Sarah Lundell
Thomas Mahre
Josh Hamel

1) How did you hash the passwords in the file? Include information such as the hashing algorithm, the library you used, how you appended the password with the salt before hashing, and other information we've discussed related to password storage. Please provide brief explanations as to why you chose to use what you did. Convince the reader that your password storage algorithm is secure.

To hash the password I first made a random salt with the OS library using urandom. I picked 16 bits for the salt because we did not need a huge amount of salts so 16 bits seemed like enough.

```
# TODO: Create a salt and hash the password
#32 bit salt
salt = os.urandom(16)
# PKCS#5 password-based key derivation. Takes name of hash, password, salt, and number of interation
hashed_password = hashlib.pbkdf2_hmac('sha256', password.encode('utf-8'), salt, 100000)
```

To join the salt and the password that was the input, I used the hashlib library with the pdkdf2_hmac function. Pdkdf2_hmac takes the name of the hash function you want to use (sha256 is what I picked), password to be hashed, salt that we came up with before, and number of iteration on the hash. This function encrypts the values and stores them in the variable hashed_password which we later store in the file along with the unhashed value of the salt and the username. In terms of picking the hash function I choose sha256 because after reading some documentation it seems like that is what most people tend to use with hashlib. Also, it is good to note that there is a warning that comes with using hashlib that says that some algorithms within the library have hash collision issues, but from what I read the function that I picked do not seem to be affected by this.

2) Describe your approach to public/private key use. How did you generate your public/private key pair? What library did you use to encrypt messages? In which folder did you store your keys? Please provide any details you think would be relevant.

The Public\Private Key Pair is based around a math principle where inorder to decrypt a public key, a private key must be used. This is RSA encryption and thus asymmetric. The private key was used in our handshake and encrypted with the RSA cipher to create our session key. This RSA cipher was generated with our given public key that was mentioned earlier and thus to decrypt the recipient had to have the private key. Thus our threeway connection is verified. These public keys that were given to us were stored in the same folder as the user as the user would need access to them. If they were stored in a different folder we would have security leaks.

3) How did you manage symmetric encryption? What encryption mode did you use, and

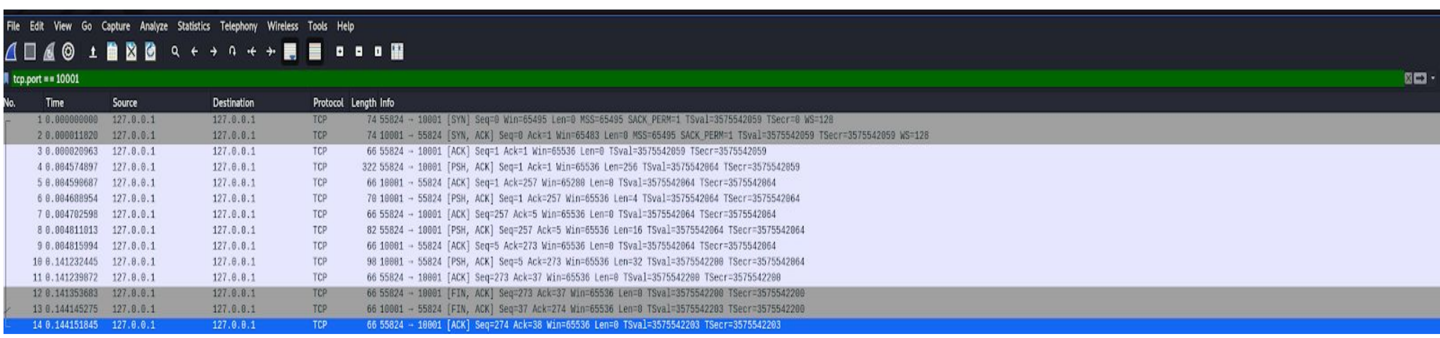
why did you think that was best? What tradeoffs did you make for that encryption mode?

For symmetric encryption we used AES_ECB. To send the key safely we initially encrypted the key with the client's public key(RSA), then the ECB key was decoded by the server's private key. We implemented ECB due too it being the simplest to implement. However the main tradeoff was ECB being seen as one of the weaker symmetric encryptions. One of the major reasons is it takes blocks of data and encrypts all the blocks using the same algorithm. Someone who is sniffing would be able to see patterns if blocks of data were the same.

```
def decrypt_key(session_key):
    private_key = RSA.importKey(open('client_cert').read())
    private_cipher=PKCS1_OAEP.new(private_key)
    message=private_cipher.decrypt(session_key)
    return message
```

4) Provide a brief explanation of why your program would be secure from eavesdroppers if you were to run it on a publicly visible network, from start to finish. Write your program like you're protecting yourself from Comcast or the NSA!

When the conversation starts, the first thing that happens is the client creates the secret AES key and encrypts it using its public key. The only thing that can decrypt it is the server's private key. An eavesdropper could not decrypt it. Once the key is passed the rest of the conversation is encrypted by AES(symmetric encryption). Unless the eavesdropper had the AES key, he/she wouldn't be able to decrypt the rest of the conversation between the



File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

tcp.port == 10001

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	74	55824 → 10001 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=3575542059 TSecr=0 WS=128
2	0.000011820	127.0.0.1	127.0.0.1	TCP	74	10001 → 55824 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=3575542059 TSecr=3575542059 WS=128
3	0.000020963	127.0.0.1	127.0.0.1	TCP	66	55824 → 10001 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3575542059 TSecr=3575542059
4	0.004574897	127.0.0.1	127.0.0.1	TCP	322	55824 → 10001 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=256 TSval=3575542064 TSecr=3575542059
5	0.004590687	127.0.0.1	127.0.0.1	TCP	66	10001 → 55824 [ACK] Seq=1 Ack=257 Win=65280 Len=0 TSval=3575542064 TSecr=3575542064
6	0.004688954	127.0.0.1	127.0.0.1	TCP	70	10001 → 55824 [PSH, ACK] Seq=1 Ack=257 Win=65536 Len=4 TSval=3575542064 TSecr=3575542064
7	0.004702598	127.0.0.1	127.0.0.1	TCP	66	55824 → 10001 [ACK] Seq=257 Ack=5 Win=65536 Len=0 TSval=3575542064 TSecr=3575542064
8	0.004811013	127.0.0.1	127.0.0.1	TCP	82	55824 → 10001 [PSH, ACK] Seq=257 Ack=5 Win=65536 Len=16 TSval=3575542064 TSecr=3575542064
9	0.004815994	127.0.0.1	127.0.0.1	TCP	66	10001 → 55824 [ACK] Seq=5 Ack=273 Win=65536 Len=0 TSval=3575542064 TSecr=3575542064
10	0.141232445	127.0.0.1	127.0.0.1	TCP	98	10001 → 55824 [PSH, ACK] Seq=5 Ack=273 Win=65536 Len=32 TSval=3575542200 TSecr=3575542064
11	0.141239872	127.0.0.1	127.0.0.1	TCP	66	55824 → 10001 [ACK] Seq=273 Ack=37 Win=65536 Len=0 TSval=3575542200 TSecr=3575542200
12	0.141353683	127.0.0.1	127.0.0.1	TCP	66	55824 → 10001 [FIN, ACK] Seq=273 Ack=37 Win=65536 Len=0 TSval=3575542200 TSecr=3575542200
13	0.144145275	127.0.0.1	127.0.0.1	TCP	66	10001 → 55824 [FIN, ACK] Seq=37 Ack=274 Win=65536 Len=0 TSval=3575542203 TSecr=3575542200
14	0.144151845	127.0.0.1	127.0.0.1	TCP	66	55824 → 10001 [ACK] Seq=274 Ack=38 Win=65536 Len=0 TSval=3575542203 TSecr=3575542203

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ··········  ······E·
0010  01 34 14 4f 40 00 40 06  27 73 7f 00 00 01 7f 00   ·4·O@·@·  's······
0020  00 01 da 10 27 11 66 5c  d3 43 8b e4 79 e0 80 18   ····'·f\  ·C··y···
0030  02 00 ff 28 00 00 01 01  08 0a d5 1e 71 30 d5 1e   ···(·····  ····q0··
0040  71 2b 7c 74 cd 7b 43 0c  1c fb 82 d1 90 3d 0a 14   q+|t·{C·  ·····=··
0050  a6 ab f0 a3 b5 b6 45 66  b3 2c 4a df 90 bf c6 bd   ······Ef  ·,J·····
0060  a9 b7 44 6b be 10 28 78  b4 f1 02 e3 a7 48 03 a8   ··Dk··(x  ·····H··
0070  87 9d a8 aa 00 6f c0 f0  23 fe 63 47 da fa 13 6a   ·····o··  #·cG···j
0080  3a 37 6f 9e 36 fd 5d 69  51 71 5a 04 9a a2 cf 2e   :7o·6·]i  QqZ····.
0090  63 2d b0 dd 1f 2c 68 89  3f d1 cf 1c 0d 79 e7 fa   c-···,h·  ?····y··
00a0  f2 a2 b6 ae c9 15 73 59  0c 8d d7 d3 65 0a 81 f5   ······sY  ····e···
00b0  2b fd cc 39 7d 52 de c1  59 1c 6e 34 b8 3b 7f 46   +··9}R··  Y·n4·;·F
00c0  9b 6a fe b8 f7 8c f8 57  ff 60 3f 4d 68 0c e7 dc   ·j·····W  ·`?Mh···
00d0  7d c2 13 13 eb a8 28 8a  bc 38 cb e7 a4 ff 4a 3d   }·····(·  ·8····J=
00e0  b4 9c c4 ad 61 f5 27 06  44 5b 64 af d4 69 f0 d7   ····a·'·  D[d··i··
00f0  9a e9 11 7a 75 75 31 8f  66 31 af db 50 57 b0 d3   ···zuu1·  f1··PW··
0100  9d 0b 5c 8a 32 1d 6a a5  0f 09 f3 73 a3 de 22 83   ··\·2·j·  ···s··"·
0110  df 01 77 54 9f b8 2b f6  da 5d ed b2 f0 35 11 6f   ··wT··+·  ·]···5·o
0120  f4 c0 f5 eb 20 dc 04 c6  cc 0c 9a bc f0 87 cd 82   ···· ···  ········
0130  ad db 07 a8 9d 4a dd d4  a3 66 26 91 5d 9c 7d 4b   ·····J··  ·f&·]·}K
0140  79 b1                                              y·
```


Photos above shows our encrypted traffic.

5) Is your program secure from replay attacks? If not, why not? Use protocols that we discussed in class to show that it isn't vulnerable. If it is vulnerable, what could you do to prevent it? Could you perform a replay attack against yourself?

We do not prevent replay attacks. We did not implement the code to do this.

It would involve another message being sent between the client and the server where the server generates a random number and the servers sends back that same random number plus one. The issue with this method would be if the attacker could guess the random number they would be able to get around this. The other way to prevent a replay attack would be to add a timestamp with your session. The time stamp makes it so an attacker couldn't re open a session. The only issue with the time solution is that if the two people who are actually meant to be communicating have clocks that are not synced then it won't allow them to communicate either.

6) What else did you learn from the project? Did you have to do some research? Collectively, or individually, give some of the key takeaways you had when doing this.
We had to do a fair amount of research for this project. We had to look up libraries and functions for how to make out salt, encryption, and decryption. We also had a lot of issues with storing the password and salt in the file since the values that we produced were in hex but storing them in the file made us change them into a string. Initially we had a misunderstanding about how reading from a file worked, even though what we were reading was in a byte type format, our code would read it only as a string since we used the parameter “r” rather than ”rb” for reading it in bytes. Changing this hex value back from a string proved to be very difficult because the os Python library stored a hex value with an \x instead of an 0x so standard converters did not work. Below is a screenshot of some of our encrypted passwords with salts added. The storage format is not optimal.

```
b'\xda\xc1}3\x13\noL\x81\xf4\xee\x0b3\xa1\xd6l\x84\x96\xdcf\xb5\xb4\x86\xe2\xcdx\xfe\xb9B\xec\xe3`'
    b'$\xf4\xef\x80\xcc\xd3#\x95\xc0\x1a\x97\xb5:smJ7\x8f@@k&\xaf~\xd2\x9fq\xed]\x9c6\xdb'
```

Also python has a lot of libraries that do a lot of this stuff for us but it is important to read them because the first one you find may not be the best option.
Another problem that occurred was VM problems in downloading python3 inorder to debug the project. This was not unique to the project but rather errors in the kali linux distro and manually installing instead of using the package manager. As kali linux has certain dependencies other distros do not, using the package manager is highly recommended.