

# X-Sig for Postfix Installation Guide

Robert Rose

March 6, 2005

## Abstract

This document summarizes X-Sig and explains how to set up automated X-Sig signing using the Postfix mail server.

## 1 Introduction

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 programatically eliminating spam based on who it was sent by, under the current Internet email infrastructure, is unrealistic.

X-Sig 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. The extensions X-Sig adds to the Internet email system are backwards-compatible with legacy email systems, so X-Sig can be iteratively deployed across the Internet without disruption.

## 2 The X-Sig Email Signature Protocol

Email contains two parts: a list of headers and a content body. RFC 822 Section 4<sup>1</sup> specifies the format of the headers. The X-Sig Protocol, in accordance with RFC 822, adds an email-extension header to email named *X-Sig* that stores the signature generated by the sender of the email. An additional header, *X-Sig-Version* specifies the version of the X-Sig Protocol being used. For example:

```
To: Email User <user@domain1.com>
From: Robert Rose <rose@domain2.com>
X-Sig-Version: 1.0 (MD5-RSA)
X-Sig: P1g/TuZIIHXXpZSMGEA6vfJGt7/OfxcT2z0x2S7
      hit7KQwWobTuK0bmc7enxsMDRV0c588tnlKkoGI6W+9K+YA==
Subject: Email signing example
```

This is a test!

### 2.1 Email Signature Generation

Each email system (client or email server) that takes part in the X-Sig Protocol is responsible for generating the X-Sig headers for every email it sends. The first header is *X-Sig-Version*, which for this version of the protocol must be *1.0 (MD5-RSA)* which specifies protocol version 1.0, MD5 hash generation, and RSA encryption. The second header is the signature itself, which is to be Base64 encoded<sup>2</sup> with line feeds removed.

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.

---

<sup>1</sup><http://www.faqs.org/rfcs/rfc822.html>

<sup>2</sup><http://en.wikipedia.org/wiki/Base64>

## 2.2 Email Signature Verification

When a client receives an email that contains the *X-Sig* header, the client is responsible for generating its 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 domain.com
...
domain.com.          IN      MX      domain.com.

$ curl "http://domain.com/sig/?addr=rose@domain.com"
-----BEGIN PUBLIC KEY-----
MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAMg5QBig9p2U5/YaQxQdl++H9eQLrDZz
ole5IZpdrzk8QW5VgUwZGX99mkOGmxtCngFggbrR0ojkavncbcnD6tkUCAwEAAQ==
-----END PUBLIC KEY-----
```

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

## 3 X-Sig for Postfix

X-Sig signing is implemented in Postfix as a Postfix Filter<sup>3</sup>.

### 3.1 Prerequisites

Before this implementation of X-Sig for Postfix can be used your email server must meet the following requirements:

1. Postfix 2.0 or greater
2. Python 2.3 or greater

---

<sup>3</sup>[http://www.postfix.org/FILTER\\_README.html](http://www.postfix.org/FILTER_README.html)

3. OpenSSL 0.9.7b or greater

## 3.2 Installation

1. Create a user to run Postfix filters.

If your Postfix installation is not already configured to run filters (it is most likely not) then you will need to ready your system with a filter user. Create a user named `filter` with no password and the shell set to `/usr/bin/false` (or equivalent). On Linux this can be accomplished with the `useradd` or `adduser` utilities.

2. Create a directory for filter processing

Create a directory `/var/spool/postfix/filter` and give the `filter` user exclusive permissions to write in this directory:

```
drwx-----  2 filter  wheel      68  6 Mar 21:31 filter
```

3. Create a directory for X-Sig files

It is necessary to create a directory on your system where X-Sig key files and scripts will be stored. The recommended location is `/var/x-sig`.

4. Install X-Sig scripts

Two scripts, `x-sig.sh` and `x-sig.py` need to be placed in the X-Sig directory you creating in the previous step. Make the permissions on these files executable:

```
-rwxr-xr-x  1 root  wheel  2436  6 Mar 21:30 x-sig.py
-rwxr-xr-x  1 root  wheel   656  6 Mar 21:56 x-sig.sh
```

5. Edit X-Sig scripts for your system

At the top of the `x-sig.py` file you may need to change the following parameters:

```
class constants:
    keystorage = '/var/x-sig'
    hostnames = ['cafwap.net', 'mail.cafwap.net']
```

Change `keystorage` to the location of the X-Sig directory you created previously.

Change `hostnames` to the list of hostnames that your mail server is known by.

#### 6. Configure X-Sig filter in Postfix

In the Postfix `master.cf` file, typically located in `/etc/postfix`, add the following lines under the `smtp` server:

```
smtp      inet  n       -       n       -       -       smtpd
  -o content_filter=filter:dummy
```

Also, add a filter that will run the `x-sig.sh` script:

```
filter    unix  -       n       n       -       -       pipe
  flags=Rq user=filter argv=/var/x-sig/x-sig.sh -f ${sender} -- ${recipient}
```

#### 7. Finally, restart Postfix

```
# postfix stop
# postfix start
```

Check your syslogs to make sure there were not any errors.

### 3.3 Configuration

If you leave your Postfix system in the state from the previous section it will continue to operate normally but will not be able to sign any emails because keys for your users do not exist (yet). To complete installation, it is necessary to configure keys for users that are interested in participating in the X-Sig protocol<sup>4</sup>.

Keys, public and private, are stored in the same directory you configured for X-Sig (`/var/x-sig` if you picked the suggested location). Keys are created using the `openssl` command line utility. To create a new private key (used for signing), run the following in the X-Sig directory you configured, where `user` is the username of the user that wants to sign messages:

```
openssl genrsa -out user.priv_key
```

---

<sup>4</sup>Hopefully all of your users!

Next, generate the public key that matches this private key (again, change *user* to the appropriate username):

```
openssl rsa -in user.priv_key -pubout -out user.pub_key
```

Change the permissions on the private key so that only the `filter` user has permission to read the private key:

```
-r----- 1 filter wheel 493 5 Mar 18:22 rose.priv_key
-rw-r--r-- 1 root  wheel 182 5 Mar 20:17 rose.pub_key
```

If the private key is compromised anyone would be able to sign emails as that user.

### 3.4 Verification

If you've done everything correctly up until this point you should now be able to test out X-Sig. Open up your favorite mail client and send an email via your Postfix server somewhere. In the syslog you should see the following entry appear from X-Sig:

```
Mar 6 22:48:12 X-Sig processing email from rose@domain.com
```

If you do not see this message appear in your syslog, check your maillog and other system log files to make sure Postfix is not reporting a problem.

Note that there is a deficiency with Postfix filters: they can only be run on mail sent via the Postfix SMTP daemon. This means that mail sent via the loopback interface will not be filtered. Your mail client must be configured to connect via the system's public email port if you are testing this from the same system. (Running `mail` on the command line of the Postfix server will not work as it uses the loopback interface).

### 3.5 Troubleshooting

If you are having difficulties getting the X-Sig system working, you can do the following to verify the individual components are working as expected.

#### 3.5.1 Testing `x-sig.py`

`x-sig.py` expects a complete SMTP email to be written to standard input, and writes the resulting signed email to standard out. You can test it on the command line by running `x-sig.py` and then pasting an email to the script's standard input. Standard input is terminated in most terminals with a Control-D. You should see something like the following:

```
/var/x-sig> ./x-sig.py
To: rose@domain.com
From: rose@domain.com
Subject: sign test
```

```
test
To: rose@domain.com
From: rose@domain.com
Subject: sign test
X-Sig-Version: 1.0 (MD5-RSA)
X-Sig: IRtKF2Y05ECOD85imV5FYCzgK5NK7DkvJ0uDQH
      Km/XaipIvfgeclTyi/RDBWisllgGkKz6EKGTbGFmS6xhUiRw==
```

```
test
```

If the X-Sig headers are not being written then the constants in `x-sig.py` are either incorrectly configured or the keys for the user you are testing have not been created (or are in the wrong location).

### 3.5.2 Testing x-sig.sh

`x-sig.sh` is called from Postfix and expects to be passed the sender and recipients of the email as arguments, and the contents of the email over standard input. The script does not output anything; but if executed properly will re-invoke Postfix instructing it to re-process the email with the modified headers. You can test the script yourself by executing the following:

```
/var/x-sig> ./x-sig.sh -f rose@domain.com rose@domain.com
To: rose@domain.com
From: rose@domain.com
Subject: sign test
```

```
test
```

In the syslog and maillog you should see messages that the email was processed.

## 4 Appendix

### 4.1 x-sig.sh

```
#!/bin/sh
```

```
#
# X-Sig email signing filter. It is meant to be invoked as follows:
#     /path/to/script -f sender recipients...
```

```
INSPECT_DIR=/var/spool/postfix/filter
SENDMAIL="/usr/sbin/sendmail -i"
```

```
# Exit codes from <syssexits.h>
EX_TEMPFAIL=75
EX_UNAVAILABLE=69
```

```
# Clean up when done or when aborting.
trap "rm -f in.$$" 0 1 2 3 15
```

```
# Start processing.
cd $INSPECT_DIR || {
    echo $INSPECT_DIR does not exist; exit $EX_TEMPFAIL; }
```

```
cat >in.$$ || {
    echo Cannot save mail to file; exit $EX_TEMPFAIL; }
```

```
/var/x-sig/x-sig.py <in.$$ >x.$$ ; mv -f x.$$ in.$$ || {
    echo Message content rejected; exit $EX_UNAVAILABLE; }
```

```
SENDMAIL "$@" <in.$$
```

```
exit $?
```

## 4.2 x-sig.py

```
#!/usr/bin/python
'''
x-sig.py
```

```
Copyright (c) 2005 Robert Rose
Permission to use this file in any manner is OK as long as
this copyright remains in tact.
```

```
Email filter that appends X-Sig headers to emails.
'''
```

```

import random
import syslog
import sys
import os
import base64

class constants:
    keystorage = '/var/x-sig'
    hostnames = ['cafwap.net', 'timonkie.cafwap.net']

def hostnames():
    c = os.popen('hostname')
    hosts = constants.hostnames
    hosts.append(c.read().strip())
    return hosts

def sign(msg, keyfile):
    c = os.popen3('openssl md5 | openssl rsautl -sign -pkcs -inkey %s' % keyfile)
    c[0].write(msg)
    c[0].close()
    error = c[2].read()
    if(error != ''):
        syslog.syslog('Error while creating signature: %s' % error)
    sigraw = c[1].read()
    sig64 = base64.encodestring(sigraw).strip()
    return sig64.replace('\n', '')

def main():
    line = sys.stdin.readline()
    email = ''
    while line != '':
        email = email + line
        line = sys.stdin.readline()

    allheaders, body = email.split('\n\n', 1)

    splitheaders = allheaders.split('\n')
    headers = {}
    for header in splitheaders:

```

```

    try:
        headername, headervalue = header.split(':', 1)
        headers[headername] = headervalue.strip()
    except ValueError:
        pass

emailfrom = headers['From']
try:
    extra, emailfrom = emailfrom.split('<', 1)
    emailfrom = emailfrom.replace('>', '')
except:
    pass

emailfromuser, emailfromhost = emailfrom.split('@', 1)
try:
    emailfromhost, extra = emailfromhost.split(' ', 1)
except:
    pass

syslog.syslog('X-Sig processing email from %s@%s (%f)' % (emailfromuser, emailfromhost, emailfrom))

newemail = ''
if(emailfromhost in hostnames() and 'X-Sig' not in headers):
    keyfile = '%s/%s.priv_key' % (constants.keystorage, emailfromuser)
    signature = sign(body, keyfile)
    newemail += allheaders
    newemail += '\nX-Sig-Version: 1.0 (MD5-RSA)'
    newemail += '\nX-Sig: %s\n\n' % signature
    newemail += body
else:
    newemail = email

sys.stdout.write(newemail)

if __name__ == "__main__":
    main()

```