An Email Signature System for Eliminating Spam

Robert Rose

November 9, 2004

Abstract

For my project, I propose to create an email client that incorporates public key / private key encryption to digitally sign email as it is being sent. The client will also perform automatic signature verification upon receipt of an email and warn the user if the signature could not be verified. A simple server system will also be constructed that is backwards-compatible with the existing Internet email infrastructure. Signature verification will allow users to quickly identify spam email.

1 Summary

As everyone who has ever used the Internet knows, spam, aka "unwanted email," is a serious problem. Although a human can easily recognize spam and ignore it, programatically detecting and removing spam is a computationally complex problem due to the nature of the Internet email infrastructure. Recognizing spam programatically is difficult because:

1. Email is open–anyone can send anyone else an email.
2. Email is built on trust–when you send an email to someone you trust that the servers involved will get your email to the right destination.
3. Email is unverified–when you receive an email you have to trust that the sender is who they say they are.

Spammers (senders of spam email) prey on these three attributes of email by forging who they say they are and abusing the trust of the servers that take part in the email system. Most often a spam is not sent from the user that it claims to be sent from–a spam may claim to be sent from a friend or colleague, a system administrator, or an account manager at your bank. Sometimes a spam may not claim to be sent by any valid person or entity at

all–the return address is completely random. Because it is so easy to forge where an email originated from programmatically eliminating spam based on who it was sent by, under the current Internet email infrastructure, is unrealistic.

This project seeks to eliminate spam email by extending the Internet email system to include signature verification. If the true sender of an email can be verified, then spam email can quickly be programatically identified because it did not originate from a verifiable sender. Extensions added to the Internet email system will be backwards-compatible with legacy email systems, so this solution can be iteratively deployed across the Internet without disruption.

2 Email Signature Protocol

Each email contains two parts: a list of headers and a content body. RFC 822 Section 4[1] specifies the format of the headers. The Email Signature Protocol, in accordance with RFC 822, adds an email-extension header to each email named *X-Sig* that stores the signature generated by the sender of the email. For example:

```
To: Dr. Cetin Koc <koc@ece.orst.edu>
From: A+ Student <rose@ece.orst.edu>
X-Sig: v1 ABC12345
Subject: Quick Question...

When will you hold the midterm? -robert
```

2.1 Email Signature Generation

Each client that takes part in the Email Signature Protocol is responsible for generating the *X-Sig* header for every email they send. The header contains two fields: 1) the version number of the signature that is about to follow and 2) the signature. These two fields are delimited by a space. At this time the only valid version number is "v1".

The signature is generated by performing an MD5 hash of the body of the email and encrypting it using the sender's private key.

So that other's may verify the signature, the public key corresponding to private key that was used must be registered with the email server that client is using to send email.

[1] http://www.faqs.org/rfcs/rfc822.html

2.2 Email Signature Verification

When a client receives an email that contains the *X-Sig* header, the client is responsible for generating it's own MD5 hash of the body of the email. The receiving client then decrypts the signature field of the *X-Sig* header using the sender's public key and checks it against the hash that it generated. If the hashes match, then the sender has been validated. If not, then the email was forged.

To retrieve the sender's public key the client inspects the *To* header of the email and parses out the domain name from where the email claims to have originated. The client then performs a DNS query for the primary MX record in the domain (the MX record specifies the primary mail server for a domain). The client contacts the server in the MX record making an HTTP request for the supposed sender's public key. For example:

```
$ dig MX ece.orst.edu
...
ece.orst.edu.          IN      MX      ece.orst.edu.

$ curl "http://ece.orst.edu/GetKey?addr=rose@ece.orst.edu"
ssh-dss AAAAB3NzaC1kc3MAAACBAPXoN+dbFAh5enocx
OgC3IrZlHV2Q4cwvHU6Dk2883ywIdVvhs3UNEhlc04WRu
Lr1tsyyIDWNJqf0EGxTZcNuqi6oiWlDbln1t5vP/F...=
rose@ece.orst.edu
```

This key may be cached on the client system so that subsequent verifications do not require additional network traffic.