Solution: CS120 Homework Assignment #4

Name: _____ ID: _____

Please read the following instructions carefully before answering the questions:

- This assignment is to be completed by each student **individually**.
- There are a total of **7** questions.
- When you write your answers, please try to be precise and concise.
- Fill your name and student ID at the first page.
- Please typeset the file name and format of your submission to the following one: YourID_CS120_HW4.pdf (Replace "YourID" with your student ID). Submissions with wrong file name or format will **NOT** be graded.
- Submit your homework through Blackboard.

1. (10 points) DNS

   (a) Suppose you can access the caches in the local DNS servers of your department. Can you propose a way to roughly determine the Web servers (outside your department) that are most popular among the users in your department? Explain.

   Yes, via checking the DNS cache

   (b) Suppose that your department has a local DNS server for all computers in the department. You are an ordinary user (i.e., not a network/system administrator). Can you determine if an external Web site was likely accessed from a computer in your department a couple of seconds ago? Explain.

   Yes, by measuring the queuing delay. None-cached DNS record results in more delay

2. (30 points) In this problem, we explore the Diffie-Hellman (DH) public-key encryption algorithm, which allows two entities to agree on a shared key. The DH algorithm makes use of a large prime number p and another large number g less than p. Both p and g are made public (so that an attacker would know them). In DH, Alice and Bob each independently choose secret keys, S_A and S_B, respectively. Alice then computes her public key, T_A, by taking $g^{S_A} \bmod p$. Bob similarly computes his own public key $T_B = g^{S_B} \bmod p$. Alice and Bob then exchange their public keys over the Internet. Alice then calculates the shared secret key S by $T_B^{S_A} \bmod p$. Similarly, Bob calculates the shared key S' by $T_A^{S_B} \bmod p$.

(a) Prove that, in general, Alice and Bob obtain the same symmetric key, that is, prove S=S'.

	Alice	Bob
secrect key:	S_A	S_B
public key:	$T_A = (g^{S_A}) \bmod p$	$T_B = (g^{S_B}) \bmod p$
shared key:	$S = (T_B^{S_A}) \bmod p$	$S' = (T_A^{S_B}) \bmod p$

$S = (T_B^{S_A}) \bmod p = ((g^{S_B} \bmod p)^{S_A}) \bmod p = (g^{(S_B S_A)}) \bmod p$

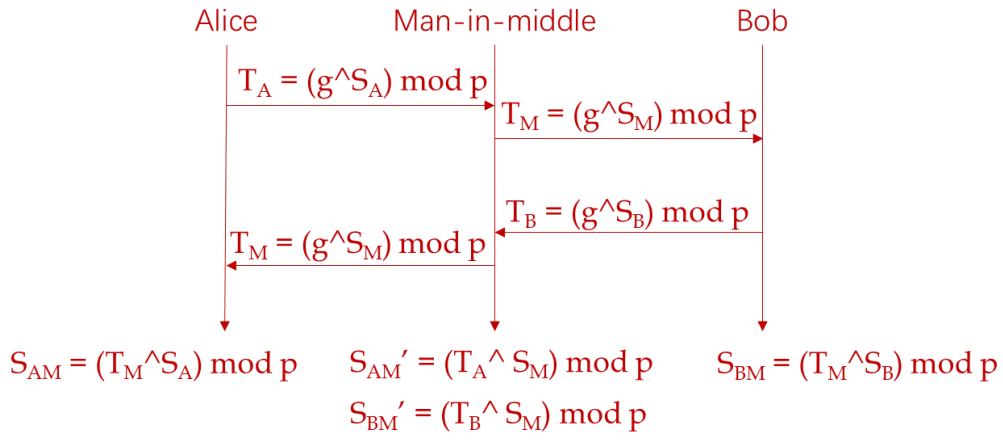
$= ((g^{S_A} \bmod p)^{S_B}) \bmod p = (T_A^{S_B}) \bmod p = S'$

(b) With p = 11 and g = 2, suppose Alice and Bob choose private keys $S_A = 5$ and $S_B = 12$, respectively. Calculate Alice's and Bob's public keys, T_A and T_B.

p = 11, g = 2

	Alice	Bob
secrect key:	$S_A = 5$	$S_B = 12$
public key:	$T_A = (g^{S_A}) \bmod p = 10$	$T_B = (g^{S_B}) \bmod p = 4$

(c) Provide a timing diagram that shows how Diffie-Hellman can be attacked by a man-in-the-middle. The timing diagram should have three vertical lines, one for Alice, one for Bob, and one for the attacker Trudy.



3. (10 points) Suppose a file contains the letters *a*, *b*, *c*, and *d*. Nominally we require 2 bits per letter to store such a file.

   (a) Assume the letter *a* occurs 50% of the time, *b* occurs 30% of the time, and *c* and *d* each occurs 10% of the time. Give an encoding of each letter as a bit string that provides optimal compression. (Hint: Use a single bit for *a*.)

   (b) What is the percentage of compression you achieve above? (Compared with using 2 bits per letter)

   (a) a: 1, b: 01, c: 001, d: 000 or other valid encodings (a: 0, b: 10, c: 101, d:100)

   (b) $1 \times 0.5 + 2 \times 0.3 + 3 \times 0.1 + 3 \times 0.1 = 1.7$ So the compressed data uses $1.7/2 \times 100 = 85\%$ as many bits, or a 15% compression gain.

4. (10 points) How might you encode audio (or video) data in two packets so that if one packet is lost, then the resolution is simply reduced to what would be expected with half the bandwidth? Explain why this is much more difficult if a JPEG-type encoding is used.

   For audio data we might send sample[n] for odd n in the first packet, and for even n in the second. For video, the first packet might contain sample[i,j] for i+j odd and the second for i+j even; dithering would be used to reconstruct the missing sample[i,j] if only one packet arrived.

   JPEG-type encoding (for either audio or video) could still be used on each of the odd/even sets of data; however, each set of data would separately contain the least-compressible low-frequency information. Because of this redundancy, we would expect that the total compressed size of the two odd/even sets would be significantly larger than what would be obtained by conventional JPEG compression of the data.

5. (20 points) Consider the following simplified BitTorrent scenario. There is a swarm of 2^n peers and, during the time in question, no peers join or leave the swarm. It takes a peer 1 unit of time to upload or download a piece, during which time it can only do one or the other. Initially, one peer has the whole file and the others have nothing.

   (a) If the swarm's target file consists of only 1 piece, what is the minimum time necessary for all the peers to obtain the file? Ignore all but upload/download time.

   (b) Let x be your answer to the preceding question. If the swarm's target file instead consisted of 2 pieces, would it be possible for all the peers to obtain the file in less than $2x$ time units? Why or why not?

   (a) All peers could have the file after n time units. During each time unit, each peer with the piece can transmit it to one peer without the piece, so the number of peers with the piece doubles with each time unit: 1 (= 2^0) at time 0, 2 (= 2^1) at time 1, 4 (= 2^2) at time 2, up to 2^n peers at time n. So it will take n unit of time.

   (b) Yes. If all pieces were downloaded to just the right peers in just the right order, it is possible for all peers to obtain the file after as few as n + 2 time units. The case is:

   Let's label the two pieces A and B. Let's label as PA the peer that initially has the file. During the first time unit, PA transmits B to another peer, call that other peer PB. Split the peers into two equal groups of 2(n−1), one containing PA and the other containing PB. Now, from the result of the first question, we know that all the peers grouped with PA can obtain A within an additional n − 1 time units. Because the two sets of peers are disjoint, with no interference or contention between them, all the peers groupedwith PB can obtain B during the same n − 1 time units.

   Together with the initial step, we have used n time units so far. Another time unit will suffice for the peers grouped with PA to transmit A to all the peers grouped with PB. One more time unit will suffice for the peers grouped with PB to transmit B to all the peers grouped with PA (except PA itself, which has had B from the beginning). The 2 time units required for each half to transmit its piece to the other half increases the total to n + 2 time units.

6. (10 points) One mechanism for resisting replay attacks in password authentication is to use *one-time passwords*: A list of passwords is prepared, and once $password[N]$ has been accepted the server decrements N and prompts for $password[N-1]$ next time. At $N = 0$ a new list is needed. Outline a mechanism by which the user and server need only remember one master password mp and have available locally a way to compute $password[N] = f(mp, N)$. Hint: Let g be an appropriate one-way function (e.g., MD5) and let $password[N] = g^N(mp) = g$ applied N times to mp. Explain why knowing $password[N]$ doesn't help reveal $password[N-1]$.

   We have $password[N] = g(password[N-1])$; the essential property of g is that it be believed that knowing g(x) does not provide any information that can be used to find x.

7. (10 points) Suppose we have a very short secret s (e.g., a single bit or even a Social Security number), and we wish to send someone else a message m now that will not reveal s but that can be used later to verify that we did know s. Explain why $m = \mathrm{MD5}(s)$ or $m = \mathrm{E}(s)$ with RSA encryption would not be secure choices, and suggest a better choice.

Because s is short, an exhaustive search conducted by generating all possible s and comparing the MD5 checksums with m would be straightforward. Sending MD5(s⁀r), for some random or time-dependent r, would suffice to defeat this search strategy, but note that now we would have to remember r and be able to present it later to show we knew s. Using RSA to encrypt s⁀r would be better in that sense, because we could decrypt it at any time and verify s without remembering r.