Secure Email Application

Richy Delaney 08479950

April 5, 2011

Project Approach

The idea of the project was to create a signed and enveloped message and some means of sending this email using open source encryption toolkit. My first task was to do research into the different areas of the project. I decided to program it in python as it seemed to have the most amount of libraries and infrastructure in which to implement this in. I also identified that the project's core would use openssl for almost all the encryption/decryption/certificate work.

My program is a basic python wrapper around a series of shell commands that do the encryption/decryption and some python to handle user input and the actual flow of the program. I could have resorted to doing it all in a bash command, but I was more familiar with python programming than bash scripting. I'll know go through how each part of the system is designed. The following sections is how I divided the project up:

1. User Input
2. Signed Digest of Message
3. Encryption of Body
4. Sending the message using SMTP
5. Receiving the message using IMAP
6. Decrypting and Verifying Integrity of message.

User Input

The first thing the program has to do when it is sending mode, is receive the full extent of the mail from the user, this includes the to, from, subject and body fields. I use a simple email address regular expression to validate emails and store the rest in variables. I also write the subject and email out to temporary files, this seems like overhead but I found this easier for encrypting later on as otherwise I had to echo the variables and when the data input wasn't standard this often broke when I went to echo it, so I decided it was safer to write to files.

After this section, if the user hasn't used the system before, a private and public key pair is generated for them using the openssl libraries rsagen command.

Digest of the Message

The next step is to get a message digest of the body, I use openssl to do a sha1 hash of the body, and then signed it using the senders public key. This proves that the document is definitely sent from the correct person as although someone can decrypt it they cannot change it and re-encrypt it.

Encryption of Body

To encrypt the body, I used AES symmetric encryption, I ask the user for a passphrase which is used as the key for this encryption, this passphrase is then encrypted using the public key of the recipient, and attached to the mail. I used openssl enc command to use aes 256 bit encryption, ask the user for a passphrase, this passphrase is then encrypted using rsa with the public key of the recipient and attached to the mail.

Sending the message using SMTP

For the sending of the message, I used pythons smtplib module to connect to gmail's smtp service, Although this limited my senders to a gmail login, I felt this was the easiest way to show the program working. the smtp.gmail.com could easily be swapped out for any service. The smtplib object then allows you to send the mail. The mail itself is a MIMEMultipart message which you can use to build up a mail and attach items. The message consists of a to, from and a encrypted subject.

The body of the email is attached as MIMEText while the other two attachments are octet stream attachments. The other two attachments are the encrypted passphrase and the signed digest.

The multipart object is then represented as a string and sent to the smtp server.

Receiving the message via IMAP

In receiving mode, Once I establish the users authentication details, I then use IMAP to receive the latest message,this is all possible through the imaplib module available in python. Once the latest message has been fetched. I use a foreach loop through the multipart message and determine which part is which, I assign each to its own variable, and end up with subject, body, passphrase and digest, all in their encrypted format.

Decrypting and Verifying

I first decrypt the passphrase, the decrypted passphrase is then used to decrypt the message body and subject. The final step of the decryption process is the verification that the main body has not been meddled with. We have to decrypt the hash using the senders public key, we then get a hash of the decrypted body and check it against this hash and if they are the same then the body has not been tampered with.

If all this has happened successfully, we simply print out the decrypted message to the user.

Conclusion

I broke the project up into a number of different area's, and through this was able to achieve the correct flow of the encryption and decryption process.I used a CA on my webspace to allow the application to download certificates and check that the cert is correct and then extract public key from that. All in all, I implemented all that was asked and created a secure email application which can easily send and receive messages in a secure format.