1. Select a message of your choice. Create a hash of the message using the MD5
(message digest - RFC 1321) algorithm. Sign the hash. Finally, verify the signed
message. Note that in order to sign and verify the message, you will need to
create public-private key pairs, something that you have done in a previous
assignment. We suggest that you use OpenSSL (e.g., learn how the openssl
digest command works) to perform all of the steps in this problem, although you are
free to use any other tools.

Solution:
(a) create an MD5 hash of some arbitrary message (e.g., of some arbitrary file)
(b) sign the hash using a private key
(c) check the signature using the matching public key
(d) compute the hash of the message and verify that the hash matches the signed
hash

2. Log in to an ftp server using user name and password. The ftp server's
network address is 152.3.136.182. The user name is ftp test and the password is
cps290 02. Demonstrate that the password is sent in clear text by providing a
snapshot of a packet trace using Wireshark or other appropriate tool. Note: This
exercise is only for understanding the security flaw with the ftp protocol. In
practice, you should always use the secure version of ftp (SFTP) or some other
secure file transfer protocol. Also note that we are only asking you to capture
your own packets and the password to this dummy account. Capturing the
packets and passwords of other users is not encouraged and may not be legal.

Solution:
Using tools like Wireshark to show the packets.

3. In the previous problem, you demonstrated that the FTP protocol sends user
names and passwords to the FTP server “in the clear”, i.e., unencrypted, which
generally seems like a bad idea. One circumstance in which it might be safe to
send passwords in the clear, though, is when a password can only be used once.
(But FTP also sends data in the clear, so solving the password problem isn't the
end of the story.)
(a) One way to set up one-use-only passwords would be to store, in advance, a
list of passwords on the server and also somewhere accessible to the user, and
then use these passwords sequentially. An observant student notices that if it is
possible to securely store a large amount of shared private data on the server
and on the user’s machine in advance, then another approach could be applied
that would provide the most secure type of communication possible. What is the
student thinking of?
(b) A cryptographically secure pseudo-random number generator (CSPRNG) is
initialized with a key, and guarantees that, given the first k bits that have been
generated, there is no polynomial time algorithm for predicting the (k + 1)st bit
with probability much higher than 50% without access to the key. How could a
CSPRNG be used to set up one-use-only passwords without storing a large
amount of shared private data on both machines in advance?
Solution:
(a) If a large amount of private shared data can be stored, then the server and the user’s machine can use that data as a one-time pad.
(b) Two possible answers here. First, the server and the user’s machine could use the bits generated by the CSPRNG as a replacement for the one-time pad described in (a). Or, more directly, suppose that during each login a b-bit one-time password is to be used. Then both the server and the user’s machine generate the next b bits using the CSPRNG and use these as the password.

4. In Problem 1 of Assignment 1, you helped a fellow student named Bob who wanted to be able to make secure connections using the SSH protocol without having to type passwords. Suppose Bob’s main account is on a university-administered Linux computer called livingontheedge.cs.duke.edu, and he frequently connects to a computer owned by his employer, gatewaymachine.getrichquick.com. Bob’s employer allows SSH access only through the use of public-private key pairs, and not through password authentication. Bob thinks this is just fine, because he hates to type passwords anyway. So he creates a public-private key pair for himself, in which the private key has no passphrase. He then installs the public key on gatewaymachine.getrichquick.com, and can log in from livingontheedge.cs.duke.edu using the private key without typing a passphrase.

(a) What protects Bob’s private key on livingontheedge.cs.duke.edu?
(b) What sort of compromise is Bob’s private key vulnerable to?
(c) What type of compromise did Bob’s employer seek to limit by disallowing password authentication on gatewaymachine.getrichquick.com?

Solution:
(a) Bob’s private key is protected by his login password and the file access controls on the Linux computer, which should not allow anyone other than Bob to read the file.
(b) Bob is not the administrator of the Linux machine, so someone else logged in with administrator privileges can change the ownership of the file, or login in as Bob and look at the file.
(c) The employer does not want passwords sent to the gateway machine. The concern is that if the machine is compromised, someone can collect user passwords, and possibly use them to gain access to other machines. (Users have a bad tendency to use the same password to access multiple resources.)

5. One minor vulnerability of the RSA cryptosystem is that given two signed messages m1 and m2, i.e., m1^d (mod n) and m2^d (mod n), where d is the signer’s private key, by simply multiplying these signatures together to form m1^d (mod n) \cdot m2^d (mod n) = (m1 \cdot m2)^d (mod n), it is possible to forge the signature of a new message m1 \cdot m2, which is the product of messages m1 and m2, without knowing d. Recall that in order to use the RSA cryptosystem it is first necessary to convert a message, such as a string of text, into an integer so that modular arithmetic operations can be performed on it, such as raising it to the power d. In general, the product of two integers representing strings will not yield an integer
representing an interesting new string. But if each of the original messages is a simply integer, and these integers are directly signed, then the new message will also be an integer. For example if m1 is the integer 290 and m2 is the integer 2, then given $290^d \pmod{n}$ and $2^d \pmod{n}$, it is possible to forge the signature $580^d \pmod{n}$, i.e., the signature for a message that is the integer 580.

There are several straightforward ways to protect against this vulnerability. You should explain in just a few sentences how each of the following two approaches solves the problem.

(a) The first approach is to establish a convention that every message that is to be signed should be padded with a string of k trailing random bits, where the length of the string, k, is fixed and known to all who obey the convention. For example, the signature for message m1 would be $(m1 \cdot 2^k + r1)^d \pmod{n}$, where r1 is a random k-bit integer. (Note that to verify the signature, the random string r1 and the original message m1 must also be known.)

(b) The second approach is to establish a convention that rather than signing the message, a hash of the message should be signed instead, e.g., the signature of m1 would be $(\text{MD5}(m1))^d \pmod{n}$, where m1 is viewed as a bit string when input to the MD5 hash function.

Solution:
(a) $(m1\cdot2^k+r1)^d \cdot (m2\cdot2^k+r2)^d$ is not guaranteed to be equal to something of the form $(m1\cdot m2 \cdot 2^k + r3)^d$, and indeed almost certainly will not be
(b) $(\text{MD5}(m1))^d \cdot (\text{MD5}(m2))^d$ is not guaranteed to be equal to something of the form $(\text{MD5}(m1 \cdot m2))^d$, and indeed almost certainly will not be

6. Find a web site that is using a CDN for full-site delivery (e.g., to thwart denial of service attacks), and demonstrate that this is the case.

Solution:
For example, first thing to do is a DNS query for a web site (www.microsoft.com) using dig and discover that the authoritative name servers were operated by Akamai.
Then do a reverse DNS query using dig or host, and discover that the actual IP address returned for www.microsoft.com has been registered with Akamai too.
If you go on a bit farther and done a packet trace of downloading www.microsoft.com to see if ALL of the components on that web page are delivered by Akamai (this means you are taking the phrase "full-site delivery" very literally). And then observe that some parts of the site are delivered by other parties and therefore the site isn't really fully delivered by Akamai. That is OK.

7. Find a web site that is using a CDN to direct browser requests to its mirror sites, and demonstrate that this is the case.

Solution:
The procedure is similar to the previous question, the difference is to observe that the IP address returned for the web server (e.g., www.aol.com) was actually registered to aol.com.

8. Suppose that a content provider gives a copy of its private keys to a Web hosting service that will operate its Web site and host its home page, so that end-user browsers can connect securely to the Web servers operated by the Web hosting service using the standard SSL/TLS protocol. The content provider doesn’t fully trust the Web hosting service, so it decides that when really sensitive information must be communicated, such as the user’s password, the Web site will direct the client to connect securely to a server operated by the content provider rather than the hosting service. You have been asked by the content provider to evaluate whether this approach is sound, assuming that the Web hosting service is malicious. You are immediately skeptical about whether this approach provides much security. How can the hosting service bypass this measure, given that it has the private keys?

Solution:
The hosting service is delivering the HTML, so before sending the HTML to the browser, the web server can modifying it so that it does not contain any links back to servers operated by the bank, but instead points to servers operated by the hosting service. The hosting service then can act as a man-in-the-middle. There is no way for the browser to know whether it is really talking to the bank or not.
Note that if the hosting service does not modify the HTML and allows the browser to connect to servers operated by the bank, then the hosting service CANNOT eavesdrop on traffic sent by the browser, because the bank is using DIFFERENT private keys on its servers than the ones that it gives to the hosting service.