friends have attested to the sender. RE: performs this set intersection using a Private Matching (PM) protocol [18] that provides the following attractive privacy properties:

- The sender S does not learn anything about the recipient's friends. Both sender and recipient do learn an upper bound on the number of real friends presented by the opposite party, however.
- The recipient R learns only the intersection of the two sets of friends, i.e., those persons f for whom $R \rightarrow f$ and $f \rightarrow S$. The PM protocol does not prevent parties from "lying" about their inputs; thus, the recipient can include arbitrary friends in his list when computing the set intersection. However, such inputs will fail later attestation verification and thus not result in a successful FoF chain.
- A third party observing all messages between sender and recipient learns only an upper bound on the size of each input, but nothing about their content nor the size of the intersection.
- No other party other than the recipient can execute the FoF query, as the AS only allows one query per valid authentication token.

RE:'s private matching protocol is a type of secure two-party computation that is optimized for computation and communication efficiency. At a high level, the basic protocol has three steps:

1. The recipient encodes his k_R friends' names in a special encrypted data structure, which he sends to the sender.
2. The sender performs a computation on the encrypted data structure with each of her k_S friends' names and the corresponding attestations, generating k_S outputs. If the sender's friend f is encoded within the encrypted data structure, an output's underlying plaintext becomes the attestation $f \rightarrow S$; otherwise, that output's plaintext becomes random. She sends the k_S outputs back to the recipient.
3. The recipient decrypts these k_S results. He recovers the attestations corresponding to their set of friends in common.

Our private matching protocol takes advantage of the special mathematical properties of certain public-key encryption schemes, such as Paillier [28] and a variant of ElGamal [17][4], that preserve the group homomorphism of addition and allow multiplication by a constant. In other words, the following operations can be performed without knowledge of the private key: (1) Given two encryptions $enc(m_1)$ and $enc(m_2)$, one can compute $enc(m_1+m_2) = enc(m_1) \cdot enc(m_2)$. (2) Given some constant c belonging to the same group, $enc(cm) = enc(m)^c$.

[4] Standard ElGamal, which encrypts a message m as $\langle g^x, g^{xr}m \rangle$, does not directly support the necessary homomorphic operations required. Thus, when requiring these homomorphic properties, we encrypt m as $\langle g^x, g^{xr}g^m \rangle$, even though decrypting this ciphertext only recovers g^m, not m! This variant is sufficient for almost all operations, with the exception of the sender encrypting its attestation for later recovery by the recipient, which therefore uses standard ElGamal.

We use the following corollary of these properties: (3) Given encryptions of the coefficients $a_0, \ldots, a_k$ of a polynomial P of degree k, and a plaintext y, one can compute an encryption of $P(y)$.

We now construct a basic secure PM protocol in the following manner:

1. The recipient R defines a polynomial P whose roots are hash values encoding his k_R friends, i.e., given $x_i \leftarrow \text{Hash}(f_i^R)$, R computes:
$$P(y) = (x_1 - y)(x_2 - y)\ldots(x_{k_R} - y) = \sum_{u=0}^{k_R} a_u y^u$$
R encrypts these k_R coefficients under his public key and sends the resulting ciphertexts to S.
2. The sender S uses the homomorphic properties of the encryption system to evaluate the polynomial on each hash value of her k_S friends, i.e., $\forall i, y_i \leftarrow \text{Hash}(f_i^S)$:
$$enc\Big(P(y_i)\Big) = enc(a_0)\Big(enc(a_1)\big(\ldots enc(a_{k_R})^{y_i}\big)^{y_i}\Big)^{y_i}$$
S then multiplies each $P(y_i)$ result by a fresh random number r to get an intermediate result, and she adds it to her corresponding attestation from f_i^S (encrypted under R's public key):
$$enc\left(r \cdot P(y_i) + \{f_i^S \to S\}\right)$$
S randomly permutes this set and returns it to R.
3. R decrypts each element of this set. For every friend in common, $P(y_i) = 0$ and R recovers the attestation $f_i^S \to S$. Otherwise, $P(y_i)$ is non-zero and the resulting decryption appears random. R checks that f_i^S is in its friends list and verifies the attestation $f_i^S \to S$ before accepting the FoF chain.

For proofs of the protocol's security, as well as efficiency optimizations and other implementation details, we refer the reader to Freedman et al. [18].

Such cryptographic tools must be applied with care to prevent information leakage that unknowingly introduces side-channel attacks against the protocol's privacy. For example, one may be tempted to *distribute* public keys inside FoF attestations, much like a web-of-trust for key distribution. However, this approach can be used to break the system's privacy. Consider the case in which a sender maintains multiple public keys. If the sender receives attestations from friends on different keys, and a recipient accepts one such key following a successful FoF protocol with the sender and then subsequently attests to it, the sender can immediately deduce which friend the two parties have in common. Thus, all identifying information (public keys, email addresses) must be retrieved directly from their corresponding parties, in order to ensure consistency across participants and thus prevent such side-channel attacks.

Public RPCs	Private RPCs
GetPK	GetWhitelist
SubmitAtt	GetAtt
CheckAuth	SetPK
FindFriend	

Table 1: RPC interface to the Attestation Server

4 Implementation

The current RE: prototype consists of the Attestation Server, Private Matching (PM) implementation, and a number of utilities for creating attestations, authentication tokens, and for checking incoming mail. The prototype is built atop the SFS toolkit [27], which provides primitives for asynchronous event-driven programming, cryptography, and RPC. We have integrated the RE: prototype with the Mutt mail client, the Mail Avenger SMTP server, and the Postfix SMTP client. Note that we made no modifications to any of these pre-existing mail tools. RE: is roughly 4500 lines of C++ source, plus 275 lines of Sun XDR code for the RE: wire protocol specification. RE: uses DSA for digital signatures on attestations and authentication tokens. For the PM protocol, RE: uses the ElGamal variant [17].

4.1 Attestation Server

Currently, each email domain that wishes to participate in RE: must run an Attestation Server. The domain publishes SRV records [22] to indicate the location of the AS. For each user A in the domain, the Attestation Server is responsible for maintaining the following:

- A's public key PK_A
- all unexpired, redeemed authentication tokens generated by A when sending email
- all attestations $A \to x$ created by A, along with x's public key PK_x
- all remote attestations $x \to A$ for A

In addition to maintaining this information, the Attestation Server provides two RPC interfaces (see Table 1). The public interface is used by email recipients to decide whether to accept mail sent from this domain. The private interface, requiring authentication, is used only by the domain's recipients: when processing incoming mail, for updating public keys, and for providing users access to their attestations and the corresponding attestee public keys. The current implementation restricts access to the private RPCs to clients running on the same machine as the AS; future versions of the Attestation Server could allow clients to authenticate through existing security mechanisms (e.g., SSL, Kerberos) and/or provide a secure Web interface to the private RPCs (e.g., SetPK). We now describe the public RPCs in detail.

GetPK retrieves the public key for a user in this domain. This function is used only when a user generates an attestation *for* a user in this domain.

SubmitAtt serves two purposes. The first is to allow users in this domain to upload attestations that they have created for other users. The Attestation Server knows the public keys for its local users, and can thus verify that an attestation is valid before accepting it. The second purpose of SubmitAtt is to accept attestations created by other users for a user in this domain. In this situation, the Attestation Server does not know the public key of the issuer of the attestation, so it stores the attestation without verifying it. Attestations of this type are only used as part of a transitive link, and they are verified by the person who wishes to use the transitive link to accept mail.

CheckAuth verifies the authentication tokens included in messages sent by users in this domain. The AS verifies the token's digital signature and that the authentication token has not been used previously. To enforce the latter, the server caches tokens that have been redeemed, but it limits the size of the cache by rejecting tokens older than a fixed window of time (e.g., one week).

FindFriend uses PM to compute the intersection between the supplied set of email addresses and the email addresses of people who have attested to the sender. The authentication token is included as a parameter to FindFriend so that the recipient does not need to issue a separate CheckAuth query.

4.2 Incremental Deployment

One requirement of any practical system is that it be incrementally deployable. Since RE: either accepts an email on the basis of attestations or passes the email to the user's traditional spam filter, deploying RE: does not adversely affect a user's ability to send and receive mail.

The current RE: prototype is deployable on a per-domain basis. A domain that wishes to use RE: must install and configure an Attestation Server plus make various RE: utilities available to its users. In the future, we envision that these utilities will exist as easily downloadable plugins for popular mail clients. To ease early adoption, future versions of RE: may also allow users to specify an alternative AS for situations in which the user's ISP does not run its own AS.

It is important to note that RE: is also incrementally deployable *within* domains themselves. Users who do not download and install the new client software will still be able to send and receive mail; they merely will not be able to take advantage of the reliability offered by RE:.

5 Evaluation

We now turn to evaluating RE:; particularly, we are concerned with its effectiveness at whitelisting email, its potential to reduce false positives, the computational cost of private matching in the system, and the end-to-end email processing throughput attainable when a standard email processing system is augmented with RE:.

5.1 Effectiveness of RE:

The goal of RE: is to accept email, thereby preventing these messages from becoming false positives in a mail rejection system. To ascertain the effectiveness of RE: at meeting this goal, our evaluation must answer three questions:

1. **Overall utility:** What fraction of inbound emails are accepted by RE:? This fraction of email is protected from becoming false positives.
2. **Utility of FoF relationships:** By how much do FoF relationships improve RE:'s effectiveness in accepting email beyond direct attestations?
3. **End-to-end effectiveness:** If RE: were deployed, would it eliminate real false positives produced by today's rejection systems?

Ideally, we would perform a controlled experiment. We would find a large population of email users and have them run *both* RE: and a traditional content-based email rejection system, side-by-side. We would pass a copy of every inbound email to each of the two systems. For RE:, users would create attestations as they saw fit, typically upon sending or receiving email. As the social network (distributedly stored attestations) builds up, we would measure the fraction of accepted email. For the rejection system, we would have users inspect a spam folder that contains messages that the rejection system flagged as spam, and manually note any false positives. Finally, we would directly count the emails accepted by RE: that would have been false positives had the rejection system been deployed alone.

Unfortunately, we do not have a large-scale deployment. We do, however, have email traces from a large corporation and a large university. For the corporate dataset, we also have data on false positives reported to the email system administrators. Using this information, we can approximate the ideal experiments. The main problem, however, is that we must emulate user generation of attestations, as described below.

Email traces. The first trace is taken from a large corporation with well over 80,000 employees, all of whom use email. The trace covers about one month of email and contains over 63 million anonymized messages, where a message with multiple recipients counts as one message per recipient. Because of the corporation's email server architecture and the placement of the monitoring point, email in the trace is nearly exclusively email sent into and out of the corporate network (not mail internal to the network). The corporation's mail server uses a popular content-based spam filter, and the trace indicates whether or not the server's spam filter judged each message to be spam (via a spam score).

The second trace covers New York University's top-level domain, which includes approximately 60,000 email users. The trace covers about one week of email and contains approximately 6 million anonymized messages. NYU uses the SpamAssassin-derived PureMessage filter, and the trace also contains a spam score for each message. Unlike the corporate trace, this university trace also contains messages sent within the domain (e.g., between two students).

A model for creating attestations. Lacking the social network for the users in our email traces, we developed a model for when users might create attestations based on the trace itself and the following assumptions. These rules were applied while processing the traces in chronological order, ignoring all messages that were flagged as spam by the server's content-filter.

1. Every sender attests to every recipient.
2. Every recipient attests to every sender.
3. (Corporate trace only) Every employee attests to every other employee.

We believe that the first rule is reasonable since we assume that the messages considered are not spam. Presumably, users are not sending email to spammers; furthermore, by attesting to the recipient, they will automatically whitelist any replies. The second rule would be dangerous, of course, if the trace contained spam, as the user would whitelist the spammer. Again, we assume that all messages considered are not spam. Even so, this rule might be optimistic, but it is useful for bootstrapping the social network from a trace. The second rule means that the recipient will accept all subsequent mail from that sender (the first message, though, would not be whitelisted as the rule is applied after processing the message). The third rule is specific to the corporate trace; the rule's purpose is to capture the large body of email messages that do *not* appear in the trace—the messages between users within the corporate email network. In a real deployment, these messages and the resulting attestations would provide critical links in the social network.

The evaluation of the corporate dataset contains two sources of error that result directly from limitations of the dataset itself. The first source of error is the attestation model described above, which assumes that every employee at the company attests to every other employee.

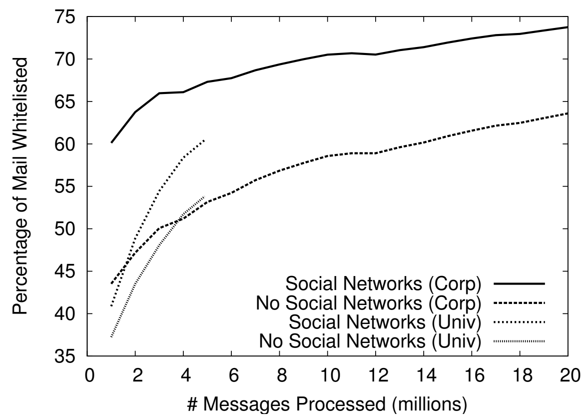

Figure 3: RE: Total coverage

This assumption may cause us to overestimate the hit rate for whitelisting. A more conservative rule for generating intra-company attestations might produce a lower hit rate. The second source of error is that the dataset does not include intra-company messages. This omission may cause us to underestimate the hit rate for whitelisting.

5.1.1 Fraction of Mail Whitelisted

Given these traces and a model for when users would create attestations, our first experiment was to measure the fraction of mail that RE: could whitelist based on social relationships (both direct and FoF). RE:'s ability to whitelist messages gives an indication of what fraction of mail the system can make reliable.

We consider two measures of whitelisting's efficacy. First, we examine total whitelisting coverage, or the fraction of all emails whitelisted. Second, we examine whitelisting coverage for email from *strangers*. When there is no previous message from a sender to a recipient, we term the sender a stranger. It is precisely in this case where FoF relationships are crucial to the acceptance of email.

Figure 3 shows the results of the total coverage experiment. The *x*-axis shows the number of email messages processed (in chronological order) so far, and the *y*-axis shows the percentage of email received so far (i.e., percentage of *x*) that the system was able to whitelist. The attestations are created as the trace is processed. The lower curve (for each trace) is the effectiveness of whitelisting, based on our attestation generation rules above, without considering social links. The upper curve plots the additional effectiveness of the whitelist when one accepts mail from FoFs.

The two traces are plotted on the same graph to ease comparison. Examining the 5 million email mark, where the university trace ends, we see that the two environments have relatively similar results. In the corporate trace, after processing 20 million non-spam mes-

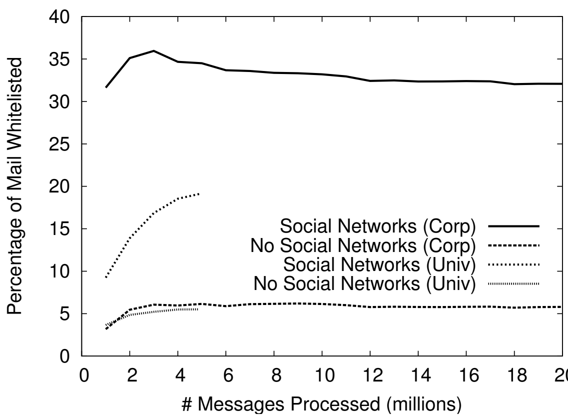

Figure 4: RE: Stranger coverage

sages, social whitelisting can accept almost 75% of email received, approximately 10% more than the direct whitelist. In the one-week university trace, after processing almost 5 million non-spam messages, social whitelisting can accept over 60% of email received, approximately 7% more than direct whitelisting.

Figure 4 shows the results of the strangers coverage experiment, with the x-axis as before, and the y-axis depicting the fraction of email received from strangers that the system was able to whitelist. While the attestation generation rules used in this experiment are the same as before, direct attestation whitelists very few emails, as one expects: only emails from never-before-heard-from senders are considered. In fact, the only emails whitelisted by direct attestation are the first replies sent by strangers in response to emails from recipients. FoF queries are able to whitelist an additional 26% (corporate) or 13% (university) of email from strangers.

5.1.2 False Positives Saved

To make email reliable, RE: must reduce the incidence of false positives. Specifically, RE: can help by accepting email messages from friends or FoFs that would otherwise be classified as spam. As noted above, we were able to obtain false positive data for the corporate environment but not for the university one.

The corporate false positive data is a list of all user-reported false positives during the one-month trace period. Typically, these reports are submitted by users outside of the company who received a bounce message from the company's mail server after the spam filter (incorrectly) determined the user's message was spam. The list contains the anonymized sender and recipient email addresses for each of these reports.

Using this data, we measured the effectiveness of RE: at reducing false positives as follows. We ran the same experiment described in Section 5.1.1, except we did not automatically drop all messages that were flagged as spam in the trace. Instead, for each of these flagged messages, we consulted the list of reported false positives to see if it contained the message's sender-recipient pair. If so, we assumed that the message was not spam, but rather a false positive. This assumption is reasonable if one assumes that spammers did not report false positives to the company in order to get their spam through and that spammers did not forge mail with the exact same sender-recipient pair that appeared in the false positive list.

For each false positive identified during the trace, we recorded if the system would accept that message based on the current social network. Out of 20 million messages, we identified 172 false positives. Of those false positives, approximately 84% would have been whitelisted by RE: using direct attestations and an additional 5% using FoF relationships. We note that the number of reported false positives is most likely much lower than the number of actual false positives.

	Encrypt	Decrypt	HAdd
Paillier	21.8	1.5	.017
ElGamal	1.7	0.8	.010

Table 2: Speed (ms) of 1024-bit public-key operations

5.2 Microbenchmarks

This section provides microbenchmarks for the speed of homomorphic cryptosystems and private matching protocol [18] implementations. Benchmarks were performed on a 3 GHz Intel® Xeon® processor-based computer running in 64-bit mode.

Table 2 shows the performance of Paillier [28] (in fast decryption mode) and ElGamal [17] operations given as the mean over 300 runs across five different keys. HAdd corresponds to the cost of performing a homomorphic addition of plaintexts, which is akin to modular multiplication. Due to its superior performance, we use ElGamal to instantiate our PM protocol. This section's subsequent PM benchmarks reflect this choice of cryptosystem.

There are three stages to evaluate in the Private Matching protocol: (1) the recipient's setup time to construct an encrypted polynomial, (2) the sender's evaluation of this polynomial on its inputs, and (3) the recipient's subsequent recovery of the intersection.

The recipient's setup time is directly measured by k_R encryptions. These encryptions only need to be precomputed *once* per input set (friendship list); the recipient recomputes these encrypted coefficients upon adding or removing an element from its input.

Figure 5 shows the sender-side performance of PM, given as the mean of three runs. We graph performance for varying sender (k_S) and recipient (k_R) input sizes. Re-

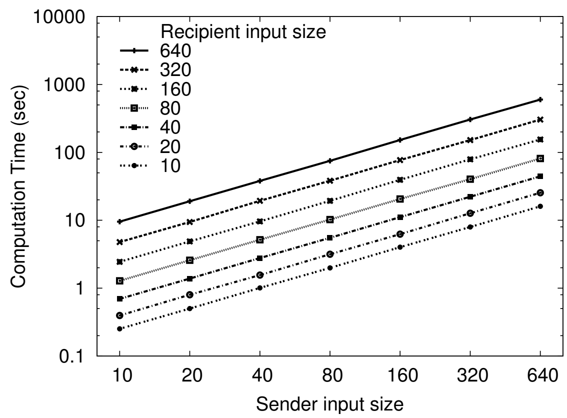

Figure 5: Sender-side PM performance (seconds).

S input size	10	20	40	80	160
R comp. time (s)	.008	.015	.032	.063	.126

Table 3: Time (s) for recipient to recover intersection

call that the sender must perform $O(k_S \cdot k_R)$ operations, i.e., to evaluate each of its inputs on a degree-k_R polynomial. Yet, we see the practical effect of using Horner's Rule for evaluating the polynomial "from the inside out": most homomorphic operations work with exponents of small size. Thus, *in practice*, the performance appears to grow only linearly in both k_R and k_S for reasonably small input sizes (i.e., its slope in log-log scale is 1).

The sender-side computation overhead is acceptable for smaller set sizes. For example, when a sender and recipient each have a set of 40 friends they want to intersect, the sender spends 2.8 seconds to complete the PM computation. At larger sizes, the overhead is more noticeable. To intersect two sets of 160 friends, for example, the sender needs almost 40 seconds of computation.

We make several observations as to this performance: (1) The asymptotic running time of the PM can be reduced to $O(k_S(1+\ln\ln k_R))$ by using multiple low-degree polynomials for potential performance improvements [18]. (2) All operations are completely parallelizable onto multi-processor architectures, as each computation on the sender's k_S inputs are independent of one another. Furthermore, a single domain can deploy multiple Attestation Servers to distribute the load. Given the money that people spend on fighting spam today, we believe the additional cost is reasonable. (3) Alternatively, the cryptographic operations can be partially or fully implemented in hardware for higher performance. (4) Finally, we observe from Section 5.1.1 that a large fraction of messages for certain RE: deployments may be satisfied with direct attestations and thus would never perform FoF queries in the first place.

Last, we examine the recipient's time to recover the intersection (see Table 3). Note that performance is independent of the recipient's input and computation can again be parallelized over k_S processors. We also conclude that most of the protocol's load is placed on the sender, which provides some protection against computational DoS attacks against email recipients.

	Mean	StdDev
Mail Avenger	16.7	0.57
RE: + Mail Avenger	14.4	0.76

Table 4: Delivery throughput (messages/second)

5.3 Throughput

To analyze end-to-end throughput, we augment a standard mail processing system with RE: and measure its impact. For all experiments, the SMTP server is a 2 GHz Intel® Pentium® 4 processor-based computer running Mail Avenger, and all machines are connected via a local area network.

First, we measure how RE: affects the rate at which mail is received by measuring how much time an SMTP server needs to process mail with and without RE:. Our experimental setup consists of our SMTP server and three senders, each running on a different machine. The senders simultaneously bombard the SMTP server with sufficiently many messages to saturate it (e.g., sending 500 messages each as rapidly as possible). In the RE: case, the recipient has attested to all of the senders, and each sender serves as its own AS for authentication checks. Table 4 shows that the addition of RE: reduces throughput by 13.5%.

Second, we measure the time it takes for an AS to process CheckAuth and FindFriend queries from recipients. Because the AS must be contacted for each outgoing mail, this time affects the rate at which a domain's users can send mail. (Note that the processing time required on the client to add an authentication token to each message is the time to generate a digital signature times the number of recipients.) To measure the query-processing rate on the AS, we have a single sender periodically send a mail to our SMTP server. For these tests, the sender's AS is a 3 GHz Intel® Xeon® processor-based computer running in 64-bit mode. The average time to process CheckAuth and FindFriend[5] queries is 0.0516 and 2.85 seconds, respectively. Equivalently, the AS can process 1162 CheckAuth queries per minute or 21.1 FindFriend queries per minute. We note here too the performance observations enumerated in the previous section, which apply to the FindFriend queries.

[5] PM protocol performed using 40 friends in each set.

6 Discussion

We now discuss various design decisions, open questions, and useful enhancements to RE:.

Compromised Senders and Audit Trails. Under normal operation, RE: assumes that a sender's machine is not compromised (see Section 3.7). If this assumption is violated, a spammer's message will appear to come from a legitimate user. Unfortunately, any acceptance system based on identity (e.g., the sender) or even an opaque token (e.g., a stamp) is susceptible to attacks in which the sender is compromised. The problem of compromised senders, though, is just about false negatives: at worst, a compromised sender in RE: can cause a recipient to accept spam.

RE: provides the nice property, however, that the recipient knows with some degree of certainty—due to the authentication token—who sent the received spam and why RE: accepted it. The "why" can be either because of a direct attestation or an FoF relationship. For example, this audit trail might indicate that RE: accepted the spam because the recipient attested to Bob and Bob attested to the sender. Given this information, a recipient can choose to reconsider using his attestations for Bob.

False Negatives. Unfortunately, our anonymized traces do not provide sufficient information to quantify how RE: would affect the false negative rate. Qualitatively, however, RE: is robust against false negatives: a false negative can occur in RE: only if the recipient has attested directly to a spammer or is connected via an FoF chain. In either case, the recipient knows who the responsible party is, providing an important audit trail that describes the trust relationship between recipient and sender.

An attestation for a spammer might exist for several reasons. A user may erroneously attest, he may have been compromised by a virus, or he may be malicious and attest to a spammer intentionally. One simple way to limit the damage caused by machine compromises is to require a password to generate a new attestation.[6]

Mail with Multiple Recipients. Our previous discussion has been confined to mail that is addressed to a single recipient. Many messages, however, contain multiple recipients. These recipients are either listed explicitly, or they are hidden behind a mailing list. In the former case, the sender must generate a different authentication token for each recipient because the attestation server rejects duplicate tokens to prevent replay attacks (i.e., the AS ensures one FoF query per token).

The sender must make sure, however, to send each token to only the corresponding recipient. Sending all of the tokens to all of the recipients would allow one misbehaving recipient to use up all of the tokens to make FoF queries to the AS. Other recipients would get an error when trying to use their tokens because the request would look like a replay attack. Note that the AS has no way to ensure that the recipient is using the "correct" authentication token (the one in which she appears) because the AS has no way to identify the requester as any particular recipient.

Regarding mailing lists, the simplest solution is for users to attest to the mailing list itself. The mailing list moderator is then responsible for ensuring that spam does not make it on to the list. Unfortunately, unmoderated lists still pose a problem—there is no simple solution beyond the current content-based filtering.

Sender Privacy and Profiles. One inherent property of the RE: design is that senders do reveal a subset of their friends to the recipient (i.e., those that intersect with the recipient's list of friends). Thus, a malicious recipient has the opportunity to query the sender with an arbitrary list of friends to discover who has attested to the sender. We can address this issue partially with *sender profiles*, which can allow the sender to control what attestations the sender uses during the FoF query. A sender might choose to have a profile with a restricted set of attestations—i.e., maintaining separate personal and work profiles—so the recipient cannot learn about specific people who have attested to the sender. The potential downside of eliminating attestations from a profile is that the chance of intersecting with the recipient's set of friends, and thus the sender having his mail automatically accepted, is lower.

Length of Social Paths. In this work, we have only explored direct and FoF social relationships, but clearly longer paths bear examination: they are more inclusive of previously unknown senders, and could thus potentially whitelist more email. On the other hand, there is also a tension between this greater coverage and an increased risk of false negatives. As the length of the shortest social path between the sender and recipient increases, it becomes increasingly unclear whether the recipient can safely automatically accept email. If and how private matching protocols would work beyond two degrees of separation is also an open question; furthermore, even if providing privacy with longer chains of trust were possible, it would likely require that other parties be online during the mail transaction besides the sender and recipient.

7 Conclusion

Motivated by the decline in the end-to-end reliability of email at the hands of spam-rejection systems, we have described RE:, a system for automatically accept-

[6] Of course, a sophisticated adversary could use a keystroke sniffer.

ing mail based on its sender. As an acceptance system, RE: is complementary to existing spam-defense systems; it simply bypasses mail rejection systems for senders who are deemed trustworthy. RE: improves upon standard whitelisting approaches in two ways: by preventing sender forgery though the use of an authentication token, and more importantly, by increasing the fraction of email that can be whitelisted through the examination of social relationships, while preserving the privacy of users' correspondents.

We have shown that RE: can reliably deliver the majority of a site's incoming mail; furthermore, augmenting direct-friend attestations with friend-of-friend relationships significantly increases the percentage of accepted mail. More importantly, experiments show that RE: can eliminate a large percentage of false positives produced by an existing content-based spam filter. Our full implementation of RE: does not significantly reduce the rate at which an SMTP server can accept incoming mail or impose a substantial bottleneck on the rate at which users of a domain can send mail.

8 Acknowledgments

The authors thank the anonymous reviewers, their shepherd Dan Rubenstein, Michael Walfish, and Mark Handley for their feedback and comments. The authors also extend a special thanks to Michael Puskar and NYU ITS, Robert Johnson, Marc Foster, and Greg Matthews, without whom the data analysis in this paper would not have been possible. Antonio Nicolosi and Benny Pinkas provided valuable help with the cryptography, and Michael Ryan contributed to the code. This work was partially supported by project IRIS under NSF Cooperative Agreement ANI-0225660.

References

[1] MessageLabs intelligence report: Spam intercepts timeline, July 2005. `http://www.messagelabs.co.uk/`.

[2] Symantec Brightmail AntiSpam. `http://www.brightmail.com/`.

[3] CloudMark. `http://www.cloudmark.com/`.

[4] Distributed checksum clearinghouse, Oct. 2005. `http://www.rhyolite.com/anti-spam/dcc/dcc-tree/dcc.html`.

[5] TrendMicro's RBL+. `http://www.mail-abuse.com/`.

[6] SpamAssassin, . `http://spamassassin.apache.org/`.

[7] SpamCop, . `http://www.spamcop.net/`.

[8] E. Allman, J. Callas, M. Delany, M. Libbey, J. Fenton, and M. Thomas. Domainkeys identified mail (DKIM). Internet Engineering Task Force (IETF) Draft, July 2005.

[9] A. Back. Hashcash, May 1997. `http://www.cypherspace.org/hashcash/`.

[10] T. Blackwell. Why spam cannot be stopped, June 2004. `http://tlb.org/whyspamcannotbestopped.html`.

[11] D. Brickley and L. Miller. FOAF, 2005. `http://xmlns.com/foaf/0.1`.

[12] M. Ceglowski and J. Schachter. LOAF, 2004. `http://loaf.cantbedone.org/`.

[13] J. R. Douceur. The sybil attack. In *First International Workshop on Peer-to-Peer Systems (IPTPS)*, Mar. 2002.

[14] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In *Advances in Cryptology (CRYPTO)*, volume 740 of *Lecture Notes in Computer Science*, 1992.

[15] C. Dwork, A. Goldberg, and M. Naor. On memory-bound functions for fighting spam. In *Advances in Cryptology (CRYPTO)*, volume 2729 of *Lecture Notes in Computer Science*, 2003.

[16] H. Ebel, L.-I. Mielsch, and S. Bornholdt. Scale-free topology of e-mail networks. In *Physcial Review E 66*, 2002.

[17] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[18] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Advances in Cryptology — EUROCRYPT 2004*, May 2004.

[19] J. Golbeck and J. Hendler. Reputation network analysis for email filtering. In *Conference on Email and Anti-Spam (CEAS)*, 2004.

[20] P. Graham. Better bayesian filtering. In *MIT Spam Conference*, Jan. 2003.

[21] P. Graham. A plan for spam, Aug. 2002. `http://www.paulgraham.com/spam.html/`.

[22] A. Gulbransen, P. Vixie, and L. Esibov. RFC 2782: A DNS RR for specifying the location of services (DNS SRV). Internet Engineering Task Force (IETF) Standards Track, Feb. 2000.

[23] J. Kong, P. O. Boykin, B. Rezaei, N. Sarshar, and V. Roychowdhury. Let your cyberalter ego share information and manage spam, May 2005. `http://arxiv.org/abs/physics/0504026`.

[24] B. Laurie and R. Clayton. Proof-of-work proves not to work. *The Third Annual Workshop on Economics and Information Security*, May 2004.

[25] T. Loder, M. V. Alstyne, and R. Wash. An economic answer to unsolicited communication. In *ACM Conference on Electronic Commerce*, May 2004.

[26] J. Lyon and M. Wong. Sender ID: Authenticating E-Mail. Internet Engineering Task Force Draft IETF, Oct. 2004.

[27] D. Mazières. A toolkit for user-level file systems. In *USENIX Technical Conference*, June 2001.

[28] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology — EUROCRYPT 99*, May 1999.

[29] V. Prakash. Razor. `http://razor.sourceforge.net`.

[30] S. Radicati. Anti-spam market trends, 2003–2007. Radicati Group Study, 2003. `http://www.radicati.com/`.

[31] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A bayesian approach to filtering junk E-mail. In *Learning for Text Categorization: Papers from the 1998 Workshop*, Madison, Wisconsin, 1998. AAAI Technical Report WS-98-05.

[32] M. Walfish, J. D. Zamfirescu, H. Balakrishnan, D. Karger, and S. Shenker. Distributed quota enforcement for spam control. In *3rd Symposium on Networked System Design and Implementation (NSDI)*, San Jose, CA, May 2006.

[33] M. W. Wong. Sender authentication: What to do, July 2005. `http://spf.pobox.com/whitepaper.pdf`.

[34] W. Yerazunis. The spam-filter accuracy plateau at 99.9% accuracy and how to get past it. In *MIT Spam Conference*, Jan. 2004.

[35] P. Zimmermann. *The Official PGP User's Guide*. MIT Press, Cambridge, 1995.