Alex Hortin
PA2

1) Cyptographic Checksum of your Source Code (5 pts)
Compute an md5 or SHA1 checksum of a tar file (or similar archive) of your source code and executable image. Include the name of this file, it's size, and it's checksum in your write-up.

To compute the MD5 sum of my Work, I used the 'md5sum' command line utility in ubuntu on my source code file.

Name	**Size**	**MD5 Checksum**
cpts425p2Hortin.tar.gz	**9219 bytes**	**60a7e1798779f8a5743b8ca513c1c404**

2) Abbreviated Software Design Document (10 pts)
Briefly describe the data structures and algorithms used to implement the three required tools. (1-2 page limit)

When preparing for an algorithm test that covered many topics, including RSA, I decided to implement the encryption algorithm myself to better understand it. While doing this I was able to test my algorithm against the RSA component of a widely accepted open source implementation to make sure I was encrypting/decrypting correctly. This software tool was called openssl and I decided to do this assignment using it because it let me access very powerful encryption methods using a few lines of shell code.

Since encryption/decryption and key generation needed to be automated for each of the programs, I needed a simple and powerful language to generate the proper shell commands to generate or modify the proper files with as little interaction from the user as possible. I was able to achieve this using Python. I chose Python for this assignment mainly because I am very comfortable with the socket class and using it to pass messages. Command line processing, while fairly trivial can also be handled very easily in Python, which is always good because it saves time that could be better used for implementing the more difficult parts of the assignment.

The only real data structures I used within python were files, sockets, and strings. Files I used for the unencrypted messages, encrypted messages, public and private keys, as well as the final decrypted messages. Python allowed me to quickly read and write the files that I would be using with the openssl tools and modify them as needed. Closely tied to the modification of these files was the string processing I took advantage of in Python to make file I/O very easy. When I needed to modify a message or change a file it was very easy. I also used this string processing to create the shell code that I sent to the terminal using the os module. This allowed me to create large paths that were easily changed in the very beginning of the file if need be. For example you will notice that despite the directory structure needed by all of the shell commands, I was able to come up with a way where you would need to only change the user(sender) and rec(person your sending to) to differentiate between alice and bob. Other than that the two programs are

identical and can each send and receive. The string processing handles the rest for the system, so it is not hard to change the code if you need to. Finally I used sockets to pass the raw information between the users.

The really interesting part of the code was within the OS calls. Below are basic versions of the openssl commands that I used:

openssl expression used to generate the private rsa key

```
$openssl genrsa -out private.pem 1024
```

openssl expression used to generate the public rsa key using a private key as input

```
$openssl rsa -in private.pem -pubout -out public.pem
```

openssl expression used to encrypt the message using the recipients (bob) public key

```
$openssl rsautl -in plain_msg -out enc_msg -encrypt -pubin -inkey
        bob_public.pem")
```

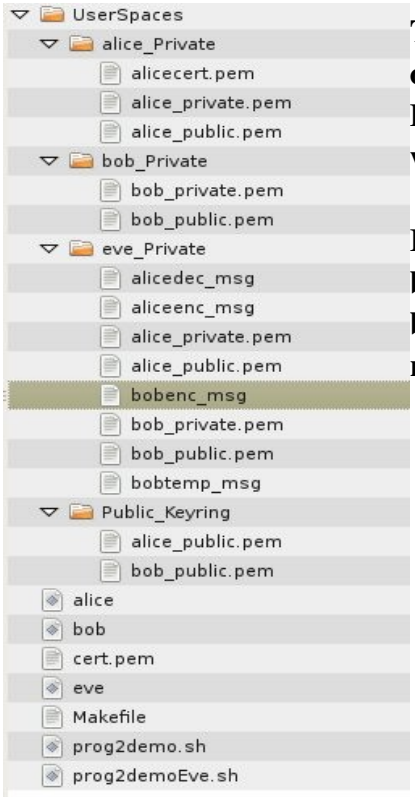
openssl expression used to decrypt the message using alice's private key

```
$openssl rsautl -in enc_msg -out dec_msg -decrypt -inkey alice_private.pem")
```

openssl expression to generate new cert

```
$openssl req -new -x509 -days 365 -out cert.pem -keyout cert.pem
```

All of these were executed using files that were spread out in a hierarchy like this



This represents the files that all of the openssl programs operate on using arguments given by the python programs that call them. Everything here except the folders is generated automatically within the program.

If the keys are altered by eve they do need to be regenerated by bob for them to work again with the test cases. This can be done by simply calling: ./bob -g or ./alice -g....or just a simple $make resets everything.

3) Software Test Document (15 pts)
Describe your software test plan and methodology that you used to verify that you implemented the three required software tools correctly. List any know deficiencies. (1-2 page limit) Include an appendix that lists the various private / public keys used for one run of your monkey-in-the-middle attack. (Page limit will vary based on font size, etc.)

I tested this program modularly starting with what I thought were the most basic components. After I was satisfied I was able to implement that next area and test that. I repeated this process until I was satisfied that everything worked.

When starting this assignment I figured the best thing to test was the thing that was easiest to implement...and I decided that my openssl commands would be the easiest to test. I needed to test generating a private key, and using that to generate the resulting public key. I generated two sets of these keys (one for theoretical alice and theoretical bob). After I had these I used one of the private keys to generate an x509 certificate and verified that it was correct. The remaining openssl commands I needed to test were encryption with a public key, than decryption with a private key. Once I was able to decrypt an encrypted file, I was satisified I could start working on the program.

Since I implemented bob and alice as only slight (2 line) variations from each other, I started by making sure that a copy of the program with a message passed in would act like a sender, while one started without a message would act as an openserver looping over and over again to receive and decrypt messages. I then generated the paths I would need for each program in strings that I could easily pass into the shell commands that I had already tested and pasted those paths into the proper area in the command strings that I would pass to openssl. This part proved to be the most difficult, because it was fairly challenging to get the proper paths and temporary files so they would match up with the right keys. Once I had this worked out I was able to test the message passing and resulting decryption to verify that bob and alice were working together correctly.

Once these were working it was only a few slight modifications to the shell code and restructing to create eve. The first thing I did was program that code that generated two private keys for alice and bob. Than I copied the old public keys into eve's private directory, replacing them with the new public keys that were a resultant of eve's private keys for each alice and bob. The only problem I run into is running the program too fast and running into sockets not being properally shut down. I believe that this is a socket implementation problem since I properly close and destroy all sockets, but I still get the problem once in 10 runs or so.

An example of the keys used in the monkey in the middle are:

Bob's Public compromised key (used for Alice encrypting the data)

-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDB6W3uQLY7C0SiAaqqzwIGKlQF
Y4lzYd1YzAdPmBqZA0W1CPErOKJf0NKjx5TXH+k5d5dQ2oR3JXDrzOuvYmRkQzk8
We0yQHWt23vW++UEyQEslZGjPKOF1pQYtCjA8qST/RVido1Fc1CaJUdE8hokEJVx
ZRNg4ZSBGp9H0SNPRQIDAQAB
-----END PUBLIC KEY-----

Eve's Copy of Bob's 'new private' Key (used for decrypting the data)

-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQDB6W3uQLY7C0SiAaqqzwIGKlQFY4lzYd1YzAdPmBqZA0W1CPEr
OKJf0NKjx5TXH+k5d5dQ2oR3JXDrzOuvYmRkQzk8We0yQHWt23vW++UEyQEslZGj
PKOF1pQYtCjA8qST/RVido1Fc1CaJUdE8hokEJVxZRNg4ZSBGp9H0SNPRQIDAQAB
AoGAebIwBbijJUcbGbmGDJDCKnBiLsebOciPi+zwIanJ94GwO7+S65ctgIijyZm1
9ZTXX/OICXw6ajQbPEFtILLVYWdl1Sl+KsnnDtMIREi6wkGQ6we/E9O7J37hxYdk
oZ5IHCzvZ8vOxLq3mecd9ZiwRedokzW0UCyV20QYpLTElIECQQDmY43pIrHzAdkb
7mp3LO9wSByvJg2coO8QUCh9XWqxmcHR4kOLhIXPfFHJZo2bVfNh2Wajm+NdVpWY
yf447fl1AkEA13fOFqT0i28FftB2Yj4+0oj1Qap2ssauEKlD5Z4hd6rRCrl++W7y
MiLzx8sTlunXuo62ukyrNXTH2v0azhB0kQJBALTgwmCgIa+U6tx4AVRRjibMJuEH
8jXfT/GxzSgnuIV5HudmSft2bwvoIdoKTvD+uRdAdMnMsOPt/4M2SwZtJ6UCQQCd
zRs9EFAc8hFHP+sDpmyiCjX8gpht8/41K5ITT41ZM9vnC9VHXo64U0rQrxTIuLgK
5ExFArN88esQob+I3RpRAkA1TNc9wRsjukMiB51nsf3OI4iU4ztOj7ZXnGyzY6Ya
4aRRm0BXS7FWEw1nhq0FuPVeaLYLXg4MESsLAmh5FqpX
-----END RSA PRIVATE KEY-----

Eve's Copy of Bob's original public key (used for encrypting the new message)

-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCv0xa6F7lSU4svXX+sZUQSQEZ0
5muScBuaBsoiZaGJt8Z7SfUpn6rQbfYvOk79OyvIdP/CfwaKs3qpm5T8bghOV4uF
zYiJwOJci7bjf2KUUaXAtayZV2P90HSwiFrCZH+eZqhmF1aPxhqLylX6XNp/yOIJ
Vmt3oN1wSrM7xzKAYwIDAQAB
-----END PUBLIC KEY-----

Bob's Private Original key (used to decrypt eve's message)

-----BEGIN RSA PRIVATE KEY-----
MIICWwIBAAKBgQCv0xa6F7lSU4svXX+sZUQSQEZ05muScBuaBsoiZaGJt8Z7SfUp
n6rQbfYvOk79OyvIdP/CfwaKs3qpm5T8bghOV4uFzYiJwOJci7bjf2KUUaXAtayZ
V2P90HSwiFrCZH+eZqhmF1aPxhqLylX6XNp/yOIJVmt3oN1wSrM7xzKAYwIDAQAB
AoGAOQuEsSXnU6WVX+228aj9V+qQsXbc6esDmYqgZF6S9Yqxk+if6TxLgEpc+F+z
pwdeojALlv/9kFzA7Gx0/x14BeFIKBhCs3HLDdC2b6BZje9nOiCKaNyes7MQPkf6
MqhJCkRNL4Fu50wdnAb64FdcBTuBZxbHe9IIySj6fGGZONECQQDU5IJ5VP5w4oM4
/MTGLjbcoeiWNyhQS5Qk9ZByIxj6Usw6NcWARveJlWlljUYzS4IFpgcRbdu120py
X7U9yrA7AkEA020fIrVoal9UEhU4ZQhqsHDktQiPzQJUAFQ2M3fWdiJW/czwemCg
A4mNGz8ojytRzi7ReG3rjekBsVJp3oTV+QJAeiae4kUtNNIS3sjRkrCDk/ZmQIJn
ZCLpXodzmLmgpEiLm76skpylfVC4R6dWpynk8Z7AoXR6Qw8O2S8nC8xXbwJADe/D
faY3m8ts30RG02IapkbKzKQj1n+wydTDPeFIYcEhutkngOmdop4Ojx06rzVLzyuG
40lbnZ8eXXHxSJm5AQJAGLsuMXx9F5Lth1Arm+S42+HnYfqPrUbszuRZyRpjnZsm
6hKp4YNsY7UBVE9t9P02apFr1u+6f+jTjmrynHuxbA==
-----END RSA PRIVATE KEY-----

4) Lessons Learned / Project Evaluation (10 pts)
Discuss the lessons learned from this project. Are there improvements that should be made in a future release? Other possible questions to address: Was this assignment as easy as you anticipated? Did this assignment adequately introduce you to the complexities of key management? Etc.

I would say the most important lesson that I learned during this project is how expandible programs can be just using a few commands passed to the terminal. If I would have reimplemented everything that openssl did I would have spent a great deal of time trying to get it to work and debugging it. Using openssl let me very rapidly develop a working prototype that did everything the assignment required.

I think it would be very interesting to try to implement future versions of this program that take other things into consideration. Most of all it would be nice to receive alice's message and than be able to write while the program was running what you wanted to send Bob. This would allow for real time interaction. I also think that it would be fun to implement this as a 2 way messaging client that actually continues to run rather than stopping after one message send while bob sits there and waits. It also might be fun to try and spoof the more advanced certificates.

The assignment was not as quick as I thought it might be, but I thought it was worthwhile because of the work I got to do with openssl. Everything else was a really a review, but I learned a lot about the tools available on any linux system to deal with very heavy encryption. I really enjoyed the book Cryptonomicon, by Neal Stephenson, as I think I mentioned in my last assignment, but this time I dealt with making an interface for programs that they had mentioned in that novel. Reading through the book in the last homework about certificate management made it very obvious that key management is not a trivial thing.

Since I plan on working on mobile devices later and dealing with things such as certificate management I enjoyed this assignment because it introduced me to the complexities of openssl and using that program to build applications. Mobile devices are already in use in many countries as a fast pay option, and it would be interesting to see the technology that keeps those devices safe from abuse.

5) Time Logs (Optional, no points)
List the time spent during software design, implementation, testing. Also list the time spent writing up your project.

Approx 6 hours on the assignment
Approx 3 hours on the written section