Secure Mailing Lists using GnuPG

Protima Chhabra

GIAC Security Essentials Certification (GSEC)
Practical Assignment Version 1.4

April 19, 2004

Abstract

Electronic mail has become an essential means of communication. Businesses have come to rely on it for their day to day operations. However, security vulnerabilities in clear text email, is a major obstacle in email becoming the preferred channel, especially for exchanging sensitive information. Email signature and encryption with PGP/GPG can provide a feasible method for secure communication. Signing emails can protect against tampering, and encrypting them can prevent eavesdropping. Any given situation may demand one or the other, or both.

In order to use GnuPG to communicate securely, the involved parties must first exchange their public keys securely. The sender of the email then signs it with his private key and encrypts it with the receiver's public key. The receiver upon receiving the email, decrypts it with his private key and verifies the signature with the sender's public key. This works well when there is only one receiver. However, in order to send the email to multiple people, the email needs to be encrypted with every receiver's public key, which can be quiet cumbersome. This implies that the email cannot be sent to a mailing list, unless the mailing list server is capable of performing the authentication and encryption operations on behalf of the sender. This paper discusses the design and implementation of such a server.

1. Introduction

Clear text email is susceptible to many security problems. Messages may be intercepted in transit, where they may be read, altered, and/or dropped. Authentication and encryption of email is one practical method used to minimize the risk posed by these vulnerabilities. Encryption of email between individuals has been a common practice for some years. However, there are currently no standardized techniques, or readily available applications, to allow for encrypting email addressed to mailing lists. This paper will outline a simple technique for implementing secure mailing lists.

Following are the requirements for implementing a system, handling secure mailing lists:

Cryptography

First and foremost is strong cryptography. GnuPG, GNU Privacy Guard, GNU's open source tool for secure communication and data storage, can be used to encrypt data and create digital signatures. GnuPG uses hybrid ciphers, which requires users to provide their public-key to the server. See http://www.gnupg.org/gph/en/manual.html for more information on GnuPG.

Authentication

Users should be able to sign up for a list and submit their public-key to the server as automatically as possible. They should also be able to download public key of the mailing lists from the server. System needs to have a method to verify the identities of the potential members who submit their keys.

The system should also be able to ensure the authenticity of all incoming messages. All incoming messages to the list need to be digitally signed by the sender. Any mail that is not should be dropped or returned.

Subscribers should be able to ensure the authenticity of the messages coming from the system (i.e, system should sign all messages with its private-key).

Encryption and Transport
System should provide a means to transmit signed and/or encrypted mailing list messages.

2. BEFORE SNAPSHOT

2.1 Mailing list and List server

Typical mailing lists are managed by computer programs, referred to as mailing list servers. Mailing list servers interconnect to the global email system as an MTA, Mail Transfer Agent. A list server reads the messages, and copies them to everyone subscribed to the list. Most list servers manage a minimal of two email addresses for each mailing list. specifically:

- **Server address.** The server address is used to process administrative commands, such as list subscriptions and un-subscriptions.
- **List address.** The list address is used for the list itself. When the server receives an email sent to the list address, it automatically copies it to everyone currently subscribed to the list.

There are three general types of mailing lists, each with a different type of operation:

- **Unmoderated lists.** These allow any subscriber to send an email to the mailing list for distribution to all subscribers, in real-time without any screening. Most lists are unmoderated.
- **Moderated lists.** These lists have a “moderator” a human being that checks all email sent to the list to make sure it does not contain inappropriate content before the server distributes it to subscribers.
- **One-way lists.** These lists are used to distribute information from a central source. Only one person or organization can send email to the list for distribution to all subscribers. These types of lists are often used by organizations to distribute information.

There are a number of list server implementations in common use, including:

- *Listserv* is one of the most widely used list servers on the Internet, with a commercial version used by many large organizations.
- *Majordomo* is widely used in its freeware version by community and small organizations.
- *Mailman* is an open source GNU mailing list manager.
- *Listproc* is widely used, primarily as a commercial product.

2.2 Mailing Lists in Sendmail

There are several ways to manage a mailing list. A simple and very common approach in Unix is direct use of aliases in the MTA; often Sendmail. This method works well for a smaller number of recipients that do not change often, as every change in the list involves human intervention. Secure mailing lists will typically have a small group of people who trust each other, such a system is adequate for our study.

Creating a mailing list using S*endmail* simply involves creating some new aliases. There are usually three addresses associated with the list:

- The alias for the list itself; for example, `listname`
- `listname-request`: This is the address to which subscription, unsubscribe, and list information requests should be sent. It points to the person or persons that manage the list
- `owner-listname:` This is the address used by *sendmail* to return errors about the list. This usually points to the same addresses as listname-*request*

Detailed configuration of the 'aliases' file is beyond the scope of this paper, but typically when there are more than a few members in the list, aliases file includes a file containing the list of all the members.

```
<listname>: ":include:/<path to the list file>/<list file>"

<listname>-request: <user>

owner-<listname>: <listname>-request
```

Once the aliases file is set up, aliases database is created by running the command `'newaliases'.`

2.3 Flow of unsigned/unencrypted Email

When an email for the list comes in, the list server reads the message and copies it to everyone subscribed on the list.

2.4 Flow of GPG signed/encrypted Email

To understand why a typical list server will not be able to do GPG signed/encrypted email, let us first understand the exchange of secure email between two people.

In order to exchange emails securely with a person, the sender and the receiver first need to exchange public keys securely. Thereafter the sender will encrypt the message with the receiver's public key, and sign it with his own private key. Upon receiving the email, the receiver verifies the signature on the email with the sender's public key and then decrypts it with his own private key.

To send the same email to multiple people, the sender first needs to exchange public keys securely with every receiver. He will then have to encrypt the same email for each receiver separately, and sign it with his private key before sending it. Clearly this is very cumbersome and time consuming.

The mailing list server as described above doesn't know anything about keys, encryption, decryption, etc. The objective of this study is to impart to the server the capabilities it will need to handle such emails, and educate the client with the working of the system.

3. DURING SNAPSHOT

3.1 Operation Theory

Every list on the list server, in addition to having a list of members, has a pair of private and public keys. To subscribe to this list, every member needs to get the public key of the list, and provide his own public key to the server. Thereafter the keys are validated and signed. Key signing and establishing a web of trust are key to the success of this system. In a small network of people, the administrator of the server could simply call the user and verify the fingerprint of the key submitted, and then sign the user key, and add the user to the mailing list. However this would not work for a larger group. For a larger network, the email address and the key of the subscriber could be put on a web page accessible only to the members of the mailing list. Members can then cast a vote of trustworthy or untrustworthy for the prospective member. If enough number of trustworthy votes are received, the subscriber can then become a member of the mailing list. For more details on key signing, refer to http://www.cryptnet.net/fdp/crypto/gpg-party.html.

Once the subscriber becomes a member, he can send signed and encrypted messages to all the members on the list. This is how it works:

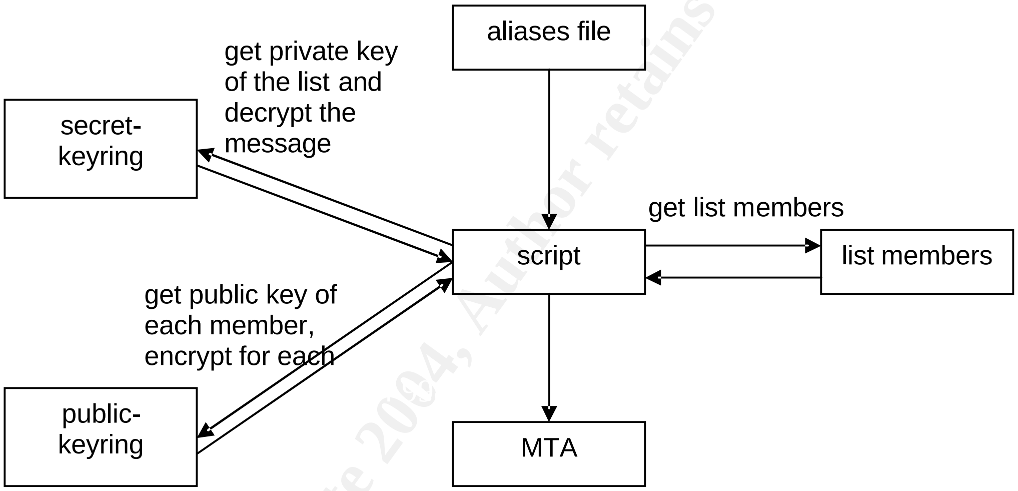
- The sender encrypts the message with the public key of the mailing list, signs it with his own private key, and sends it
- The server upon receiving the message verifies the signature with the sender's public key, and then decrypts with the list's private key
- The server then encrypts the message with every receiver's public key, signs with the lists private key, and sends it
- Each receiver then verifies the signature with the list's public key, and decrypts the message with his own private key.

The system can handle signed, encrypted, signed and encrypted, and unsigned and unencrypted message.

If the sender is not a member of the list, he can download the public key of the list, and send encrypted messages to the list, provided the server has validated his public key. Note that the sender will not receive those messages.

3.2 System Architecture

When an email for the list comes in, the server invokes a script. The script then gets the list of recipients from the appropriate file, keys from the appropriate keyring(s), verifies and/or decrypts the message, signs and/or encrypts the message appropriately and forwards the message to all the members of the list.


aliases file
get private key of the list and decrypt the message
secret-keyring
get list members
script
list members
get public key of each member, encrypt for each
public-keyring
MTA

The system can also include a web based interface which allows creation of new lists, deletion of lists, subscription to a list, and un-subscription to a list. Subscription to a list involves being able to securely submit your public key, and download the public key of the list. Screenshots of such an interface are presented in Appendix C.

3.3 Step by Step Guide

In this section I will go through all the pieces of the system in detail. Some of the details might be specific to the system used for testing.

3.3.1 Getting the system ready

3.3.1.1 Operating System

Red Hat is the most commonly used Linux distributions, and is the choice of many system administrators for mail. However, Red Hat will soon begin charging a license fee for patches and upgrades. As a result, I selected Gentoo 1.4.3 for my development and testing. However, the system works well on Red Hat 9 as well. I have not tested it on other flavors of UNIX, but I expect that it will work well on those as well, except there may be some issues getting various PERL modules to work.

Installation instructions for Gentoo on X86 systems can be found at
http://www.gentoo.org/doc/en/gentoo-x86-install.xml

Installation instructions for Red Hat 9 can be found at
https://www.redhat.com/docs/manuals/linux/RHL-9-Manual/install-guide/

3.3.1.2 SMTP Server

There is a wide variety of SMTP servers available out there, but Sendmail is by far the most widely used. Exim is another one that is becoming quite popular. For the purposes of testing the system, sendmail-8.12.8-4 was used. Details on secure configuration of MTA are beyond the scope of this paper.

Installation and configuration information for Sendmail can be found at
http://www.sendmail.org

3.3.1.3 GNU Privacy Guard (GnuPG)

GnuPG or Gnu Privacy Guard is GNU’s tool for secure communication and data storage.

GnuPG or GPG source and/or binaries can be downloaded from
http://www.gnupg.org/(en)/download/index.html

Installation and configuration information can be found at
http://webber.dewinter.com/gnupg_howto/english/GPGMiniHowto.html

It is fully compatible with the newer versions of PGP. To communicate with old versions of PGP 2.x, check out
http://www.gnupg.org/gph/en/pgp2x.html

Installation and configuration steps for GnuPG on the system are outlined below

Please note that you have to be logged in as root to install GnuPG.

```
$ su
Password:
$ cd /usr/local/src
$ wget ftp://ftp.gnupg.org/gcrypt/gnupg/gnupg-1.2.4.tar.gz
$ tar xzvf gnupg-1.2.4.tar.gz
$ ./configure
$ make
$ make install
```

Important: GPG should be installed as setuid(root). This is necessary to lock memory pages. Locking memory pages prevents the operating system from writing them to disk and thereby keeping the secret keys really secret. If you get no warning message about insecure memory your operating system supports locking without being root. The program drops root privileges as soon as locked memory is allocated. To setuid(root) permissions on the gpg binary, do either

```
$ chmod u+s /path_to_gpg/gpg
```

OR

```
$ chmod 4755 /path_to_gpg/gpg
```

3.3.1.4 PERL

Typical install of PERL-5.8.0.

3.3.1.5 PERL modules

The following PERL modules need to be installed. All these modules can be downloaded from http://search.cpan.org..Essentially 8 modules need to be installed, and the process is very simple and same for all of them. For details of the PERL modules, check Appendix B.

Module Name	Module Description	Dependencies	Version
Mail::GnuPG	To process email with GPG	GnuPG::Interface	Mail-GnuPG-0.07
GnuPG::Interface	Perl Interface to GnuPG	Class:MethodMaker	GnuPG-Interface-0.33
Class:MethodMaker	Generic methods for object oriented PERL		Class-MethodMaker-2.00
MIME::Entity	Class for parsed-and-decoded MIME message	Mail::Field Mail::Header Mail::Internet Unicode::Map Unicode::String IO::Stringy	MIME-tools-6.200_02
MIME::Parser	Class for parsing MIME messages		
Mail::Field	Base class for manipulation of mail header fields		MailTools-1.60
Mail::Header	For manipulationof mail RFC822 compliant headers		
Mail::Internet	For manipulation of Internet format (RFC 822) mail messages		

Unicode::Map	For mapping character sets from and to utf16 unicode		Unicode-Map-0.112
Unicode::String	String of Unicode characters (UCS2/UTF16)		Unicode-String-2.07
IO::Stringy	I/O on in-core objects like strings and arrays		IO-stringy-2.109

Download the tar files for all the modules, and start installing in the order reversed from the one in the table.

```
$ su
Password:
$ cd /usr/local/src
$ tar xzvf <module_name>.tar.gz
$ cd <module_name>
$ perl Makefile.PL
$ make
$ make test
$ make install
```

3.3.1.6 Apache

The system was running a typical install of apache-1.3.28. You can download the sources from
http://www.apache.org.

The documentation on the site is more than adequate, but to make sure that you can run PHP, follow the instructions at:
http://www.php.net/manual/en/install.apache.php

3.3.1.7 PHP

The system was running php-4.3.3. No special configurations were made.

3.3.2 Installing and configuring the custom software

The source code for the scripts can be found in Appendix A. The scripts in themselves are configurable, but it is easier to explain the setup with an example.

The list that will be created here is `gpglist@bar.com`, and the members are, `webctl@bar.com`, `pchhabra@bar.com`, `mlg@bar.com`.

Important: Steps below require you to be logged in as root

File/Directory Structure: The directory structure can be configured differently in the scripts.

File/Directory	Description
/etc/mail/smail	Base directory
/etc/mail/smail/scripts	Directory containing scripts
/etc/mail/smail/scripts/smail	Script that processes the email
/etc/mail/smail/keyrings	Directory containing keyrings. Note that it is not the default directory, /root/.gnupg
/etc/mail/smail/lists	Directory containing files with the list members. Each list has its own file with the members belonging to it. The filename is the same as the listname. For our example list, the file will be called “gpglist@oak.bbn”
/usr/local/apache/htdocs	Directory containing the PHP based web interface for list and key management. The files are index.php and Common.php
/tmp/smail	Directory used by the web interface for saving information needed to process requests for list creation or deletion, member subscription or un-subscription. Note: It is important that this directory and all the sub-directories and files be owned by the user the web server is running as.
/tmp/smail/<delete_requests>	File containing requests to delete list(s) or member(s).
/tmp/smail/<listname>	File containing information about the list owner, number of members, email addresses of the members, etc. Note: This file is created by the web interface when a request for a list or new members comes in.
/tmp/smail/d_<listname>	Directory containing the public keys of the members belonging to the list <listname>, submitted through the web interface. Note: The directory is created by the web interface.
/tmp/smail/lists	Directory containing the public keys of all the lists available. These keys are served through the web interface for people to download.

3.3.2.1 Script setup

Create the top level directory and the directory for the script. Copy the script in the appropriate directory.

```
$ mkdir /etc/mail/smail
$ mkdir /etc/mail/smail/scripts
$ cp smail /etc/mail/smail/scripts/
$ cp wrapper /etc/mail/smail/scripts/
```

Set the setuid bit of the wrapper so that it executes as root. This is important because the keyring is owned by root

```
$ chmod u+s /etc/mail/smail/scripts/wrapper
```

Ownership and permissions can be an issue if not handled carefully.

In order for the script to get executed when an email comes in, two important changes need to be made.

3.3.2.1.1 Update aliases database

As root, edit the aliases file to call the script. It is usually located in /etc or /etc/mail.

```
gpglist:"|/etc/mail/smail/scripts/wrapper /etc/mail/smail/scripts/smail"
```

Next, update the aliases database by executing the following command

```
$ newaliases
```

3.3.2.1.2 Make the script available to Sendmail

Sendmail uses a restricted shell 'smrsh' to execute programs, which are limited to be in a single directory. Check the manpage for smrsh for the directory location. Once in the directory, create a link to the script.

```
$ cd /usr/adm/sm.bin
$ ln -s /etc/mail/smail/scripts/smail smail
```

3.3.2.2 List Setup

3.3.2.2.1 Create 'lists' directory

```
$ mkdir /etc/mail/smail/lists
```

3.3.2.2.2 Create the lists file

Create a separate file for each list. **This step will have to be repeated for every new list that is created**.

```
$ cd /etc/mail/smail/lists
$ vi gpglist@bar.com
```

`gpglist@bar.com` should contain the following

```
      webctl@bar.com
      pchhabra@bar.com
      mlg@bar.com
```

3.3.2.3 Keyring(s) Setup

The assumption is that you are setting up a server to do GPG based mailing lists, so you should be familiar with the workings of GPG. If not, refer the link below for details.
http://webber.dewinter.com/gnupg_howto/english/GPGMiniHowto.html

For even more understanding, refer to http://www.gnupg.org/gph/en/manual.html

Important: Remember we are not using the default keyring. This implies that in every gpg command, we have to include the option

```
      --homedir /etc/mail/smail/keyrings
```

3.3.2.3.1 Create ‘keyrings’ directory

Create the keyrings directory.

```
$ mkdir /etc/mail/smail/keyrings
$ chmod 700 /etc/mail/smail/keyrings
```

3.3.2.3.2 Generate key pair for a list

As mentioned earlier, we need a key pair for each list. So lets generate one for `gpglist@bar.com`. **This step will have to be repeated for every new list that is created**.

```
$ gpg –homedir /etc/mail/smail/keyrings –gen-key
```

When prompted for a passphrase, hit enter i.e. no passphrase.

Next, generate a revocation certificate for the key. Though the revocation certificate is not needed for the functioning of this system, but if your private key is compromised or lost, this revocation certificate may be published to notify others that the public key should no longer be used.

```
$ gpg –homedir /etc/mail/smail/keyrings –gen-revoke gpglist@bar.com
```

This would be a good time to export the public key of the list for members to download. By default, the key is exported in a binary format, but this can be inconvenient when the key is to be sent though email or published on a web page. So lets generate one in ascii format. We want to save it in the /tmp/smail/lists directory, so that the web interface can access it and allow potential members to download it.

```
$ gpg –homedir /etc/mail/smail/keyrings --armor --output
/tmp/smail/lists/gpglist@bar.com –export gpglist@bar.com
```

Make sure that the permissions on the public key file allow everyone to read it, or the web interface will not be able to read it. If not, do

```
$ chmod a+r /tmp/smail/lists/gpglist@bar.com
```

3.3.2.3.3 Add a member to the list

Adding a member to the list involves getting their key and validating it before trusting. Earlier we briefly touched the subject of key signing, but the issues there are beyond the scope of this paper. We are going to assume that we trust the subscriber, so we will sign it with our own key.

Import subscriber’s public key

```
$ gpg –homedir /etc/mail/smail/keyrings --import
/tmp/smail/gpglist@bar.com/<members_public_key_file>
```

Sign subscriber’s public key

```
$ gpg –homedir /etc/mail/smail/keyrings --sign-key <members_email_id>
```

3.3.3 Front End Setup

There are two main PHP scripts – index.php and Common.php. Install them in /usr/local/apache/htdocs or whatever is the ServerRoot directory on your system.

Create the following directories.

```
$ mkdir /tmp/smail
$ mkdir /tmp/smail/lists
```

Change the ownership of the directory to that of your webserver e.g. nobody or apache. Assuming it is running as nobody

```
$ chown –R nobody:nobody /tmp/smail
```

The screenshots of the front end are available in Appendix C.

4. AFTER SNAPSHOT

Mail components just like the rest of the Internet were developed in an ad-hoc fashion. As a result the mail servers can be configured in numerous ways. For this study, I have chosen a single configuration. However the scripts that do the real work do not depend on the configuration of the server.

To make matters worse, between windows and UNIX, we have seemingly infinite mail clients. These mail clients differ in the way they handle MIME, PGP, GPG, etc. We also have versions of PGP that are not compatible with GPG.

4.1 Caveats

Before I discuss the results of this study, I would like to go over some caveats on the way.

4.1.1 Ownership and Permissions

A word about ownership and permissions is in order here. Sendmail does privileged tasks as root, and other tasks as the user, typically ‘mail’, configured through the ‘DefaultUser’ directive in the Sendmail configuration file. For GPG to work properly, it is important that the script is executed as the user who owns the keyring, which in this case would have to be mail. So the following options were available:

Option 1: Grant user ‘mail’ shell access. Since home directory for mail is typically /var/spool/mail, the keyring would be in /var/spool/mail/.gnupg. This setup is clearly not very secure.

Option 2: No shell access for mail, but set ‘mail’ as the owner of the keyring. This would mean that every time the administrator added a new list or a member to the keyring, he would have to reset the permissions on the keyring. This can be quite cumbersome for the administrator.

Option 3: Run the script as a different user than mail i.e. set the setuid bit of the script to run it as whatever user you choose. For security reasons Sendmail usually uses ‘smrsh’, a restricted shell, for executing programs. The programs that smrsh can execute have to be explicitly specified. Setuid can be made available to smrsh, but allowing Sendmail scripts to change user for execution, without proper environment checking, is not a secure practice. So we insert a layer of protection in the form of a wrapper. The wrapper sets the appropriate secure environment for executing the script.

I went with option 3 for setting up the system, and chose ‘root’ as the user to execute the script. I chose ‘root’, because all my mailing lists include files are in /usr/MailingList, which is owned by root. If you want to setup a separate user to do this, all you need to do is create a user, and configure the script to point to user’s home directory as the base directory.

4.1.2 Passphrase for the Private Keys of the Lists

Lets talk about passphrases for a bit. Remember in section 3.3.2.3.2, *Generate key pair for a list*, I asked you to leave the passphrase blank. From the perspective of security, a passphrase is essentially the weakest points in public encryption systems, but it is still a lock for the private key. If someone gets a hold of your private key, they need your passphrase to unlock it. So why did I not set one for the private key of the list? Well in order for the script to decrypt messages sent to it, it would need to use the lists private key. If the key was locked, it would need the passphrase. For the script to have the passphrase, I would need to have them on the hard drive somewhere, where the script can access it. In the event the system gets hacked, and someone gets the private key, I am going to assume, they got my passphrase as well, and will discard all the keys. Hence I chose not to have any, at least until I can figure out a better way of providing the passphrase to the script.

All said and done, the email encryption and/or signing is an added step towards security, and it is only as secure as your system. If your operating system, or the MTA, or any other piece of software that you are using provides a backdoor, none of this is any use. If the system ever gets hacked, you will have to regenerate all list keys, and all the members of all the lists, will have to update their key rings.

4.2 Setup and Results

Now that we have gone through various caveats, let me talk about the test setup and results.

As mentioned before, the mailing list server is running **Gentoo-1.4.3**, and Sendmail-8.12.8-4.

Mail clients used for testing were Pine and Kmail. The results were great. I could send signed, encrypted, signed and unencrypted, and of course plain emails, with and without attachments, flawlessly. I have not forgotten windows, and will test some windows email clients some time soon.

I also setup a list server on **Red Hat 9**. PERL that comes as a Red Hat rpm may not support setuid and setguid. In that case you can get perl-suid at ftp://216.254.0.38/linux/redhat/updates/9/en/os/i386/perl-suidperl-5.8.0-88.3.i386.rpm

This package allows for more secure running of PERL setuid scripts by checking for tainted code.

5. Further Improvements

There is always room for improvement, especially in something which is at this point is still being brainstormed. In addition there is a learning curve for the average user, who needs to be able to generate and manage keys, and setup their clients to use PGP/GPG.

Some key conceptual issues that I think need to be addressed are incompatibility between GPG and versions of PGP-2.x and key authentication for larger groups.

This module is really a first cut at doing something like this. The list passphrase problem discussed earlier needs to be addressed. Also the module needs to be tested with other MTAs. Also some testing with windows mail clients need to be done.

Some minor improvements to this module can be a script for automating the set up of the system, an interactive script which makes it easier to create new lists, add new members, and allows importing and signing of keys belonging to the subscribers. One could definitely write a more secure wrapper.

References

[1] Robbins, Daniel, Gentoo Linux 1.4 Installation Instructions, January 23, 2004.
URL: http://www.gentoo.org/doc/en/gentoo-x86-install.xml

[2] Moore, Sandra A, Red Hat Linux 9: Red Hat Linux x86 Installation Guide, 2003
URL: https://www.redhat.com/docs/manuals/linux/RHL-9-Manual/install-guide/

[3] Costales and Allman, Sendmail, Third Edition, O'Reilley, December 2002

[4] Mathew Copeland, Joergen Grahn, and David A. Wheeler, The GNU Privacy Handbook, Version 1.1
URL: http://www.gnupg.org/gph/en/manual.html

[5] Brenno J.S.A.A.F. de Winter, Gnu Privacy Guard (GnuPG) Mini Howto (English), Version 0.1.3 May 17, 2002
http://webber.dewinter.com/gnupg_howto/english/GPGMiniHowto.html

[6] Spier, Rober, Mail::GnuPG, Version: 0.07
URL: http://search.cpan.org/~rspier/Mail-GnuPG-0.07/GnuPG.pm

[7] Eryq [ZeeGee Software Inc], MIME::Entity, Version: 5.404
URL: http://search.cpan.org/~eryq/MIME-tools-5.411a/lib/MIME/Entity.pm

[8] Several other PERL modules
URL: http://search.cpan.org

[9] Details of installing PHP with Apache on Unix, Jan 30, 2004
URL: http://www.php.net/manual/en/install.apache.php

Appendix A: Source Code

```
#!/usr/bin/perl

#############################################################
# Name: smail                                               #
# Author: Protima Chhabra <pchhabra@bbn.com>                #
# Version: 0.1                                              #
# Last updated: 02/09/2004                                  #
# Function: Script to do GPG based mailing lists.           #
#############################################################

use strict;

use MIME::Parser;
use GnuPG::Interface;
use IO::Handle;
use Mail::GnuPG;

use vars qw($basedir $keydir $listdir $scriptdir $dbgfile $ent $head $subject $dmg @recp
$DEBUG);

#############################################################
# Configurable portion of the script

#$basedir = "";            # base directory
#$keydir = "";             # directory containing the keyrings
#$listdir = "";            # directory containing files with the
                           # lists of members belonging to each list
#$scriptdir = "";          # directory containing the scripts
#$dbgfile = "";            # debug file

$DEBUG = 0;         # set to 1 for debugging

# Default directories
$basedir = "/etc/mail/smail";
$keydir = "$basedir/keyrings";
$listdir = "$basedir/lists";
$scriptdir = "$basedir/scripts";
$dbgfile = "$scriptdir/debug";
#############################################################

sub send_signed_and_encrypted;
sub send_signed;
sub send_encrypted;
sub send_plain;
sub send_error;
sub send_mail;

open(D, ">$dbgfile") if($DEBUG);

# Create a new MIME parser:
my $parser = new MIME::Parser;
$parser->output_to_core(1);

# Read the MIME message:
$ent = $parser->read(\*STDIN) or die "couldn't parse MIME stream";

# extract the header
$head = $ent->head;

# get the subject if provided
$subject = $head->get('Subject') || "";

# create a new mail entity
$dmg = new Mail::GnuPG(keydir => $keydir);

# construct the name of the file containing the list of the members
my $rcvr = $head->get('To');
```

```
chomp($rcvr);
$rcvr =~ s/(.*)\<(.*)\>(.*)/$2/;
my $file = $listdir . "/" . $rcvr;

print D "list file -- $file\n" if $DEBUG;

# now get the members belonging to this list from the list file
open(F, $file);
@recp = <F>;
close(F);
if($DEBUG) {
	print D "Members of list $file\n";
	print D @recp;
}

if($dmg->is_encrypted($ent)) {
	# encrypted message
	# success is set to 0 if the message can be successfully decrypted
	# if the message is both signed and encrypted, then keyid and email
	# are defined
	my ($success, $keyid, $email) = $dmg->decrypt($ent);
	if($DEBUG) {
		print D "$success, $keyid, $email\n";
		# last_message contains the error message if there was
		# trouble decrypting
		print D "\n******* last_message after decryption ************\n";
		print D @{$dmg->{last_message}};
		print D "\n******* end last_message after decryption ********\n";
	}
	if(!$success) {
		# successfully decrypted the msg. now see if the message
		# is signed
		if($keyid && $email) {
			print D "++ signed and encrypted\n" if $DEBUG;
			send_signed_and_encrypted();
		} else {
			# message was not signed
			print D "++ encrypted only\n" if $DEBUG;
			send_encrypted();
		}
	} else {
		my $error = "Your message did not reach some or all of the intended
recipients.\nThe message could not be decrypted.";

		send_error($head->get('From'), $error, "@{$dmg->{last_message}}");
	}
} else {
	if($dmg->is_signed($ent)) {
		print D "++ signed only\n" if $DEBUG;
		my ($success, $keyid, $email) = $dmg->verify($ent);
		if($DEBUG) {
			print D "$success, $keyid, $email\n";
			print D "\n******* last_message after signature verification *\n";
			print D @{$dmg->{last_message}};
			print D "\n******* end last_message after verification ******\n";
		}
		send_signed();
	} else {
		# message is neither encrypted, nor signed
		print D "++ Plain\n" if $DEBUG;
		send_plain();
	}
}

sub send_signed_and_encrypted() {
	print D "++ Sign and Encrypt\n" if $DEBUG;
	foreach my $recp(@recp) {
		chomp($recp);
		  my $deme =  MIME::Entity->build(Type => "Multipart/Mixed",
			                    From => $head->get('From') ,
				         To => $head->get('To'),
                                    Subject => $subject);
                $deme->add_part($dmg->{'decrypted'});
                my $mg = new Mail::GnuPG(keydir => $keydir);
                if(!$mg->mime_signencrypt($deme, $recp)) {
                        # message successfully signed and encrypted
                        $deme->smtpsend(To => $recp);
                } else {
                        # problems encrypting the message
                        if($DEBUG) {
                                print D "\n******* $recp: last_message after encryption  ****\n";
                                print D @{$mg->{last_message}};
                                print D "\n**************************************************\n";
                        }
                        my $error = "There is an encrypted message for you, but I cannot send it
to you for the reason below.";
                        send_error($recp, $error, "@{$mg->{last_message}}");
                }
        }
}

sub send_signed() {
        # sign only
        foreach my $recp(@recp) {
                chomp($recp);
                my $deme = $ent->dup;
                my $mg = new Mail::GnuPG(keydir => $keydir);
                $mg->mime_sign($deme);
                $deme->smtpsend(To => $recp);
        }
}

sub send_encrypted() {
        # encrypt only
        foreach my $recp(@recp) {
                chomp($recp);
                 my $deme =  MIME::Entity->build(Type => "Multipart/Mixed",
                                          From => $head->get('From') ,
                                       To => $head->get('To'),
                                    Subject => $subject);
                $deme->add_part($dmg->{'decrypted'});
                my $mg = new Mail::GnuPG(keydir => $keydir);
                if(!$mg->mime_encrypt($deme, $recp)) {
                        # message was successfully encrypyed
                        $deme->smtpsend(To => $recp);
                } else {
                        # problems encrypting the message
                        if($DEBUG) {
                                print D "\n******* $recp: last_message after encryption  ****\n";
                                print D @{$mg->{last_message}};
                                print D "\n**************************************************\n";
                        }
                        my $error = "There is an encrypted message for you, but I cannot send it
to you for the reason below.";
                        send_error($recp, $error, "@{$mg->{last_message}}");
                }
        }
}

sub send_plain() {
        # send plain message
        foreach my $recp(@recp) {
                chomp($recp);
                my $deme = $ent->dup;
                $deme->smtpsend(To => $recp);
        }
}

sub send_error() {
        my ($recp, $err, $lastmsg) = @_;

        my $contents = $err;
        $contents .= "\n\tSubject: " . $subject ."\tSent: " . $head->get('Date');
        $contents .= "\n\n-------------- Details are included -----------------\n";
        $contents .= $lastmsg;
         my $deme =  MIME::Entity->build(
                        From => $head->get('To') ,
                        To => $head->get('From'),
                        Subject => "Undeliverable: " . $subject,
                    Data => [$contents]);

        $deme->smtpsend(To => $recp);
}
```

```
<?php
        /************************************************************
        * Name: index.php                                          *
        * Author: Protima Chhabra <pchhabra@bbn.com>               *
        * Version: 1.0                                             *
        * Last Updated: 02/16/2004                                 *
        * Supporting files: Common.php                             *
        *                                                          *
        * PHP based script that interacts with subscribers/members.*
        * It provides the following functions: create a new list,  *
        * delete an existing list, add members to an exisitng list,*
        * delete members from an existing list, and allow users to *
        * download public keys of lists.                           *
        *                                                          *
        * NOTE: By design the frontend itself WILL NOT CHANGE      *
        * anything. It is only a means to gather information. Every*
        * change requires human intervention                       *
        ************************************************************/

        include "Common.php";

        $params = $_REQUEST;
        $action = $params[action];
        switch($action) {
                case 'add_list':
                        /* display the page to gather information about
                        a new list. once we have some basic information
                        about the list like list name, owner, number of
                        members, etc, we will display the form to gather
                        email addresses     and the public keys of the members */
                        disp_header($params, NULL);
                        disp_sec_header();
                        disp_add_new_list($params);
                        disp_submit("Add List");
                        break;
                case 'Add List':
                        /* we have basic information about the list, now
                        gather member information. But before that,
                        validate the information gathered so far */
                        $return = validate_add_new_list($params, $ERROR);
                        if(!empty($return)) {
                                /* the data had some errors. display the
                                form again to fix it. */
                                disp_header($params, $return);
                                disp_sec_header();
                                disp_add_new_list($params);
                                disp_submit("Add List");
                        } else {
                                /* no errors. get member email addresses,
                                and their public keys */
                                disp_header($params, NULL);
                                disp_sec_header();
                                disp_add_new_list($params);
                                disp_get_list_members($params);
                                disp_submit("Add List Members");
                        }
                        break;
                case 'Add List Members':
                        /* now we have basic and member information
                        about the list. validate everything */
                        $return = validate_add_new_list($params, $ERROR);
                        if(!empty($return)) {
                                disp_header($params, $return);
                        }
                        $ret = get_list_members($params, $tmpdir, $ERROR);
                        if(!empty($ret)) {
                                disp_header($params, $ret);
                                disp_sec_header();
                                disp_add_new_list($params);
                                disp_get_list_members($params);
                                disp_submit("Add List Members");
                        } else {
                                /* everything looks good. send an email to
                                the admin about a request for a new mailing list*/
                                disp_header($params, "Your request for a new list has been
submitted. Someone will contact you shortly.");
                                disp_sec_header();
                                send_mail("Request for a new list", $params, $admin);
                        }
                        break;
                case 'delete_list':
                        /* request to delete a list. display form to
                        get information */
                        disp_header($params, NULL);
                        disp_sec_header();
                        disp_get_list($params);
                        disp_submit("Delete List");
                        break;
                case 'Delete List':
                        /* have information to delete. verify it */
                        $ret = del_list($params, $delreqfile, $ERROR);
                        if(!empty($ret)) {
                                /* problems with information. get it fixed */
                                disp_header($params, $ret);
                                disp_sec_header();
                                disp_get_list($params);
                                disp_submit("Delete List");
                        } else {
                                /* all good. send an email to the admin
                                with this delete list request */
                                disp_header($params, "Your request for deleting list
'$params[lname]' has been noted. Someone will contact you about the request before any
action is taken upon it.");
                                send_mail("Request for deleting a list", $params, $admin);
                                disp_sec_header();
                        }
                        break;
                case 'add_member':
                        /* request to add a member. display form
                        to get relevant information */
                        disp_header($params, NULL);
                        disp_sec_header();
                        disp_get_list($params);
                        $params[num] = 1;
                        disp_get_list_members($params);
                        disp_submit("Add Member");
                        break;
                case 'Add Member':
                        /* have information to add  a member. verify it */
                        $ret = add_member($params, $tmpdir, $ERROR);
                        if(!empty($ret)) {
                                /* problems with information. get it fixed */
                                disp_header($params, $ret);
                                disp_sec_header();
                                disp_get_list($params);
                                $params[num] = 1;
                                disp_get_list_members($params);
                                disp_submit("Add Member");
                        } else {
                                /* all good. send an email to the admin
                                with this new member request */
                                disp_header($params, "Your request for subscription to list
'$params[lname]' has been noted. Someone will contact you about the request before any
action is taken upon it.");
                                disp_sec_header();
                                send_mail("Request for subscribing to list $params[lname]",
$params, $admin);
                        }
                        break;
                case 'delete_member':
                        /* request to delete a member. display form
                        to get information */
                        disp_header($params, NULL);
                        disp_sec_header();
                        disp_del_member($params);
                        disp_submit("Delete Member");
                        break;
                case 'Delete Member':
                        /* have information to delete. verify it */
                        $ret = del_member($params, $delreqfile, $ERROR);
                        if(!empty($ret)) {
                                /* problems with information. get it fixed */
                                disp_header($params, $ret);
                                disp_sec_header();
                                disp_del_member($params);
                                disp_submit("Delete Member");
                        } else {
                                /* all good. send an email to the admin
                                with this delete member request */
                                disp_header($params, "Your request for un-subscription from list
'$params[lname]' has been noted. Someone will contact you about the request before any
action is taken upon it.");
                                disp_sec_header();
                                send_mail("Request for un-subscribing from list $params[lname]",
$params, $admin);
                        }
                        break;
                case 'disp_list_keys':
                        /* show page public keys of all the lists */
                        disp_header($params, NULL);
                        disp_sec_header();
                        disp_list_keys($tmpdir);
                        break;
                default:
                        disp_header($params, NULL);
                        disp_sec_header();
                        break;
        }
        disp_footer();

        function del_member($params, $delreqfile, $ERROR) {
                if(empty($params[lname])) {
                        return($ERROR['mandatory']);
                }
                if(!preg_match("/^[A-z0-9\-_@.]+$/", $params[lname])) {
                        return($ERROR[invalid_list]);
                }
                $fh = fopen($delreqfile, "a");
                $date = `date`;
                fwrite($fh, $date);
                fwrite($fh, " ++ Delete member $params[mememail] from $params[lname]\n");
                fclose($fh);
        }

        function add_member($params, $tmpdir, $ERROR) {
                if(empty($params[lname])) {
                        return($ERROR['mandatory']);
                }
                if(!preg_match("/^[A-z0-9\-_@.]+$/", $params[lname])) {
                        return($ERROR[invalid_list]);
                }
                $keydir = $tmpdir . "/d_" . $params[lname];
        if(!is_dir($keydir)) {
            mkdir($keydir);
        }
        $lstfile = $tmpdir . "/" . $params[lname];
        $fh = fopen("$lstfile", "a");
        $email = "email_0";
        $key = "key_0";
        if(!empty($params[$email])) {
            if(!empty($params[$key])) {
                if(!preg_match("/^[A-z0-9\-_@.]+$/", $params[$email])) {
                          return($ERROR[invalid_email]);
                    }
                fwrite($fh, "$params[$email]\n");
                $keyfile = "$keydir/$params[$email]";
                $kfh = fopen("$keydir/$params[$email]", "w");
                fwrite($kfh, "$params[$key]");
                fclose($kfh);
            } else {
                return($ERROR[missing_key]);
            }
        } else {
            if(!empty($params[$key])) {
                return($ERROR[missing_email]);
            }
        }
        fclose($fh);
        return;
    }

    function del_list($params, $delreqfile, $ERROR) {
        if(empty($params[lname])) {
            return($ERROR['mandatory']);
        }
        if(!preg_match("/^[A-z0-9\-_@.]+$/", $params[lname])) {
            return($ERROR[invalid_list]);
        }
        $fh = fopen($delreqfile, "a");
        $date = `date`;
        fwrite($fh, $date);
        fwrite($fh, " ++ Delete list $params[lname]\n");
        fclose($fh);
    }

    function get_list_members($params, $tmpdir, $ERROR) {
        $keydir = $tmpdir . "/d_" . $params[lname];
        if(is_dir($keydir)) {
            if($dh = opendir($keydir)) {
                while(($file = readdir($dh)) !== false) {
                    if(is_file("$keydir/$file")) {
                        unlink("$keydir/$file");
                    }
                }
                closedir($dh);
            }
        } else {
            # make directory for storing keys
            mkdir($keydir);
        }
        $lstfile = $tmpdir . "/" . $params[lname];
        $fh = fopen("$lstfile", "w");
        fwrite($fh, "#$params[ownername], $params[owneremail],
$params[ownerphone]\n");

        for($cnt = 0; $cnt < $params[num]; $cnt++) {
            $email = "email_" . $cnt;
            $key = "key_" . $cnt;
            if(!empty($params[$email])) {
                if(!empty($params[$key])) {
                    if(!preg_match("/^[A-z0-9\-_@.]+$/", $params[$email])) {
                              return($ERROR[invalid_email]);
                    }
```

```
                                        fwrite($fh, "$params[$email]\n");
                                        $keyfile = "$keydir/$params[$email]";
                                        $kfh = fopen("$keydir/$params[$email]", "w");
                                        fwrite($kfh, "$params[$key]");
                                        fclose($kfh);
                                } else {
                                        return($ERROR[missing_key]);
                                }
                        } else {
                                if(!empty($params[$key])) {
                                        return($ERROR[missing_email]);
                                }
                        }
                }
                fclose($fh);
                return;
        }

        function validate_add_new_list($params, $ERROR) {
                /* make sure that all essential information, like listname, number of members,
                 owner of the list, and owners email, has been obtained */
                if((empty($params[lname])) || (empty($params[num])) ||
(empty($params[ownername])) || (empty($params[owneremail]))) {
                        return($ERROR['mandatory']);
                }
                /* validate form data */
                /* list name is a valid email address */
                if(!preg_match("/^[A-z0-9\-_@.]+$/", $params[lname])) {
                        return($ERROR[invalid_list]);
                }
                /* number of members is an integer */
                if(!is_numeric($params[num])) {
                        return($ERROR['integer']);
                }
                /* owner's email address is valid */
                if(!preg_match("/^[A-z0-9\-_@.]+$/", $params[owneremail])) {
                        return($ERROR[invalid_email]);
                }
                /* phone number is valid */
                if(!empty($params[ownerphone])) {
                        if(!preg_match("/^[0-9-\(\) ]+$/", $params[ownerphone])) {
                                return($ERROR[invalid_phone]);
                        }
                }
                return;
        }

        function send_mail($action, $params, $admin) {
                   $subject = $action;
                   foreach ($params as $key => $value) {
                            if((! empty($value)) && ($key)){
                                      $content .= "$key\t\t$value\n";
                            }
                   }
                   mail($admin, $subject, $content);
        }
?>
```

```
<?php
        /******************************************************************
        * Name: Common.php
        * Author: Protima Chhabra <pchhabra@bbn.com>
        * Version: 1.0
        * Last updated: 02/16/2004
        * Supports index.php
        ******************************************************************/

        /******************************************************************/
        /* Configurable part of the script */

        /* directory to save all incoming requests */
```

```
       $tmpdir = "/tmp/smail";

       /* file to save all delete requests */
       $delreqfile = "$tmpdir/delete_requests";

       /* admin who will receive email about all requests */
       $admin = "pchhabra@bar.com";
       /****************************************************************/

       $ERROR = array(
             'mandatory' => "ERROR: All fields marked * are mandatory",
             'integer' => "ERROR: # members should be an integer",
             'invalid_list' => "ERROR: Invalid characters in list name",
             'invalid_email' => "ERROR: Invalid characters in email address",
             'invalid_phone' => "ERROR: Invalid characters in phone number",
             'missing_key' => "ERROR: No key given for a member",
             'missing_email' => "ERROR: Key provided for a member with no email address",
       );

       function disp_list_keys($tmpdir) {
?>
<tr><td>
<table border=1 cellpadding=8 cellspacing=4 align=center><tbody>
<?php
             $keydir = $tmpdir . "/lists";
             if(is_dir($keydir)) {
                        if($dh = opendir($keydir)) {
                          while(($file = readdir($dh)) !== false) {
                                 $str = file_get_contents("$keydir/$file");
                                 if(strlen($str) > 0) {
                                        print "<tr valign=top><td font
class=tile>$file</td>";
                                        print "<td><pre>$str</td></tr>";
                                 }
                          }
                    }
             }
?>
</tbody></table></td></tr>
<?php
       }
       function disp_del_member($params) {
?>
<tr><td>
<table border=0 cellpadding=8 cellspacing=4 align=center><tbody>
<tr valign=top>
       <td font class=caption>List*</td>
       <td font class=caption><input type=text name=lname value="<? print $params[lname]
?>"></td>

       <td font class=caption>Member [email address]*</td>
       <td font class=caption><input type=text name=mememail value="<? print
$params[mememail] ?>"></td>
</tr>
</tbody></table></td></tr>
<?php
       }
       function disp_get_list($params) {
?>
<tr><td>
<table border=0 cellpadding=8 cellspacing=4 align=center><tbody>
<tr valign=top>
       <td font class=caption>List*</td>
       <td font class=caption><input type=text name=lname value="<? print $params[lname]
?>"></td>
</tr>
</tbody></table></td></tr>
<?php
       }

       function disp_add_new_list($params) {
```

```
?>
<tr><td>
<table border=0 cellpadding=8 cellspacing=4 align=center><tbody>
<tr valign=top>
        <td font class=caption>List*</td>
        <td font class=caption><input type=text name=lname value="<? print $params[lname]
?>"></td>
        <td font class=caption># Members*</td>
        <td font class=caption><input type=text name=num size=3 value="<? print $params[num]
?>"></td>
</tr><tr valign=top>
        <td font class=caption>Owner of the List*</td>
        <td font class=caption><input type=text name=ownername value="<? print
$params[ownername] ?>"></td>
        <td font class=caption>Email*</td>
        <td font class=caption><input type=text name=owneremail value="<? print
$params[owneremail] ?>"></td>
        <td font class=caption>Phone</td>
        <td font class=caption><input type=text name=ownerphone value="<? print
$params[ownerphone] ?>"></td>
</tr>
</tbody></table></td></tr>
<?php
        }

        function disp_submit($value) {
?>
<table border=0 cellpadding=8 cellspacing=4 align=center><tbody>
<tr><td font class=caption>
<input type=submit name=action value="<? print $value ?>">
</td>
</tbody></table></td></tr>
<?php
        }

        function disp_get_list_members($params) {
?>
<tr><td>
<table border=0 cellpadding=8 cellspacing=4 align=center><tbody>
<tr valign=top>
        <td font class=caption>Please provide below the email address(s) and the public
key(s)</td>
</tr>
</tbody></table></td></tr>

<tr><td>
<table border=1 cellpadding=8 cellspacing=4 align=center><tbody>
<tr>
        <td font class=caption>Email</td>
        <td font class=caption>Public Key</td>
</tr>
<?php
        for($cnt = 0; $cnt < $params[num]; $cnt++) {
                $email = "email_" . $cnt;
                $key = "key_" . $cnt;
                print "<tr>
                        <td font class=content><input type=text name=$email
value=$params[$email]></td>
                        <td font class=content colspan=4><textarea name=$key rows=5
cols=45>$params[$key]</textarea></td>
                        </tr>\n";
        }
?>
</tbody></table></td></tr>
<?php
        }

        function disp_avail_lists($tmpdir) {
?>
<tr><td>
<table border=0 cellpadding=8 cellspacing=4 align=center><tbody>
```

```
<tr><td font class=caption colspan=2>Available Lists</td></tr>
<?php
              if($dh = opendir($tmpdir)) {
                     while(($file = readdir($dh)) !== false) {
                            if(is_file("$tmpdir/$file")) {
                                   print "<tr><td font class=content>$file</td></tr>";
                            }
                     }
                     closedir($dh);
?>
</tbody></table></td></tr>
<?php         }

       }

       function disp_sec_header() {
?>
<tr><td>
<table border=0 cellpadding=4 cellspacing=4 align=center><tbody><tr>
<td font class=bcontent>Lists[<a href=index.php?action=add_list>Add</a>/<a
href=index.php?action=delete_list>Delete]</a></td>
<td font class=bcontent>Members[<a href=index.php?action=add_member>Add</a>/<a
href=index.php?action=delete_member>Delete]</td>
<td font class=bcontent><a href=index.php?action=disp_list_keys>Download List Keys</a></td>
</tr></tbody></table>
</td></tr>
<?php
       }


        function disp_header($params, $error) {
                $str = "
<HTML>
 <HEAD>
 <TITLE>Secure List Management </TITLE>
 <meta http-equiv=\"Content-Type\" content=\"text/html; charset=iso-8859-1\">
 <style type=\"text/css\">
 <!--
  BODY {
        BACKGROUND-COLOR: #ffffff
 }
  A {   TEXT-DECORATION: none }
  A:visited {   COLOR: #0000cf; TEXT-DECORATION: none }
  A:link {      COLOR: #0000cf; TEXT-DECORATION: none }
  A:active {    COLOR: #0000cf; TEXT-DECORATION: underline }
  A:hover {     COLOR: #0000cf; TEXT-DECORATION: underline }
  OL {  COLOR: #333333; FONT-FAMILY: tahoma,helvetica,sans-serif }
  UL {  COLOR: #333333; FONT-FAMILY: tahoma,helvetica,sans-serif }
  P {   COLOR: #333333; FONT-FAMILY: tahoma,helvetica,sans-serif }
  BODY {        COLOR: #333333; FONT-FAMILY: tahoma,helvetica,sans-serif }
  TD {  COLOR: #333333; FONT-FAMILY: tahoma,helvetica,sans-serif }
  TR {  COLOR: #333333; FONT-FAMILY: tahoma,helvetica,sans-serif }
  TH {  COLOR: #333333; FONT-FAMILY: tahoma,helvetica,sans-serif }
  FONT.title {  BACKGROUND-COLOR: white; COLOR: #363636; FONT-FAMILY:
        tahoma,helvetica,verdana,lucida console,utopia; FONT-SIZE: 12pt; FONT-WEIGHT: bold
}
  FONT.sub {    BACKGROUND-COLOR: white; COLOR: #000000; FONT-FAMILY:
        tahoma,helvetica,verdana,lucida console,utopia; FONT-SIZE: 10pt }
  FONT.layer {  COLOR: #ff0000; FONT-FAMILY: courrier,sans-serif,arial,helvetica; FONT-
SIZE: 8pt; TEXT-ALIGN: left }
  FONT.error {  COLOR: #ff0000; FONT-FAMILY: courrier,sans-serif,arial,helvetica; FONT-
SIZE: 10pt; TEXT-ALIGN: left }
  TD.title {    BACKGROUND-COLOR: #FFFFFF; COLOR: #555555; FONT-FAMILY:
          tahoma,helvetica,verdana,lucida console,utopia; FONT-SIZE: 12pt; FONT-WEIGHT:
bold; HEIGHT: 20px; TEXT-ALIGN: center }
  TD.subtitle {    BACKGROUND-COLOR: #FFFFFF; COLOR: #555555; FONT-FAMILY:
          tahoma,helvetica,verdana,lucida console,utopia; FONT-SIZE: 10pt; FONT-WEIGHT:
bold; HEIGHT: 20px; TEXT-ALIGN: left }
  TD.error {    BACKGROUND-COLOR: #FFFFFF; COLOR: #000000; FONT-FAMILY:
          tahoma,helvetica,verdana,lucida console,utopia; FONT-SIZE: 10pt; HEIGHT: 20px;
TEXT-ALIGN: left }
```

```
  TD.sub {      BACKGROUND-COLOR: #DCDCDC; COLOR: #555555; FONT-FAMILY:
          tahoma,helvetica,verdana,lucida console,utopia; FONT-SIZE: 10pt; FONT-WEIGHT:
bold; HEIGHT: 18px; TEXT-ALIGN: center }
  TD.caption {      BACKGROUND-COLOR: #FFFFFF; COLOR: #555555; FONT-FAMILY:
          tahoma,helvetica,verdana,lucida console,utopia; FONT-SIZE: 10pt; FONT-WEIGHT:
bold; HEIGHT: 18px; TEXT-ALIGN: left  }
  TD.bcontent {  BACKGROUND-COLOR: white; COLOR: #000000; FONT-FAMILY:
        tahoma,arial,helvetica,verdana,lucida console,utopia; FONT-SIZE: 10pt; FONT-
WEIGHT:bold; TEXT-ALIGN: left; VERTICAL-ALIGN: top }
  TD.content {  BACKGROUND-COLOR: white; COLOR: #000000; FONT-FAMILY:
        tahoma,arial,helvetica,verdana,lucida console,utopia; FONT-SIZE: 10pt; TEXT-ALIGN:
left; VERTICAL-ALIGN: top }
  TD.one {  BACKGROUND-COLOR: white; COLOR: #000000; FONT-FAMILY:
        tahoma,arial,helvetica,verdana,lucida console,utopia; FONT-SIZE: 10pt; TEXT-ALIGN:
left; VERTICAL-ALIGN: top }
  TD.two {  BACKGROUND-COLOR: #E0E0E0; COLOR: #000000; FONT-FAMILY:
        tahoma,arial,helvetica,verdana,lucida console,utopia; FONT-SIZE: 10pt; TEXT-ALIGN:
left; VERTICAL-ALIGN: top }
  TD.default {  BACKGROUND-COLOR: WHITE; COLOR: #000000; FONT-FAMILY:
        tahoma,arial,helvetica,verdana,lucida console,utopia; FONT-SIZE: 8pt; }
  TD.border {   BACKGROUND-COLOR: #cccccc; COLOR: black; FONT-FAMILY:
        tahoma,helvetica,verdana,lucida console,utopia; FONT-SIZE: 10pt; HEIGHT: 25px }
  TD.border-HILIGHT {   BACKGROUND-COLOR: #ffffcc; COLOR: black; FONT-FAMILY:
                verdana,arial,helvetica,lucida console,utopia; FONT-SIZE: 10pt;
HEIGHT: 25px }
-->
-->
</style>
</HEAD>
<BODY bgcolor=\"#ffffff\">
<form>
<br>
<table bgcolor=\"#ffffff\" border=0 cellpadding=0 cellspacing=0 width=90%\"><tbody>
<tr><td>
        <table border=0 cellpadding=2 cellspacing=1 width=\"90%\" align=\"center\"><tbody>
                <tr><td class=title align=center>Secure List Management</td></tr>
                </tbody></table>
</td></tr>\n";
             print $str;
             if($params[email])
                    fwrite($params[fh], $str);
             if(isset($params[subtitle])) {
                    $str = "
<tr><td>
<table border=0 cellpadding=2 cellspacing=1 width=100% align=center><tbody>
<tr><td font class=subtitle>$params[subtitle]</td></tr>
</tbody></table></td></tr>";
             print $str;
             if($params[email])
                    fwrite($params[fh], $str);
             }
             if(isset($error)) {
                    $str = "
<tr><td>
<table border=0 cellpadding=2 cellspacing=1 width=100% align=center><tbody>
<tr><td font class=error>$error</td></tr>
</tbody></table></td></tr>";
             print $str;
             if($params[email])
                    fwrite($params[fh], $str);
             }
             print "<tr><td><hr></td></tr>\n";
             if($params[email])
                    fwrite($params[fh], "<tr><td><hr></td></tr>\n");
             } /* end of disp_header */

       function disp_footer() {
             print "</form></body></html>";
       }
?>
```

Appendix B: PERL Modules

Module Name	Module Description	Also Provides		Dependencies	Version
Mail::GnuPG	To process email with GPG			GnuPG::Interface	Mail-GnuPG-0.07
GnuPG::Interface	Perl Interface to GnuPG	GnuPG::Fingerprint	GnuPG Fingerprint Objects	Class:MethodMaker	GnuPG-Interface-0.33
		GnuPG::Handles	GnuPG handles bundle		
		GnuPG::Interface	Perl interface to GnuPG		
		GnuPG::Key	GnuPG Key Object		
		GnuPG::Options	GnuPG options embodiment		
		GnuPG::PrimaryKey	GnuPG Primary Key Objects		
		GnuPG::PublicKey	GnuPG Public Key Objects		
		GnuPG::SecretKey	GnuPG Secret Key Objects		
		GnuPG::Signature	GnuPG Key Signature Objects		
		GnuPG::SubKey	GnuPG Sub Key objects		
		GnuPG::UserId	GnuPG User ID Objects		
Class:MethodMaker	Generic methods for object oriented PERL	Class::MethodMaker	Create generic methods for OO Perl		Class-MethodMaker-2.00
		Class::MethodMaker::Constants			
		Class::MethodMaker::Engine	The parameter passing, method installation & non-data-structure methods of Class::MethodMaker.		
		Class::MethodMaker::OptExt	Constants for C::MM's option extension mechanism		
		Class::MethodMaker::V1Compat	V1 compatibility code for C::MM		
MIME::Entity	Class for parsed-and-decoded MIME message	MIME::Body	the body of a MIME message	Mail::Field Mail::Header Mail::Internet Unicode::Map Unicode::String IO::Stringy	MIME-tools-6.200_02
MIME::Parser	Class for parsing MIME messages	MIME::Decoder	an object for decoding the body part of a MIME stream		MIME-tools-6.200_02
		MIME::Decoder::Base64	encode/decode a "base64" stream		
		MIME::Decoder::Binary	perform no encoding/decoding		
		MIME::Decoder::Gzip64	decode a "base64" gzip stream		
		MIME::Decoder::NBit	encode/decode a "7bit" or "8bit" stream		
		MIME::Decoder::QuotedPrint	encode/decode a "quoted-printable" stream		

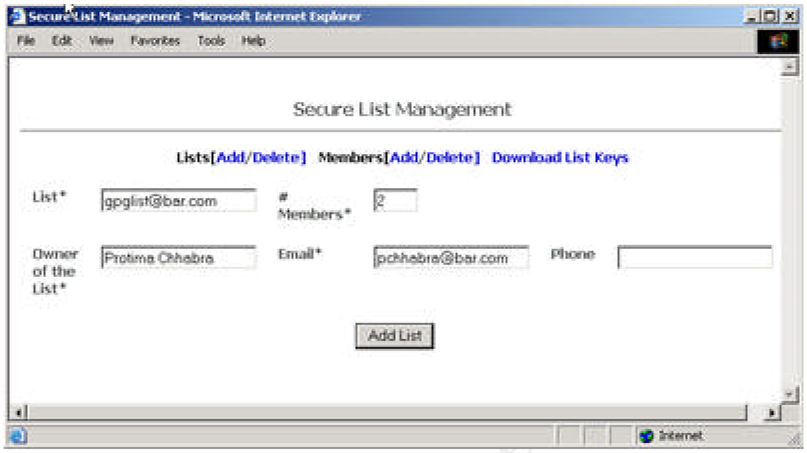
		MIME::Entity	class for parsed-and-decoded MIME message	
		MIME::Field::ConTraEnc	a "Content-transfer-encoding" field	
		MIME::Field::ContDisp	a "Content-disposition" field	
		MIME::Field::ContType	a "Content-type" field	
		MIME::Field::ParamVal	subclass of Mail::Field, for structured MIME fields	
		MIME::Head	MIME message header (a subclass of Mail::Header)	
		MIME::Parser	experimental class for parsing MIME streams	
		MIME::Parser::Filer	manage file-output of the parser	
		MIME::Parser::Reader	a line-oriented reader for a MIME::Parser	
		MIME::Parser::Results	results of the last entity parsed	
		MIME::Tools		
		MIME::WordDecoder	decode RFC-1522 encoded words to a local representation	
		MIME::Words	deal with RFC-1522 encoded words	
Mail::Field	Base class for manipulation of mail header fields	Mail::Address	Parse mail addresses	MailTools-1.60
		Mail::Cap	Parse mailcap files	
Mail::Header	For manipulationof mail RFC822 compliant headers	Mail::Field	Base class for manipulation of mail header fields	
Mail::Internet	For manipulation of Internet format (RFC 822) mail messages	Mail::Field::AddrList	object representation of e-mail address lists	
		Mail::Field::Date		
		Mail::Filter	Filter mail through multiple subroutines	
		Mail::Header	manipulate mail RFC822 compliant headers	
		Mail::Internet	manipulate Internet format (RFC 822) mail messages	
		Mail::Mailer	Simple interface to electronic mailing mechanisms	
		Mail::Mailer::qmail		
		Mail::Mailer::rfc822		
		Mail::Mailer::sendmail		
		Mail::Mailer::smtp		
		Mail::Send	Simple electronic mail interface	
		Mail::Util	mail utility functions	
Unicode::Map	For mapping character sets from and to utf16 unicode			Unicode-Map-0.112

Unicode::String	String of Unicode characters (UCS2/UTF16)	Unicode::CharName	Look up Unicode character names	Unicode-String-2.07
		Unicode::String	String of Unicode characters (UCS2/UTF16)	
IO::Stringy	I/O on in-core objects like strings and arrays	IO::AtomicFile	write a file which is updated atomically	IO-stringy-2.109
		IO::Clever		
		IO::InnerFile	define a file inside another file	
		IO::Lines	IO:: interface for reading/writing an array of lines	
		IO::Scalar	IO:: interface for reading/writing a scalar	
		IO::ScalarArray	IO:: interface for reading/writing an array of scalars	
		IO::Stringy		
		IO::Wrap	wrap raw filehandles in IO::Handle interface	
		IO::WrapTie	wrap tieable objects in IO::Handle interface	

Appendix C: Screenshots

C.1 Add List

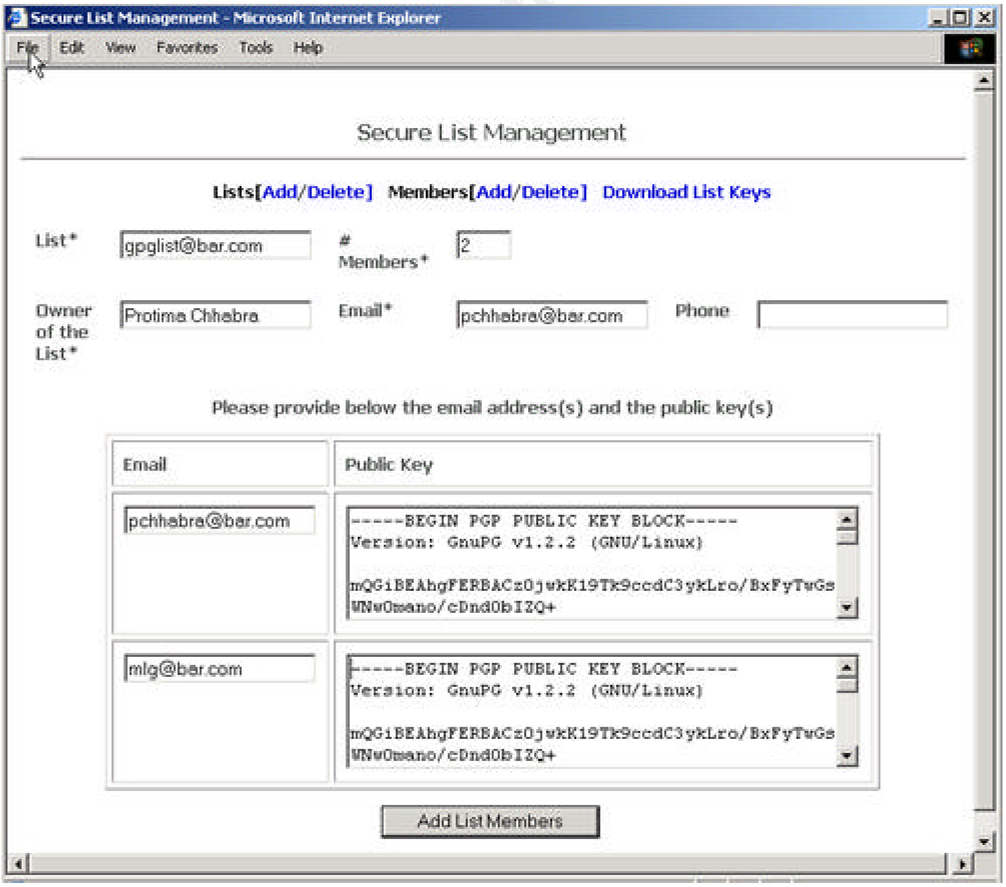
C.1.1 Get Basic List Information


Secure List Management - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Lists[Add/Delete] Members[Add/Delete] Download List Keys
Add List
Internet

Secure List Management

List*	gpglist@bar.com	# Members*	2		
Owner of the List*	Protima Chhabra	Email*	pchhabra@bar.com	Phone	

C.1.2 Get List Member Information


Secure List Management - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Lists[Add/Delete] Members[Add/Delete] Download List Keys
Add List Members

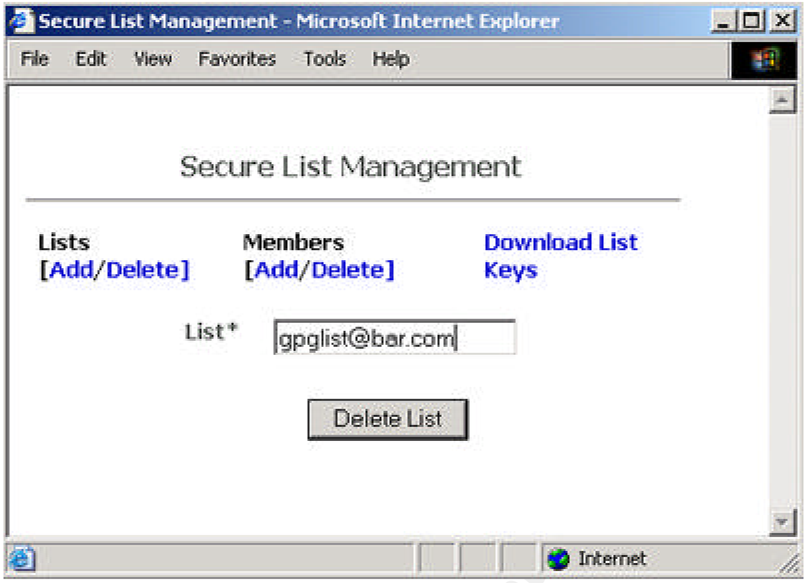
Secure List Management

List*	gpglist@bar.com	# Members*	2		
Owner of the List*	Protima Chhabra	Email*	pchhabra@bar.com	Phone	

Please provide below the email address(s) and the public key(s)

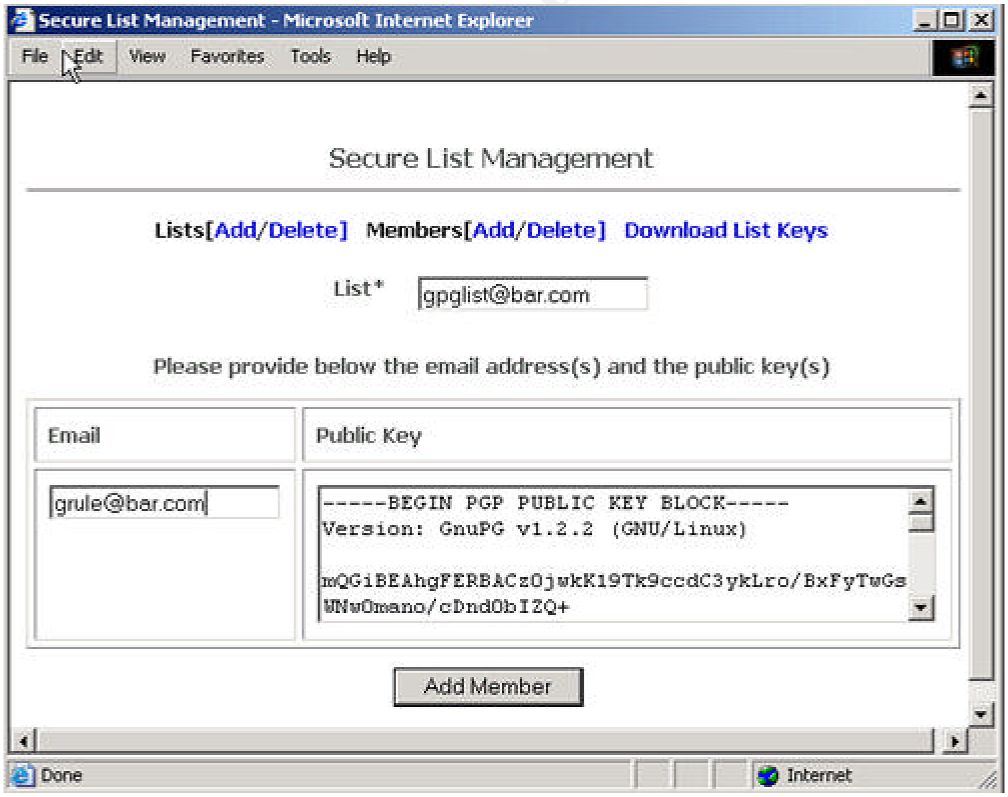
Email	Public Key
pchhabra@bar.com	-----BEGIN PGP PUBLIC KEY BLOCK----- Version: GnuPG v1.2.2 (GNU/Linux) mQGiBEAhgFERBACz0jwkK19Tk9ccdC3ykLro/BxFyTwGs WNw0mano/cDnd0bIZQ+
mlg@bar.com	-----BEGIN PGP PUBLIC KEY BLOCK----- Version: GnuPG v1.2.2 (GNU/Linux) mQGiBEAhgFERBACz0jwkK19Tk9ccdC3ykLro/BxFyTwGs WNw0mano/cDnd0bIZQ+

C.2 Delete List



List* gpglist@bar.com

C.3 Add Member


Secure List Management - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Secure List Management
Lists[Add/Delete] Members[Add/Delete] Download List Keys
Add Member
Done
Internet

List* gpglist@bar.com

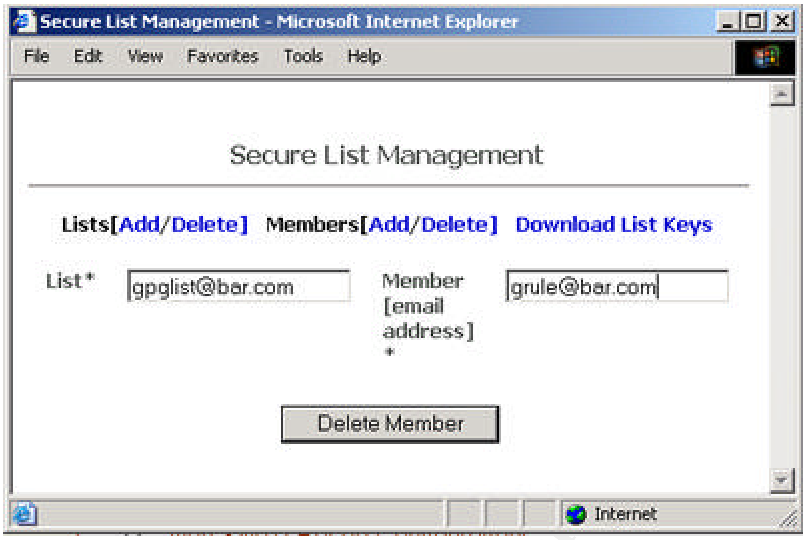
Please provide below the email address(s) and the public key(s)

Email	Public Key
grule@bar.com	see key below

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.2.2 (GNU/Linux)

mQGiBEAhgFERBACz0jwkK19Tk9ccdC3ykLro/BxFyTwGs
WNwOmano/cDndObIZQ+
```

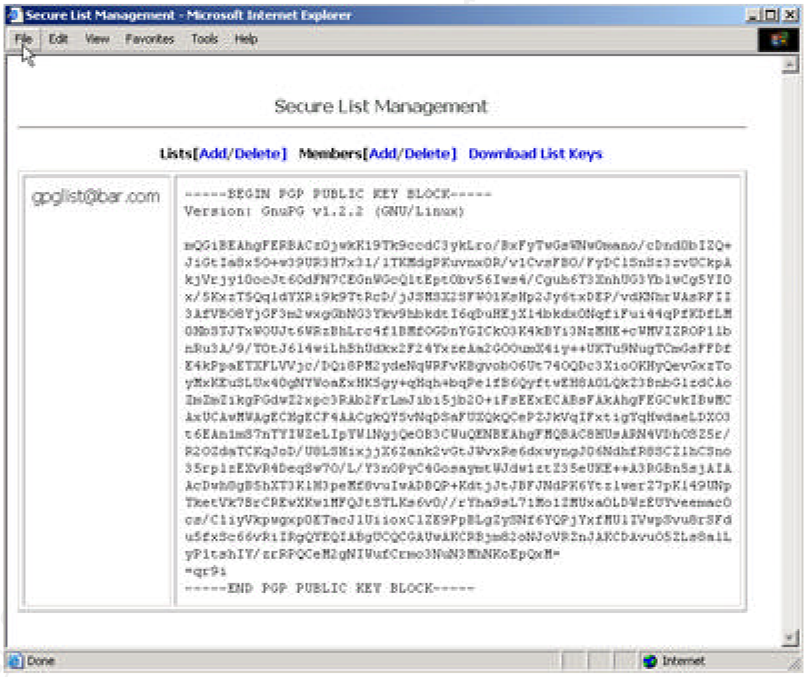
C.4 Delete Member


Secure List Management - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Secure List Management
Lists[Add/Delete] Members[Add/Delete] Download List Keys
Delete Member
Internet

List* gpglist@bar.com

Member [email address]* grule@bar.com

C.3 Display Public Keys of Lists for Download


Secure List Management - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Secure List Management
Lists[Add/Delete] Members[Add/Delete] Download List Keys
Done
Internet

gpglist@bar.com

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.2.2 (GNU/Linux)
```

```
-----END PGP PUBLIC KEY BLOCK-----
```