DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.033—Computer System Engineering | Handout 37—April 24, 1997

Quiz 2 Solutions

This quiz contains 4 problems whose relative weights are given below. Please write your answers in the space provided, and hand in this booklet at the end of the examination.

Put your name on this cover sheet AND at the bottom of each page of this booklet. Be sure that you don't put part of the answer to a problem on the back of a sheet for another problem.

Some parts of some problems may be much harder than others. Read them all through first and attack them in the order that allows you to make the most progress. Some problems are intentionally ambiguous; be sure to write down any assumptions you make. Show your work, or you risk losing partial credit. Be neat. If we can't figure out your answer we can't give you credit. On answers that involve numbers, be sure to clearly specify the units of your answer.

Circle your recitation:

10:00	1. Lynch/Banerji			
11:00	2. Lynch/Lau	7. Engler/Winship	12. Shrira/Banerji	
12:00	6. Engler/Lau			
1:00	3. Sussman/Sapuntzakis	9. Ward/MacGregor	8. Saltzer/Schindler	11. Shrira/Winship
2:00	4. Sussman/MacGregor	5. Ward/Schindler	10. Saltzer/Sapuntzakis	

For Official Use Only

Problem	Your Score	Maximum Score
1		16
2		32
3		24
4		28
Total		100

Name: ____________________

1. (*16 points*) Louis Reasoner has been using a simple RPC protocol that works as follows:

```
client -> server:    {nonce, function, arguments}
server -> client:    {nonce, response}
```

The client sets a timer, and if it does not receive a response before the timer goes off, it restarts the protocol from the beginning, repeating this sequence as many times as necessary until a response returns. The server maintains a table of nonces and responses, and when it receives a request containing a duplicate nonce it repeats the response, rather than repeating execution of the function. The client similarly maintains a list of nonces for which no response has yet been received, and it discards any responses for nonces not in that list, assuming that they are duplicates. One possible response is "unknown function", meaning that the server received a request it didn't know how to handle. A lower network layer checksums all packets and discards any that are damaged in transmission.

Louis wants to make this protocol secure against eavesdroppers. Louis has discovered that the client and the server already share a secret key, K_{cs}, for a symmetric-key encryption system. So the first thing he tries is to encrypt the requests and responses of the simple RPC protocol:

```
client -> server:    {nonce, function, arguments}^Kcs
server -> client:    {nonce, response}^Kcs
```

This seems to work, but Louis has heard that if you use the same key to repeatedly encipher predictable fields such as function names, someone may eventually discover the key by cryptanalysis. So he wants to use a different key for each RPC call. To minimize the coding effort, he changes the protocol to work as follows:

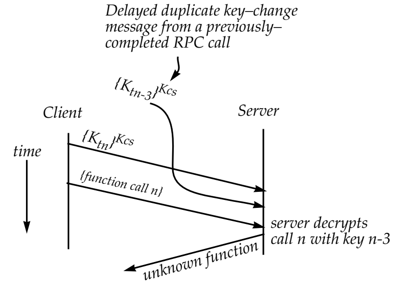
```
client -> server:    {K_tn}^Kcs
client -> server:    {nonce, function, arguments}^Ktn
server -> client:    {nonce, response}^Ktn
```

in which K_{tn} is a one-time key chosen by the client to be used only for the n'th RPC call before beginning the protocol. When the server receives a key, it deciphers it and uses it until the server gets another key message. Louis figures that since K_{cs} is now being used only to encipher randomly selected keys, it should be safer from cryptanalysis.

At first, this protocol, too, seems to work. Then Louis notices that the client is receiving the response "unknown function" much more often than it used to.

1a. Explain why, using a timing diagram to demonstrate an example of the failure.

1a. The problem is that the server is trying to decrypt a function call using the wrong key--the client and the server have gotten out of sync. If the server doesn't know what to expect for a nonce, and the arguments depend on what function is being called, the only clue that the decryption did not go well is that the function name doesn't appear in the list of known functions.



Name: ____________________

1b. Offer a suggestion that should fix this problem.

There are at least two ways to fix this protocol. Both involve tying the key to the corresponding function call more tightly.

1. Include the nonce for this call in the key-change message, so the server can discard duplicate key–changes and record new keys in the nonce/response table. Then, when a function call arrives, decrypt it with the newest key in that table and compare nonces to confirm that the decryption worked. If it didn't, go through the table trying earlier keys to try to find one that works. If none work, discard the packet.

2. Combine and send the two messages from client to server in a single packet:

client -> server: $(\{K_{tn}\}^{Kcs}, \{nonce, function, arguments\}^{Ktn})$

2. *(32 points)* ByteStream Inc. sells three data-transfer products: Send-and-wait, Blast, and Flow-control. Mike R. Kernel is deciding which product to use for his advanced kernel, and he performs a series of back-of-the-envelope calculations to select the appropriate product. Since each product transfers lots of data, the stream has to be broken up into packets. The protocols work as follows:

Send-and-wait sends one packet and then waits for an acknowledgment before sending the next packet.

Flow-control uses a sliding window of 8 packets. The sender sends until the window closes (i.e., until there are 8 unacknowledged packets). The receiver sends an acknowledgment as soon as it receives a packet. Each acknowledgment opens the sender's window with one packet.

Blast uses only one acknowledgment. The sender blasts all the packets in a message to the receiver. The last packet of the blast contains a bit indicating that it is the last packet of the message. After sending all packets in a single blast, the sender waits for one acknowledgment from the receiver. The receiver sends an acknowledgment as soon as it receives the last packet.

You are asked to compute for each the protocol the maximum throughput. Mike is planning to use a 1,000,000 byte/sec network with a packet size of 1,000 bytes. The propagation time from the sender to the receiver is 500 microseconds. Mike assumes (1) there is no processing time at the sender and the receiver; (2) that the time to send an acknowledgment is just the propagation time (number of data bytes is zero); and (3) the data packets are always 1,000 bytes. He also assumes that the underlying communication medium is perfect (packets are not lost, packets are not duplicated etc.) and that the receiver has unlimited buffering.

2a. What is the maximum throughput for the Send-and-wait? *Check one.*

___1,000,000 byte/sec

___666,000 byte/sec

X__500,000 byte/sec

___200,000 byte/sec

It takes 1,000 microseconds to push a 1,000-byte packet into the network. Since the propagation delay is 500 microseconds, the last byte of the packet will arrive at the receiver after 1,500 microsecond. The acknowledgment takes 500 microseconds, so the time to send one packet and receive an acknowledgment is 2,000 microseconds. The maximum throughput therefore is 500,000 byte/sec.

2b. What is the maximum throughput for Flow-control? *Check one.*

X_1,000,000 byte/sec

___666,000 byte/sec

___500,000 byte/sec

___200,000 byte/sec

The sender can push one packet into the network every 1,000 microseconds. The send window will never close, since before the sender has sent 8 packets, acknowledgments will have come in for earlier packets, keeping the window open (assuming the network is full duplex). Therefore, the throughput is 1,000,000 byte/sec.

2c. What is the maximum throughput for Blast? *Check one.*

X__1,000,000 byte/sec

___666,000 byte/sec

___500,000 byte/sec

___200,000 byte/sec

For a large message the maximum throughput is equal to the rate at which the sender can push packets into the network, since the one acknowledgment at the end is neligible compared to sending many 1,000-byte data packets. Since the sender can push one 1,000-byte packet into the network every 1,000 microseconds, the maximum throughput is 1,000,000 byte/sec.

Mike needs to choose one for an application which periodically sends arbitrary-sized messages. He has a reliable network, but his application involves unpredictable computation times at both the sender and the receiver. There is also a 20,000-byte buffer for receiving packets at the receiver.

2d. Which product should he choose for maximum reliable operation? *Check the best answer of the ones listed (i.e., the most specific answer).*

___Send-and-wait, the other might hang.

___Blast, which outperforms the others.

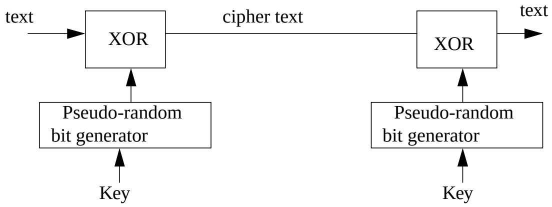
X__Flow-control, since Blast will be unreliable and Send-and-wait is slower.

___I cannot tell from the info given (*briefly explain*!).

Because messages can be very large and the receiver might take a long time to process a packet, Blast might overrun the 20,000-byte buffer at the receiver, causing packets to be dropped. Flow-control will not overrun the 20,000-byte buffer, since its send window is only 8,000 bytes (8 packets). Since Blast might be unreliable and Send-and-wait is slower than Flow-control, Flow-control is the right product to use.

Name: ____________________

3. *(24 points)* Because of the recent interest in security in the popular press, ByteStream Inc. decides to extend the functionality of its products with encryption. ByteStream decides to use a simple symmetric-key system:



Both for encryption and decryption ByteStream uses the exclusive-OR (XOR) function as shown above. The pseudo-random number generator generates a stream of random bits. If seeded with the same key, it will generate the same stream of random bits. Messages are encrypted by computing the XOR of the message and the random bits produced by the generator, seeded with some key. The resulting ciphertext is decrypted by computing the XOR of the ciphertext and the random bits produced by the random-bit generator, seeded with some key. If the keys used in encrypting and decrypting are the same, then the resulting decrypted ciphertext will produce the original message. The code for computing the XOR and the pseudo-random bits is publically available, but the key is secret.

To check the implementation, ByteStream Inc. hires a tiger team that include Eve S. Dropper and Lucy Fer. The tiger team verifies that the code for computing the XOR and the random-bit generator does not contain cryptographic weaknesses. The tiger team subsequently studies the following scenario. Alice shares a 200-bit key K with Bob. Alice encrypts a message with K as the seed for the random-bit generator. She sends the resulting ciphertext to Bob. Bob decrypts this message with K as the seed for the random-bit generator. The result after decryption is Alice's message. Assume that every message is equally likely (i.e., Alice's message does not contain redundancy).

3a. Given that Eve sees only the cipher text, is she able to decode Alice's message? *Check the best answer possible of the ones listed (i.e., the most specific answer).*

X_ No, since only Alice and Bob know the key, and the pseudo-random generator generates a 0 or 1 with equal probability, Eve has no way of telling what the content of Alice's message is.

___Yes, since with a supercomputer Eve could try out all possible combinations of 0s and 1s for K and check whether they match the cipher text.

___No, since it is hard to compute the XOR of two bit streams.

___Yes, since XOR is a very simple function, and Eve can just compute the inverse of XOR.

Eve can try all keys K, but she cannot verify whether a guessed key is correct or not, since every message is equally likely. Therefore, answer 1 is the correct one.

Name: ____________________

3b. Alice and Bob switch to a new shared key K. Lucy mounts an active attack by tricking Alice into sending a message that begins with 500 1s, followed by Alice's original message. Given the ciphertext can Lucy figure out Alice 's message? *Check the best answer possible of the ones listed (i.e., the most specific one).*

X__Yes, since the key is smaller than 500 bits.

X__Maybe, but with probability so low that it is negligible.

___No, since only Alice and Bob know the key and the pseudo random generator generates a 0 or 1 with equal probability, Lucy cannot guess Alice's message.

___No, since it is hard to compute the XOR of two bits.

Suppose C is the output of the random-bit generator when seeded with key K and M is Alice's message. Lucy knows the first 500 bits of the message M and the first 500 bits of M XOR C. She can now start guessing keys K', generate a random-bit stream C' (the code is publically available), and verify whether her guess is correct by comparing the first 500 bits of M XOR C and M XOR C'. If they are identical, then K' must be the same as K, and she can decrypt the complete message M. Since it is unlikely that Lucy will quickly guess the correct 200-bit key, the correct answer is 2. (One can argue that answers 1 and 2 are equally specific, so we accepted both.)

ByteStream is interested in a product that supports two-way communication. ByteStream implements two-way communication by having one stream for requests and one stream for replies. ByteStream seeds both streams with the same key. Since ByteStream worries that using the same key in both directions is a weakness, it asks the tiger team to check the implementation.

The tiger team studies the following scenario. Alice seeds the random-bit generator for the request stream with K and sends Bob a message. Upon receiving Alice's message, Bob seeds the random-bit generator for the reply stream with K, and sends a response to Alice. Again, assume that every request and response is equally likely.

3c. What can Eve deduce about the content of the messages? *Check the best answer possible of the ones listed (i.e., the most specific one).*

___Nothing.

___The content of the request, but not the reply.

X_The XOR of the request and the reply, but not the content of either the request or reply.

___The content of both the request and the reply.

Suppose C is the output of the random-bit generator when seeded with key K, M is Alice's message, and R is Bob's response. The encrypted request M is then M XOR C. The encrypted reply is R XOR C, since Bob seeds the random-bit generator with the same key K. (M XOR C) XOR (R XOR C) is M XOR R, so answer 3 is the correct answer.

Name: ______________________

4. *(28 points)* ByteStream Inc. extends its encrypted ByteStream products with an authentication protocol. The authentication protocol is based on a key server to simplify key management for users. ByteStream Inc. studies Kerberos, but decides it is too complicated. So, it designs a home-grown version of Kerberos that involves a single centralized ticket service CTS that grants tickets for services and session keys for communication. In ByteStream's system there is no separate ticket-granting service---CTS handles all the requests for tickets and keys.

CTS maintains a shared key for every principal. The protocol for a client C accessing service S is as follows. C sends a request to CTS, in the clear, containing C's name, S's name, and a nonce N (which is unique over time and unpredictable):

Message 1: C $\longrightarrow$ CTS : {C, S, N}

CTS sends a temporary session key K, S's name, C's nonce from Message 1, and a ticket for S to C, all encrypted using the shared key for C. The ticket consists of the session key, C's name, and CTS's current local time T, all encrypted using the shared key of S, K_S:

Message 2: CTS $\longrightarrow$ C : $\{K, S, N, Tick\}^{KC}$, where Tick = $\{K, C, T\}^{KS}$

C accepts Message 2 provided that S's name and N match up with what it sent to CTS in Message 1. C initiates communication with S by sending the ticket in the clear, together with an authenticator containing the new current time T' and a key K':

Message 3: C $\longrightarrow$ S : $\{\{T', K'\}^K$, Tick}

S decrypts Message 3, and accepts it if T (inside Tick) is no more than an hour old and T' is no more than five minutes old. Once it has accepted the message, S is prepared to communicate with C using key K' for encryption of all communications, until T becomes stale (older than an hour).

The following questions ask you to describe attacks if various mechanisms used in the ByteStream authentication protocol are left out. You might consider active and passive attacks such as eavesdropping, spoofing, and replay, among others.

4a. If the encryption with K_C of Message 2 were omitted what attacks are possible? Give one example scenario if one exists.

Multiple attacks are possible. A simple passive one is as follows:

1. Since Message 2 is sent in the clear, Eve can obtain key K. (Eve can obtain a copy of Message 2 by eavesdropping on CTS's response to C, or ask Lucy to mount an active attack by sending Message 1 to CTS and receiving CTS's response).

2. When Eve eavesdrops on Message 3, she can obtain K', since with key K she can decrypt $\{T', K'\}^K$.

3. Eve can eavesdrop on all subsequent message between C and S, because she can decrypt them using key K'.

Name: ______________________

4b. If T and T' are omitted from the protocol what attacks are possible? Give one example scenario if one exists.

Multiple attacks are possible. A simple active attack is as follows:

1. Lucy records all communication between C and S.

2. At some point later Lucy can replay the complete conversation. Since the server cannot detect replays, it will execute all the replayed requests.

Another possible active attack is:

1. Lucy records Messages 1 and 2.

2. Lucy goes to her supercomputer and finds K_C by attempting to decrypt Message 2 with every possible key until she gets the right S and N. (This will take a very long time...)

3. Lucy now knows K from Message 2 and can send Tick to S to authenticate to it as C. (Presumably, C will have changed his password since Lucy began her brute-force attack, so the K_C she discovered will no longer be useful.)

4c. If S is omitted from Message 2 what attacks are possible? Give one example scenario if one exists.

A simple active attack is spoofing C into believing it is talking to S while in reality C is talking with the attacker's server T:

1. Lucy runs some server T. She, CTS, and T know K_T

2. Lucy captures Message 1, replaces S with T, and sends it to CTS, ensuring that the return address is still C.

3. Lucy captures and redirects all messages that C tries to send to S so that they go instead to T, and so that responses from T to C appear to be coming from S.

THE END

Name: ______________________________