CSCD27 Assignment 2

1. **Key-Handling Vulnerabilities with Public-Key Systems**
   a. **Diffie-Hellman (DH) Key-Exchange Protocol with the AES symmetric-key encryption algorithm.**

   Global params: g, p;
   Alice's private key: a & public key: $A = g^a \bmod p$;
   Bob's private key: b & public key: $B = g^b \bmod p$;
   Shared public key: $g^{a*b} \bmod p$.

      i. **against Eve, a passive eavesdropper**

      Yes the confidentiality is protected against Eve. Eve can only observe and get their public keys A, B and global params g and p, but in order to break the shared key $g^{a*b} \bmod p$, Eve needs at least one of a or b which is only known by the two participants, or she will need to solve a discrete log (i.e.: iterate through g^n to find a/b that equals A/B), which is computationally infeasible.

      ii. **against Mallory, an active MITM attacker**

      No, it's not protected against Mallory as the public key-exchange provides no authentication, Mallory can pretend to be either party in the communication by tampering the public keys included in messages sent in between and send her public keys to both party instead.
      i.e.:

      (Assume Mallory has her own key pair as m and $M = g^m \bmod p$)
      Alice sends Bob $A(= g^a \bmod p)$ ->
      Mallory tampers A tp be $M(=g^m \bmod p)$ ->
      Bob receives M thinking it was Alice's public key -> shared $g^{mb} \bmod p$
      Bob sends Alice $B(= g^b \bmod p)$ ->
      Mallory tampers B tp be $M(=g^m \bmod p)$ ->
      Alice receives M thinking it was Bob's public key -> shared $g^{ma} \bmod p$
      Now Mallory has A, B and m; Alice has a, M; Bob has b, M:
      When Alice sends encrypted message using $(g^m)^a \bmod p$, Mallory decrypt by $(g^a)^m \bmod p$, conduct tampering, and re-encrypt using $(g^b)^m \bmod p$, Bob can then decrypt using $(g^m)^b$ without noticing anything; and vice versa.

   b. **RSA public-key cryptography**

      i. **against Eve, a passive eavesdropper**

      Yes it is secure against Eve. To break the encrypted message, Eve will need to find both the public and private keys in the communication, while only public keys are known to her; the difficulty of factoring large numbers also prevent her from knowing the primes used.

ii. **against Mallory, an active MITM attacker**

No it is not secure against Mallory. Similarly as above, Mallory can replace the public keys sent in the messages by her own and pretend to be another identity without being noticed. i.e.:
Bob: send his public key $\{e_b, n\}$ to Alice
Alice: send her public key $\{e_a, n\}$ to Bob
Mallory: intercept both messages and re-send Alice and Bob her public key $\{e_m, n\}$ instead (n can actually be different, just need to keep a reference to each)

c. **meet at the Tempesta Cafe**

i. **against Eve, a passive eavesdropper**

Yes it is secure against Eve due to the same reason mentioned in b)i). Eve cannot obtain the private keys of the two participants(either by eavesdropping or factoring), thus cannot decrypt the message.

ii. **against Mallory, an active MITM attacker**

Yes it is secure against Mallory. MITM attack succeeds because of Mallory's ability to temper the exchanged public keys in the middle and pretend to be either party before Alice and Bob knows each other's public keys. In this case Alice and Bob both received the identified public keys of each other, therefore authentication is guaranteed.

d. **fresh secure random DH private-key with RSA public-keys to encrypt their DH public-key values**

i. **against Eve, a passive eavesdropper**

Yes it is protected against Eve. As stated above, Eve need to solve discrete logs and do factoring on large numbers to get the privates key in order to decrypt, both of which are computationally infeasible. Also, different private keys are used everyday, which provides the communication with perfect forward secrecy, meaning it will not help even if Eve break the keys for one time if would not help with any decryptions in the future.

ii. **against Mallory, an active MITM attacker**

Yes it is secure against Mallory. The public keys exchange are protected by RSA encryption, where the RSA public keys have not been communicated, therefore Mallory can not obtain/tamper the exchanged keys in the middle and pretend to be anyone else as described in b)ii).

iii. **any important improvement in security?**

Yes, it does improve the security. RSA protects the exchanged DH public keys from being tampered by MITM and thus guarantees the authentication. The random DH key values chosen everyday also

provided perfect forward secrecy, as breaking the key for one time does not help on any future attacks.

2. **Public-Key Certificates**
   a. **Generate an RSA Public Key-pair and a CSR**
      commands:

```
$ openssl genrsa -out mykey.pem 2048
$ openssl req -new -key mykey.pem -out mycsr.pem
```

      see mathlab submission q2/mycsr.pem
   b. **CA signing of CSR**
      see mathlab submission q2/mycert.pem
      i. **Certificate Issuer:**
         `C=CA, ST=ON, L=Toronto, O=UTSC, OU=CMS, CN=cms-chorus.utsc.utoronto.ca`
         Subject:
         `C=CA, ST=ON, L=Toronto, O=MintyMinty, OU=Computer Science, CN=mathlab.utsc.utoronto.ca`
      ii. **For self-signed certificate:**
         Myself will be listed as the issuer and subject, which is :
         `C=CA, ST=ON, L=Toronto, O=MintyMinty, OU=Computer Science, CN=mathlab.utsc.utoronto.ca`
   c. **Test Drive your Certificate 35381**
      i. **error:**

```
Your connection is not private.
Attackers might be trying to steal your information
from mathlab.utsc.utoronto.ca (for example,
passwords, messages, or credit cards).
NET::ERR_CERT_AUTHORITY_INVALID
```

         It's because the certificate-authority(CA) we used to issue our certificate is invalid(the "CSCD27 Root CA"), thus the generated certificate is not trusted, and therefore the identity of the website cannot be verified.
      ii. **How to avoid this in real deployment?**
         Request to a actual and trusted root CA(or a CA whose certificate was in turn (transitively) signed by a root) to issue the certificate using their certificate and key with my CSR.

3. **HTTPS Usage and Vulnerabilities**
   a. **HTTPS Practicalities**

Course Website:

i. **encryption bit-level:** 256 bits
ii. **TLS version:** 1.2
iii. **key-exchange mechanism:** ECDHE_RSA
iv. **encryption mechanism:** AES_256_CBC
v. **authentication mechanism:** HMAC-SHA1

Course Website on Qualsys SSL Server Test:

i. **letter grade and why:** A, the course website ranked high among all 4 fields including certificate, protocol support, key exchange and cipher strength.
ii. **supported SSL/TLS version**
TLS 1.2, TLS 1.1, TLS 1.0 but not SSL 2/3
iii. **preferred cipher suite**
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
(ECDH 256 bits (eq. 3072 bits RSA) FS)

www.bankofamerica.com

i. **encryption bit-level:** 128 bits
ii. **TLS version:** 1.2
iii. **key-exchange mechanism:** RSA
iv. **encryption mechanism:** AES_128_CBC
v. **authentication mechanism:** HMAC-SHA1

ON Qualsys SSL Server Test:

i. **letter grade and why:** A-, the server ranked high in the four fields evaluated, but does not support Forward Secrecy with the reference browsers.
ii. **supported SSL/TLS version:**
TLS 1.2, TLS 1.0
iii. **preferred cipher suite**
TLS_RSA_WITH_AES_128_CBC_SHA (0x2f)

b. HTTPS Forward Secrecy

i. **Ephemeral DH key rather than a long-lived RSA key?**
Given random fresh keys every time, an attacker who is able to break the public/private key used for key-agreement in one session gains no advantage in breaking the keys for other sessions
ii. **Security in terms of relative resistance to cracking**

	Time of generation	**security**
DH	real 0m0.034s	easier to crack, but breaking one key does not

	user sys	0m0.019s 0m0.000s	help with any future breakings.
RSA	real user sys	0m0.342s 0m0.267s 0m0.000s	harder to crack, but once broken can lead to the compromise of multiple messages (past and future)

c. HTTPS Usage Vulnerabilities

log file snippet:

```
2015-11-11 12:08:19,607 SECURE POST Data (login.live.com):
loginfmt=mintyminty%40hotmail.com&passwd=cscd24f15&SI=Sign+in&login=mintyminty%4
0hotmail.com&type=11&PPFT=DXvQ4rMoemMH9Pjg6eUU2PI*K5O*Z7vPeIsHLLCituk0tRCLCB2Tj4
GVcfIrZNIjKTsQpdtx*oh%21at9Pbz2*E5W4c4uoSNhMY7rOgSjGMBaTvC0rUsEMif%21DPaWn9VRUeo
db78NuRf3GZ9dQuJ6MiwnN4zWP5udppck2dFx9FevHnXOmQNtyHJeFx3PlkxOgWALmKXlOdQyqg%21F%
2100WCTvc0XqhdZVmmg0S9cal8RdUo&PPSX=Passp&idsbho=1&sso=0&NewUser=1&LoginOptions=
3&i1=0&i2=1&i3=21234&i4=0&i7=0&i12=1&i13=0&i14=266&i15=484&i17=0&i18=__Login_Str
ings%7C1%2C__Login_Core%7C1%2C
```

i. primary "use-cases" that SSLstrip relies on?

User visits a secure Web site without the bother of typing in a complete URL and uses some subset of the domain name instead, sends out an HTTP request to the server, then the network attacker interpose themselves on the connection by sending the HTTP request from themselves instead. Once the server responds by setting up a HTTPS connection, the attacker then modify the HTTPS response by downgrading it to a HTTP response and send it back to the client. The client sees the correct content without noticing the unsecure HTTP connection. Therefore, the attacker will be able to see all requests & responses happened in between the server and client. (MITM attack)

ii. How can browser-users protect themselves against SSL-stripping attacks?

Always uses the HTTPS connection(i.e.: https://url.com) and pay attention to browser warnings if any when visit a site.

iii. How does a site like facebook.com provide SSL-stripping protection for its users?

They use HSTS which tells the browser to only accept HTTPS connections, so that users cannot make an insecure HTTP request to the server.

iv. Does HSTS (HTTP Strict Transport Security) prevent SSL-stripping from intercepting client's unencrypted requests? How?

In general it does prevent SSL-stripping from intercepting client's unencrypted requests by automatically turn any insecure links into secure ones. However it doesn't perfectly prevent SSL stripping. It cannot guarantee security of the initial requests from the clients if the request uses an insecure protocol(e.g. plain HTTP) or if the URL was obtained over an insecure channel. Also, if the request is sent after the activity period that specified in the advertised HSTS Policy(max-age), it will not be protects against SSL-striping.

4. **Protecting Code Distribution**
   a. **download patches/updates over HTTPS connections to its home server with company's public-key signed by a well-known trustworthy CA.**
   What checks should app instances perform on these server-certificates in order to protect themselves against attackers? Could attackers use SSLstrip to undermine this approach? Why/not?

      The app instances need to check if the public key is signed by a trusted CA, or a CA that roots back to a trusted root CA, by decrypting using CA's known public key and compare the hash to see if a match could be found, so that authentication can be ensured before an update/patch.

      The attacker cannot use SSLstrip to undermine this approach since all connections are secure from the beginning.

   b. **perform patches/updates over HTTPS without the need to purchase a CA-signed public-key certificate (the public key can still be used). Can secure patches/updates be delivered to apps as requested by management? Why/not?**

      Yes it could. The app instances(client) and the server are in the same factory when app instances are manufactured before distribution, therefore the digital signature mechanism can be adopted. i.e.:

      The company can encode the public key in the app instances when manufacturing, and signs the hash of update/patch by its private key before distribution. When the app instances receive the path/update over HTTPS, it can decrypt the hash using encoded public key and thus verify the server identity.

   c. **In order to implement management's P2P update proposal, would you recommend using the signature-scheme or the existing update-process based on HTTPS?**

I would recommend using the signature-scheme for the P2P update proposal. On P2P network, network connection may not always be secure (depending on user's behaviour), therefore SSLstripping become possible. However it will be prevented using the signature as the app instance cannot decrypt the signed hash if anything has been tampered, thus security is assured.

Which of the patch/update mechanisms would be more efficient, in terms of public-key operations performed for encryption and digital signatures?

The CA-signed public-key certificate will be more efficient in terms of public-key operations. For the signature mechanism, the company needs to encrypt the hash of the update/patch every time a new update/patch has been issued, and the app instances need to decrypt accordingly; while for the HTTPS connection with CA-signed certificate, we only need to verify if the CA is trusted once.

5. Exploiting Weak RSA Keys to Break HTTPS

(partnered with Yang Wang, utorID wangyang)

```
Modulus=E4AFE6A2C223F0C37BCEBACA8D21A69C31D98AC54C597DC9032F5CA37354B24525464211A1EDC0C9C44E48
30AE3A0DC2E8C3396892C2D938F5AF4080D0A5E4E4B181EB92CE3F0E8AED7FD50FF4A4069EBAD688EE4597AC6EDB0B
86D740D0A0030C6BF75D4DDB2583C1C6EBC0EBC951FA5BAAC4D774F7C521BE0E100BAB29E853

To decimal:
160589551961873197828445495034570233363247818671691660589162331874420605817572168663844611732 2
108260173257887638689302104402238364748394182569610119371998564446197844032343626118205628163 4
656869245493022813035201215225174738839659124310339376649114890270103759046805967585646999835 4
735584903579056642573592659

Using fermat factorization algorithm:
p=126723933004730088288935033953830579812702446908973871007259847154624397896523436992407609231
830466659974587125598542193198338312888984044510053565332201 67
q=126723933004730088288935033953830579812702446908973871007259847154624397896523436992407609231
830466659974587125598542193198338312888984044510053565332194 77

Create key file:
$ python rsatool.py -f PEM -o key.pem
-p
126723933004730088288935033953830579812702446908973871007259847154624397896523436992407609231 8
304666599745871255985421931983383128889840445100535653322016 7
-q
126723933004730088288935033953830579812702446908973871007259847154624397896523436992407609231 8
304666599745871255985421931983383128889840445100535653321947 7
```

Secret message:

```
POST /cscd27f15/LoginServlet HTTP/1.1
Host: cms-chorus.utsc.utoronto.ca:41414
Accept-Encoding: deflate, gzip
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Referer: http://www.utsc.utoronto.ca/~rosselet/refresh.html
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 AppleWebKit/537.36 Chrome/46.0.2490.71 Safari/537.36
Content-Length: 42
Content-Type: application/x-www-form-urlencoded

username=cscd27f15&password=dcf35c470d090eHTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=4546DC99C69ECD611121CD3D77D7ADF6; Path=/cscd27f15; Secure
Content-Type: text/html
Content-Length: 1600
Date: Wed, 11 Nov 2015 16:49:09 GMT

<html>
<head>
<title>Secrets of Happiness</title>
</head>
<body>
<h1>Secrets of Happiness</h1>
<ul>
<li>All happiness depends on courage and work.</li>
<li>The secret of happiness (and therefore of success) is to be in harmony with existence, to
be always calm, always lucid ... to let each wave of life wash us a little farther up the
shore.</li>
<li>It is neither wealth nor splendor; but tranquility and occupation which give you
happiness.</li>
<li>Action may not always bring happiness; but there is no happiness without action.</li>
<li>Be happy with what you have and are, be generous with both, and you won't have to hunt for
happiness.</li>
<li>True happiness comes from the joy of deeds well done, the zest of creating things
new.</li>
<li>Happiness is good health and a bad memory.</li>
<li>The grand essentials of happiness are: something to do, something to love, and something
to hope for.</li>
<li>Success is getting what you want. Happiness is wanting what you get.</li>
<li>Derive happiness in oneself from a good day's work, from illuminating the fog that
surrounds us.</li>
<li>Beauty is nothing other than the promise of happiness.</li>
<li>There are shortcuts to happiness, and dancing is one of them.</li>
<li>When I was 5 years old, my mother always told me that happiness was the key to life. When
I went to school, they asked me what I wanted to be when I grew up. I wrote down 'happy'. They
told me I didn't understand the assignment, and I told them they didn't understand life.</li>
<li>Ice cream is happiness condensed.</li>
</ul>
```

```
</body>
</html>
```

Process:
I passed the “reuse primes” case first, since there are no other servers we have access to; and I also decided to skip the two “low-value keys” options, thinking it is unlikely that the server will be using this as both sides might consider it being insecure; only balance/unbalanced cases are left. I looked up Pollard’s “rho” and Brent’s algorithm for Unbalanced case, and Fermat’s factorization algorithm for the Balanced case. I started with the Fermat’s factorization algorithm since it looks easier to implement, and luckily got the pair of p and q instantly. :)

6. **ARP-Cache Poisoning**
   a. **a simple mechanism, requiring little technical expertise, to perform an ARP-cache poisoning attack**

      One can use the Arpspoof tool from the DSniff suite to perform a MITM attack, by changing the IP mapping and pretending to be another identity. Steps are described as follows:

      1) enable IP forwards:
         `echo 1 > /proc/sys/net/ipv4/ip_forward`
      2) run arpspoof. Use “-i <network interface>”, followed by “-t <target>” (your victim’s internal IP), and specify the internet gateway in the end. e.g..:
         `arpspoof -i eth2 -t <victim IP> <internet gateway>`
      3) in a new terminal issue an iptables command
         `iptables -t nat -A PREROUTING -p tcp –destination-port 80 -j REDIRECT –to-ports 10000`
      4) run SSLStrip. Use “-w <output file>” to specify somewhere to dump off the data we are redirecting.
         `python sslstrip.py -w output.dmp`

      Thereby any HTTP/SSL web browsing that victim does will be dumped to the output file as we specified in the sslstrip command.

(reference: http://robospatula.blogspot.ca/2013/12/man-in-the-middle-attack-arpspoof-sslstrip.html)

   b. **Explain how how a man-in-the-middle attack can be conducted against a computer by poisoning that computer's ARP cache.**

      1) Alice wants to send a packet to Bob while Bob’s MAC address (b) is not in Alice’s ARP table.
      2) Alice then broadcast an ARP query packet containing Bob’s IP address (B), all machines on LAN receive this ARP query

3) Both Bob and Mallory eavesdrops the ARP query, replies to Alice with their MAC addresses(b and m)
4) The later response will override the earlier one, assume Alice receives the response from Bob first, then when Mallory's response arrives it will take Mallory's response assuming it's Bob, caches (B-m) IP to MAC address-pair in its ARP table
5) Afterwards all message Alice sends to Bob will be sent to Mallory
5) Mallory intercept the message, tampers it, and forward it to Bob on her behalf pretending to be Alice
6) In turn if Bob replies Alice by issuing an ARP query, Mallory can do the same thing to pretend to be Alice, and Bob caches a (A-m) address-pair.
7) Since Alice and Bob can both send & receive messages as expected, they will not notice Mallory in the middle.

c. **Describe one method for mitigating ARP-cache poisoning attacks. Which type(s) of attack are addressed by this method? Is this method always successful? Why/not?**

Enabling filtering protection offered by switches, which reduce network traffic by sending packets only where they are required, thus protected it from the eavesdropper as they loses access to packets sent between target hosts on the network. (MITM attack addressed).

It is not always successful, since by ARP MAC flooding the attacker sends out enough bogus ARP messages and bump legitimate entries out of limited-sized ARP-lookup table in switches, thus packets destined for bumped entries will revert to broadcast mode and enable eavesdropping again.

We can also use static ARP table if the LAN is small enough, by associating IP addresses of all hosts with their MAC addresses statically, thus eavesdropper won't have the chance to temper any entry. However if become very inefficient and impractical for large network since we need to keep this table for all hosts.

d. **Knowing the tool installed, how could they use that knowledge to mount a denial-of-service attack on that computer?**

An IP address will be blocked if the attacker sends out multiple ARP responses with the same IP addresses with more than 3 different MAC addresses within the 30 seconds frame. Therefore a Denial of service attack can be mounted by an attacker sending out massive false ARP responses to all devices on the network in the timeframe, thereby all IP addresses in the network will potentially be blocked and await admin's investigation.

7. **Denial of Service Attacks**

TCP SYN-Cookies

a. **Server using SYN-cookies, can a denial-of-service be achieved by flooding the server with ACK requests (third step of the TCP 3-way handshake)? Why/not?**

   No, the attacker cannot achieve a denial-of-service attack with SYN-cookies. “SYN-Cookies” return secure “token” to client when request arrives, where the token is derived from client request and server secret in sequence number; the valid client then send back ACK based on the returned cookie, the server will allocate resources to handle the request only if returned cookie matched and the handshake completed. However, the massive ACK requests are random and won’t match any cookies on the server, thus no server resources will be occupied by these connections.

b. **run Wireshark to collect a large number SYN-cookie values sent by the server, can the attacker achieve a denial-of-service by reflecting (sending) these SYN-cookie values back to the server in quick succession? Why/not?**

   Yes the attacker can achieve a denial-of-service in this case. The reflected cookies are related to the server in some sense, therefore it will take the server some time to recompute the cookies and compare with returned cookies to see if a match could be found. Therefore, sending those cookies in quick succession might exhausts the server resources (in the matching process).

Dos and Connection Tampering

c. **Describe how this ISP could conduct its DoS against any of its subscribers by injecting just a single packet into the network connections of that subscriber.**

   The ISP could send a packet with reset flag set, and thereby close the connection.

d. **Does the TCP/IP protocol suite have any built-in defence itself against this kind of "attack"? If so, describe it; if not, why not?**

   No, it doesn’t have any built-in defence. TCP/IP headers are not encrypted/encapsulated, thus it won’t detect any differences whether a RST packet has been set or not.

e. **Could someone other than the ISP conduct a similar attack? If not, why? If so, how would this work (be sure to mention any special resources that this other party would need access to)?**

   Yes, if the attacker can guess/obtain the current sequence number for that connection, tampering the connection by injecting a packet with reset flag set would close the connection and achieve the Dos attack. i.e.:

The attacker can obtain the TCP state by eavesdropping (e.g. with Wifi AP) and monitor the traffic by some tools like wireshark, and guess possible sequence number(with initial sequence number and amount of traffic), sending a flood of packets with these possible numbers and find out the correct one.

f. **Could subscribers protect themselves against this attack by using only HTTPS/SSL-encrypted packets? Why/not?**

No, SSL can only encrypt the message in the TCP packet, but not the TCP and IP header. Thus the DoS attack stated above will still work.

g. **ISP (Bell) injected content into a subscriber's Web connection**

i. **Describe how "mixed-content" Web app/pages can be exploited to perform "man-in-the-middle" (MITM) attacks.**

The main page loads over https, but includes non-secure content such as images, scripts or css files. In the case of unsecure image loading, the attacker would be able to change the actual image presented to the user, which affects the data integrity. While if the script is also insecure, it will be even more dangerous since the attacker might be able to change the behavior of the HTTPS page by tampering the script, send it to the client and mislead the user.

ii. **Can an ISP tamper with the data in a secure HTTPS connection? If not, what prevents tampering? If so, what mechanism would the ISP utilize to perform tampering?**

No it cannot tamper with the data. In a secure HTTPS connection, everything is encrypted between user computer and the remote server, so the ISP can't intercept any of the data. The secure socket layer(SSL) prevents tampering.

iii. **Could an ISP tamper with the data in an insecure HTTP connection run over a VPN connection? If not, what prevents tampering? If so, what mechanism would the ISP utilize to perform tampering?**

No it cannot tamper with the data over a VPN connection. A Virtual Private Network creates a tunnel through the Internet to an internal VPN server and assigns an internal IP address to the client, therefore it provides protection to the connection around user's local network, thus protect the data against MITM attack such as ISP injection (The ISP will have no access to the data user send out/receive thus won't be able to tamper with it in the communication).

8. **Password Security**

(coding part partnered with Yang Wang, utorID wangyang)

a. **Password Storage: Encryption or Hashing?**

i. **Was 3DES, together with their plaintext password hints a stronger or weaker choice than storing an SHA-1 hash of user-passwords?**
3DES is weaker than SHA-1 hash in terms of comparative cracking vulnerabilities. SHA-1 is a hash algorithm, which reduces the plaintext to a hash and cannot be reversed; while 3DES is a symmetric cipher with a reversible encryption algorithm, which means plaintext password can be recreated from the ciphertext once the key is broken.

ii. **What advantage would use of ECB in 3DES with the various fields such as userid, password, hint, aligned on block boundaries provide password crackers?**
ECB mode splits messages into blocks and each is encrypted separately. The disadvantage of this method is that identical plaintext blocks are encrypted to identical ciphertext blocks; therefore it somehow provides plaintext data patterns to the cracker.

Suppose your Adobe account was one of the ones stolen in the breach.

iii. **Would knowing your password and having access to the encrypted stored value give you any advantage in cracking other user's passwords?**
Yes**.** Knowing one password and encrypted value is not sufficient for breaking the 3DES algorithm using current computing technology, however, since identical plaintext blocks are encrypted to identical ciphertext blocks, it exposes those passwords with identical plaintext blocks as my password.

From 3DES to SHA-256. Attacker has access to GPU hardware capable of performing 1.2G-hashes/second (1.2×10^9). Passwords as truly random 16-byte values.

iv. **How many bits of entropy do these passwords have?**
Each byte(8 bits) has 2^8 possible values, with 16-bytes values the size of the random-value space is $(2^8)^{16}$. Therefore the bits of entropy is $\log_2 (2^8)^{16} = 8*16 = 128$

v. **About how long, in years, would it take the attacker using the above GPU to find a hash match for one such user?**
total number of possible hashes: $(2^8)^{16}$
time spent to find a match out of these hashes:
$2^{218} / (1.2 * 10^9) / 60 / 60 / 24 / 365 = 899190255900501182414 2$
$\sim 9 * 10^{21}$ years

vi. **password check for user-authentication based on SHA-256:**

see mathlab submission : pwd_hash.py

The reality is that users selects a 6-character password that is memorable, but not necessarily random, choosing characters from the set {A-Za-z0-9}.

vii. **How many bits of entropy do these passwords have in the best case (most secure/random assumptions)?**
Each character has 26 + 26 + 10 = 62 possible values, with 5 characters the size of the random-value space is 62^6.
Therefore the bits of entropy is $\log_2 62^6 \sim 35.7$

viii. **On average, how many years would it take the attacker with the GPU to find a hash-match for one such user (under the most secure/random assumptions)?**
total number of possible hashes: 62^6
time spent to find a match out of these hashes:
62^6 / (1.2 * 109) / 60 / 60 / 24 / 365 = $1.5009363791645527 * 10^{-6}$ years

ix. **If you were the attacker, would a brute-force search of the password-string space be an efficient approach for password recovery? Briefly either explain why, or describe an approach that might be faster/cheaper.**
Yes I will use a brute-force search on the {A-Za-z0-9} space. From the question above we can see that breaking one such hash only takes around 47 seconds, therefore it is worthwhile to try as it's highly likely that many people chooses their password from that set.

b. **Cracking Passwords with Lookup Tables**

i. **How many bytes of storage space would be required, for the most-secure/random case for the "average" user, whose 6 password-characters are chosen from the set {A-Za-z0-9} ?**
There are 62^6 different arrangements of the characters, each character takes 1 byte with 6 characters in total, and each hash takes 256 bits which is 32 bytes
62^6 * (6 + 32) = 2158408952192 ~ $2.16 * 10^{12}$ bytes

ii. **"low hanging fruit" as hash of 10-million 6-character "words" taken from dictionaries and previous password breaches, what are the storage requirements in bytes?**
10 million entries in the lookup table , each entry takes (6 + 32) bytes: $10^7 * 38 = 3.8 * 10^8$ bytes

"Rainbow Table"

iii. **If the attacker sets their chain-length to 10,000, and we assume a perfect non-intersecting set of chains that cover the plaintext input space, how much storage space will the rainbow table consume for the most-secure/random password case in question 8Bi) above?**

Total number of possible passwords: 62^6
Given the chain-length is 10^4, the total number of entries(chains) in the rainbow table is: $62^6 / 10^4 \sim= 5.68 * 10^6$
For each entry, we need to store 2 6-character plaintext: 12 bytes.
$5.68 * 10^6 * 12 \sim= 68160282.7008$ bytes
Also need to store reduction functions, assuming each costs x bytes, the storage space needed is :

1000x + 6.8 * 10^7 bytes

iv. **For a plaintext-password set of size N, suppose rainbow-table construction costs roughly 1.7*N hash operations, and hash-lookup in a rainbow-table with k-length chains costs roughly (k * k)/2**
Compare these time-cost estimates with the time to create a complete lookup-table of (password, hash) pairs, and the time to lookup a hash value in such a complete table of (password,hash) pairs.

Time-cost estimates for constructing a complete lookup-table is N, while look-up is constant time. Both estimates are less than that of the rainbow table (1.7*N and (k*k) /2). That's why we say raintable is a space-time trade-off.

c. **Password Salting**

i. **Describe an approach to reliably determine whether LinkedIn was salting passwords.**

Check two accounts using same plaintext as passwords and see if their hash values match. If so, no salting is used, otherwise salting is used.

ii. **Describe 2 specific advantages that an attacker would gain if salting was not used by LinkedIn.**

1. People who used same passwords will share the same hash
2. Less storage space is needed since no need to combine passwords with random strings (of length of the salt)

iii. **Suppose an attacker has targeted a select set of specific accounts to crack from the LinkedIn data. What impact does salting have on this targeted attack?**

With salting, users with same password have different entries in the password table; attack table must account for all possible hash values therefore the cost of attack is much higher.

iv. **Modify your Python code from the preceding problem to hash passwords with a salt value.**

see mathlab submission : pwd_hash_salt.py

v. **What impact would the addition of a 16-byte salt value have on the time/space cost of the rainbow-table attack from the previous problem?**

The total number of possible arrangements will be $62^6 * 256^{16}$ which is way greater than before. It's will be impractical to store such a large number of entries in the rainbow-table (even worse for the complete lookup table!).

d. Slow Hashing

see mathlab submission : pwd_hash_salt_slow.py, pwd_hash_salt_bcrypt.py