DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.033—Computer System Engineering | Handout #25—April 5, 1994

QUIZ #2—Solutions

This quiz contains five problems whose relative weights are indicated below. Please place each answer on the page with the question 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. The booklet may be separated for grading so that each problem can be evaluated by one grader. Be sure that you don't put part of the answer to a problem on the back of a sheet for another problem!

Beware: some parts of some problems are much harder than others. Read them all through first and attack them in the order that allows you to make the most progress. Write down any assumptions and 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. For problems that require a numerical answer, be sure to indicate the units of your answer.

Median score: 75

Problem 1. *(25 points)* Lucifer is determined to figure out Alice's password by a brute-force attack. From watching her log in he knows that her password is eight characters long and all lower-case letters. He sets out to try all possible combinations of eight lower-case letters.

a. Assuming he has to try about half the possibilities before he runs across the right one, one trial can be done in one machine cycle, and he has a 60 MHz computer available, about how long will the project take?

Since there are 26 lower-case letters, there are 26^8 passwords. It will take

$$T1 = \frac{26^8}{2 \cdot 60 \cdot 10^6} = \frac{(2.6)^8}{1.2} = 1740.22$$ seconds, or about half an hour.

b. How long will it take if Alice chooses an eight-character password that includes upper- and lower-case letters, numbers, and special characters? There are 78 characters available for her to choose from.

This time it will take $$T2 = \frac{78^8}{2 \cdot 60 \cdot 10^6} = \frac{(7.8)^8}{1.2} = 1.14 \cdot 10^7$$ sec, or about 132 days.

c. Suppose processors continue to get faster, improving by a factor of three every two years. How long will it be until Alice's new password can be cracked as easily as her old one?

The ratio of the numbers of passwords is $78^8/26^8 = 3^8$. Thus eight triplings of performance are required, and that will take 16 years.

An equivalent approach is to first calculate how fast a processor would be needed to crack the password in the same time, then calculate how many years it will take until processors are that fast.

A legalistic reading of the question yields the answer, "Never. No matter how fast the processor gets, it will always take 3^8 times as long to crack the new password as it takes to crack the old one."

Problem 2. *(24 points)* Louis Reasoner is fascinated with the discovery that some encryption algorithms are *commutative*.

For this problem we use the usual notation: A message M that is encrypted with key k is written $\{M\}^{k}$. A commutative scheme has the interesting property that for every message and every pair of keys k1 and k2, $\{\{M\}^{k1}\}^{k2} = \{\{M\}^{k2}\}^{k1}$. That is, you get the same result no matter which order two encryptions with different keys are done.

Louis has proposed that Alice, in San Francisco, and Bob, in Boston, use the following scheme for secure delivery of messages between their computers, which are connected via the Internet:

Alice chooses a never-before-used (nonce) encryption key, Ka, encrypts her message M, and sends the result, $\{M\}^{Ka}$, to Bob. Bob chooses another nonce encryption key, Kb, encrypts the already-encrypted message to produce $\{\{M\}^{Ka}\}^{Kb}$ and sends the doubly-encrypted result back to Alice. By commutativity, this message is identical to $\{\{M\}^{Kb}\}^{Ka}$, which is a message that Alice can decrypt with her nonce Ka. She does so, revealing $\{M\}^{Kb}$. This result she sends back to Bob, who can now decrypt it with his nonce Kb to reveal M.

The appealing thing about this scheme is that Alice and Bob did not have to agree on a secret key in advance. Louis calls this the "No-Prior-Agreement" protocol.

a. Is it possible for a passive intruder (that is, one who just listens to the encrypted messages) to discover M? If so, describe how. If not, explain why not.

No. Assuming the encryption algorithm itself is reasonably good, all that a passive intruder will see is packets encrypted under unknown, never-before-used keys.

b. Is it possible for an active intruder (that is, one who can also insert, delete, or replay messages) to discover M? If so, describe how. If not, explain why not.

Yes. Lucifer need merely trap the first packet $\{M\}^{Ka}$ and send it back to Alice; according to the protocol she will decrypt it and send the plaintext to Bob, which Lucifer can also intercept. If Lucifer wants to ensure that Alice doesn't become suspicious, he can encipher the packet in a key he knows before sending it back to Alice. He can similarly mimic the protocol with Bob to ensure that Bob ends up with M and is no wiser about Lucifer's intervention. The problem with this protocol is that it provides no authentication whatever.

Problem 3. *(10 points)* One way of speeding up a name resolution system is to implement a cache that remembers recently looked-up {name, object} pairs. *Synonyms* are multiple names for the same object. What problems do synonyms pose for cache designers?

The trouble with synonyms is that if *name1* and *name2* refer to the same object, *{name1, object}* is in the cache, and someone refers to *name2*, we are likely to not realize that we already have *object* in the cache, and we will look it up again and add it to the cache as the pair *{name2, object}*. We have wasted time, we have used up valuable cache space, and we have set ourselves up for a disaster if the system permits modifying objects by name. A modification to the copy of *object* associated with *name1* should be propagated to the copy of *object* associated with *name2*.

One can invent various mechanisms to allow synonyms in caches. For example, if each object has a unique ID one could change the cache to hold {name, object, UID} triples. Then, whenever someone modifies an object by name, it is possible to search the cache for any other entries that have the same value for UID in their third field and either modify or invalidate those copies.

Problem 4. *(15 points)* Alyssa is trying to organize her notes on virtual memory systems, and it occurred to her that virtual memory systems can usefully be analyzed as naming systems. She has gone through her 6.033 notes and made a numbered list of some technical terms from the lectures on naming systems; that list is on the right, below. She then listed some mechanisms found in virtual memory systems on the left. But she isn't sure which naming concept goes with which mechanism. Help Alyssa out by placing in each blank on the left the number of the term on the right that best applies to this mechanism. Hint: no number needs to be used twice.

	1. Search path
_6_ a. page map	2. Naming network
_5_ b. virtual address	3. Closure
_4_ c. physical address	4. Object
_7_ d. a TLB entry	5. Name
_3_ e. the processor register that points to the page map	6. Context
	7. None of the above

Note that although a TLB entry is involved in a kind of search, it is not itself a "search path", which is a list of places to search.

Problem 5. *(25 points)* The network level of the Internet names hosts with 4-byte *host addresses*. These names are assigned regionally, with all hosts in one major region receiving host addresses for which the first byte is the same. A region is divided into sub-regions in which all host addresses have the same value for the second byte, and so on. This hierarchical naming scheme is used by the routing system to reduce the size of the tables it must maintain.

a. Explain briefly how hierarchical naming of host addresses might reduce the size of routing tables.

Hierarchical naming creates structural regularity that allows us to compress the size of the routing table. From a given point in the network, it is normally the case that the appropriate "next hop" in the network is the same for many destination addresses. When several such destinations happen to be in the same sub-sub-region, sub-region, or region, we can save space by recording the common "next hop" information once for all those addresses. The hierarchical naming makes it likely (but not certain--see below) that we can cleanly group up entire levels of the hierarchy and store a single route entry suitable to large chunks of the network without loss of information.

Note that having hierarchically organized names does not require that the routers themselves be organized hierarchically for this space reduction to be effective; it is sufficient if just the names are hierarchical. Thus there can be direct links between any two routers, even if they are from different regions, and we do not need to require that packets destined to a different region must go up three levels and then down three levels. If we did assume that the network's physical layout were hierarchical, then we would be guaranteed to find the opportunity for space compression that we currently can only utilize when it happens to be available. Another way of thinking about this difference is that we are taking advantage of the hierarchical shape of the names to do a lossless compression. If the network were organized hierarchically, the compression would be greater. If the network is not hierarchical, but we compress as though it were, then we would have a lossy compression scheme, for which the losses would show up as not knowing the best available routes. Curiously, everything would still work "correctly," except that the performance would probably be non-optimal.

b. Ben Bitdiddle is thinking about designing a system for wireless attachment of portable computers to the Internet. He has decided that the basic architecture should consist of a number of fixed base stations, each of which will communicate by radio with those portable computers in its area. He plans to make each base station a sub-sub-region, using the third byte of the Internet address to identify it. Alyssa points out that this plan limits the number of portables that can simultaneously work with one base station. The base station uses up one of the host addresses for its sub-sub-region. How many portables can work with one base station?

A byte being eight bits, there are 256 addresses available. One of these is used up by the sub-sub-region base station, leaving 255 for portables. (Or, if the network reserves one address for broadcast, 254.)

c. Ben proposes not to permanently assign host addresses to portable computers. Instead, when a portable is switched on, it will ask some base station to dynamically assign a host address for it to use. When dynamic address assignment is used, what else has to happen in order for other computers to initiate communication with a portable?

In order for someone to initiate communication with the portable, some persistent name needs to be associated with the dynamically assigned host address. For example, the name service needs to be updated.

d. Ben wants his design to work with the portable computer mounted on the handlebars of his bicycle. The base stations he will use were designed for

cellular phones, so they have the ability at the physical transport level to transfer (or "handoff") a conversation from one base station to another as the signal strength changes. But in Ben's design a handoff means that the host address of the portable must change. Ben wants to be able to maintain connections through handoffs, so that he can transfer long files without disruption while cycling across town. His first idea is to modify the File Transfer Protocol to allow handoff. Whether or not he figures out how to do the modification, what is the main defect in this proposal?

Ignoring safety problems that may arise from Ben watching the computer display instead of traffic, there is a major abstraction violation here. Ben is trying to solve a network layer problem by tinkering with the end-to-end layer. As a result, he is going to have to track down every implementation of FTP he intends to talk to and modify it, and he is also going to have to modify all other end-to-end protocols he wants to use.

e. ***Briefly*** suggest a way of accomplishing handoff entirely within the network layer.

This one is challenging. One way to start is to have the portable, when it realizes it has been handed off, send a message via the network to the previous base station, asking it to forward packets to the new address. All outgoing packets from the portable should continue to claim to come from the old address, to avoid confusing correspondents. And when the portable does disconnect it would be polite of it to send another message to the earlier base station suggesting that it can discard the forwarding pointer and reassign the host address.

An alternate approach might be to create pseudo-sub-sub-regions in which portables can have permanently assigned addresses, and design a special forwarder that receives all packets addressed to a given pseudo-sub-sub-region. When a portable receives a dynamic host address assignment, its first project is to send a message to its pseudo-sub-sub-region forwarder telling it its current address.

We got into this problem to begin with by condensing routing tables to take advantage of the hierarchical nature of the addresses (and, as it happens, to reduce routing table update traffic in the network). If we reversed that decision, then a host address could be physically anywhere in the network, and it would not be necessary to change host addresses when there is a physical level handoff. So in the end we see that our hierarchical routing strategy has actually muddied the distinction between the physical and network layers.

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.033—Computer System Engineering | Handout 28—April 25, 1995

Quiz #2 Solutions

169 students took the quiz.
Median score = 69
Mean score = 68.1
Std. deviation = 12.8

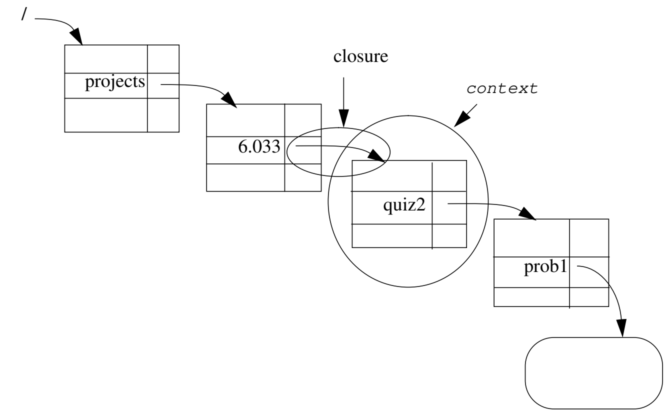
Score	Frequency
31-35	** (2)
36-40	* (1)
41-45	******* (7)
46-50	****** (6)
51-55	*************** (15)
56-60	************ (12)
61-65	********************* (21)
66-70	*************************** (27)
71-75	**************************** (28)
76-80	************************** (26)
81-85	********** (10)
86-90	********* (9)
91-95	***** (5)
96-100	

For Official Use Only

Problem	Your Score	Maximum Score
1		10
2		25
3		25
4		15
5		25
Total		100

Name: ________________________________

1. *(10 points)* Consider the UNIX file system naming hierarchy illustrated below:



You have been handed the following path name:

closure → /projects/6.033/quiz2/prob1

and you are about to resolve the third component of that path name, the name “quiz2”.

1a. Circle the closure, and write the word “closure” next to the thing you have circled.

The closure identifies the context that one should use to resolve “quiz2”. There are two equally valid things in the picture that identify that context to the name interpreter:

a) the pathname “/projects/6.033/”
b) the inode number (shown as a pointer) that is associated with the name “6.033” in the superior directory.

1b. Circle the context, and write the word “context” next to the thing you have circled.

A context is a set of bindings of names to objects. In this problem the context for a single component of a path name is a single UNIX directory. The context that one should use is the one identified by the closure, the third box from the left in the picture.

Name: ________________________________

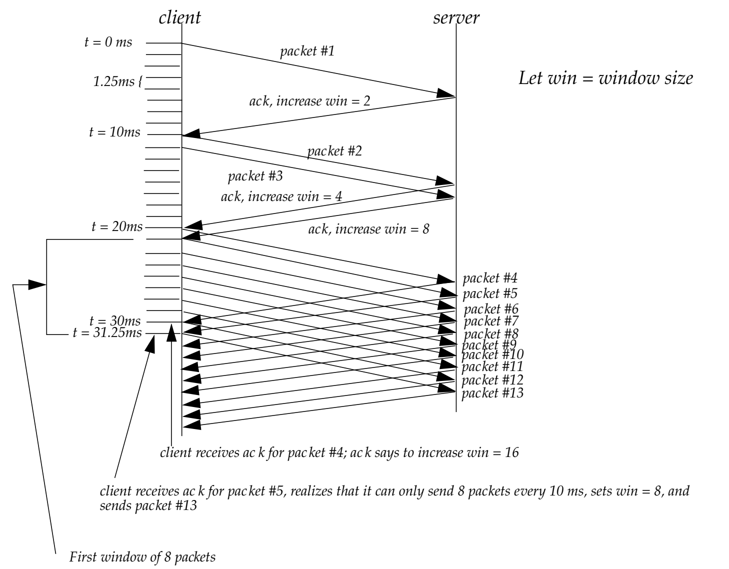
2. *(25 points)* To obtain high throughput, communication applications pipeline messages: after sending the first packet of a message, they immediately send the next one without waiting for an acknowledgement of the first packet. A key question is how to establish how many packets to pipeline between a request and its acknowledgement (i.e., what is the window size).

2a. *(5 points)* Assume the client and the server are connected by an 800,000 byte/sec link. Also assume that the client and the server produce and consume packets at the same rate. Using the server's acknowledgements, the client measures the round-trip between itself and the server to be 10 milliseconds. If the client is sending 1000-byte packets, what is the smallest window size that allows the client to achieve 800,000 byte/sec throughput?

In order to achieve 800,000 byte/sec throughput, the client must send 800 packets per second, or 8 packets per 10 ms. If the round trip time is 10 ms, then the client must send 8 packets before it receives an acknowledgement for any of them, making the window size 8 packets.

Alternatively, on an 800,000 byte/sec link the client takes 1.25 msec to send one 1000 byte packet. Thus, the client can send 8 packets in 10 msec and saturate the link bandwidth.

2b. *(5 points)* A popular scheme for establishing the window size in the Internet is *slow start.* In a simple version of slow start, a client starts with a small window size and increases the window size. Assume that the client starts with a window size of one packet. Also assume that for every packet, the server responds with an acknowledgement telling the client to double the window size. How long would it take for the client to know it has reached the maximum throughput?



The client knows it has reached maximum throughput after 31.25 ms. See above figure.

2c. *(5 points)* Another scheme for establishing the window size is *fast start.* In fast start, a client starts by assuming a default large window size (say 8 packets) and sends the server a burst of that size. On the server's response, the window is adjusted to the size that is supplied by the server. If a client starts with a window size of 8, how long does it take for the client to know it has achieved the maximum throughput?

In this case, the client is guessing a window size that is exactly the maximum. The client will know that this guess is the maximum because he will spend 10 ms transmitting 8 packets and receive the acknowledgement for the first packet right at the end of this 10 ms. Thus, the client knows that he will never be able to transmit more than 8 packets in 10 ms, and so he knows that he has achieved the maximum throughput (8 packets per 10 ms, or 800,000 bytes/sec) after only 10 ms.

2d. *(10 points)* In the circumstances described above, fast start seems to be preferable. In what circumstances would it be preferable to use slow start over fast start? You may assume that one of the back-pressure techniques used in the Internet is dropping packets.

One situation where slow start might be preferable is when there are multiple clients but only a single server and the server applies back pressure by dropping packets that it can't handle. With fast start, the clients might all send large bursts of packets, and the server may drop many of them. This will cause the clients to timeout and resend the lost packets. (If the clients do not slow down after this happens then this cycle of dropping packets, timing out, and resending may occur repeatedly.) It will take a certain amount of time to transmit, lose packets, timeout, and resend before the optimal window size is found. It may very well be that slow start would find the optimal window size much more quickly and hence have better performance.

There are other reasons why slow start might be preferable. In general, slow start will cause fewer packets to be dropped than fast start. (However, there is still no guarantee that slow start won't cause any packets to be dropped!) This reason in itself is not sufficient, though; there has to be a reason why dropping packets is bad. Dropping packets can be bad because there is no reason to think that the dropped packets are always from the new client's connection; a fast start may cause the network or server to drop packets from existing connections, forcing those connections to resize their windows.

Name: ______________________________

3. *(25 points)* Secure Inc. is developing an improved version of AFS, Secure AFS (SAFS), which automatically encrypts files to guarantee better privacy of information. When a request to store a file arrives, SAFS enciphers the file using the client's key. On arrival of a request to read a file, SAFS looks up the client key, deciphers the file, and sends the file back to the client. SAFS keeps for each client a separate key to encipher and decipher files.

3a. *(5 points)* The designers of Secure Inc. are wondering how long it would take to crack an enciphered file that is encrypted using RSA with 512-bit key, which is what is typically used today. To crack an RSA-enciphered file one has to factor the key. The designers know from the literature that factoring a 100 decimal digit number takes about 1 month using idle cycles from 300 3-MIPS SUN-3 workstations. It is estimated that factoring an additional 3 decimal digits roughly doubles the computation time needed. How many 3-MIPS computers are needed to factor a 155 decimal digit number (which corresponds to about 512 bits) in one month?

100-digit number ==> 300 workstations
Difficulty doubles with every three digits.

155-digit number ==> 55/3 doublings ==> $300 * 2^{55/3}$ *workstations*
This is about 99 million workstations.

3b. *(5 points)* If processors are doubling computation performance per year, how many computers would it take to factor a 512-bit key in one month in the year 2000?

Performance doubles every year.
2000 - 1995 = 5 years ==> 5 doublings
So, dividing the answer from part (a) by 2^5*, we now need*

$300 * 2^{40/3}$ *workstations, i.e., about 3 million workstations.*

3c. *(5 points)* Let's assume that enciphering and deciphering can be done at 250 Kbyte/sec. How much would the throughput be reduced for reading files stored by SAFS, if the current maximum throughput without enciphering/deciphering is 800 Kbyte/sec? (You may assume that enciphering/deciphering cannot be pipelined with sending/receiving.)

Time per unit data = time to decrypt + time to send
= 1/250 sec/KB + 1/800 sec/KB = 21/4000 sec/KB
The new throughput is the inverse of this number, i.e., data per unit time:
4000/21 KB/sec = 190.4 KB/sec
The problem asked for the <u>reduction</u> in throughput, which is
Old throughput - new throughput = 800 KB/sec - 190.4 KB/sec = 609.6 KB/sec
This is 609.6/800 = 76% of 800 KB/sec or a reduction <u>of</u> (800-609.6)/800 = 24%.

Name: ______________________________

3d. *(10 points)* Secure Inc. is also considering adding automatic compression of files to SAFS. Compression reduces redundancy of information in a file so that the file takes less disk space. Should they first compress files, then encipher them, or should they first encipher files and then compress them? Explain your answer briefly.

It is better to compress a file first, then encipher it. There are several reasons for this. Among them are the following, ordered strongest to weakest in our estimation.

1. Effective compression depends on eliminating redundancy (e.g., patterns of character frequency) in the file being compressed. However, effective encryption eliminates patterns in order to disguise content. Therefore, if the file were enciphered first, little or no compression would be achieved.

2. Compression reduces the size of the file, whereas encryption (we assume) does not. Assuming both operations take time proportional to the length of the source being operated on, the greatest total efficiency is achieved by compressing the file to create a smaller source for the encryption operation.

3. If the file is compressed before it is encrypted, it is harder for an attacker guessing a key to realize he has correctly decrypted the file, as the decrypted file still looks pretty random. Even if he knows the decompression algorithm to apply, it still slightly increases the work factor for cracking the encryption.

Name: ____________________________________

4. *(15 points)* Baybank is struggling to convince itself of the validity of a message it just received, and has asked your help in what to do next. So far, they know the following two facts to be true:

Louis says (Ben says (Transfer $1,000,000 to Alyssa))

Jim speaks for Ben

According to Ben's account representative, Ben's account has enough money for such a transaction, so if they can convince themselves that Ben really said that, they will do the transfer.

4a. *(5 points)* Which of the following things should they attempt to establish the truth of?

a. Louis speaks for Jim

b. Ben speaks for Louis

c. Ben says (Jim speaks for Louis)

4b. *(10 points)* Why?

Choice (a) allows Baybank to establish the following, two-step line of reasoning:

1. Jim speaks for Ben [given]
Louis speaks for Jim [choice a]

Reasoning rule #2 (chaining of delegation) says that
(A speaks for B) and (B speaks for C) implies (A speaks for C)

Applying this rule tells Baybank that (Louis speaks for Ben)

2. Louis says (Ben says (Transfer $1,000,000 to Alyssa)) [given]
Louis speaks for Ben [established in step 1]

Reasoning rule #1 (delegation of authority) says that
(A says (B says X)) and (A speaks for B) implies (B says X)

so if step 1 is sound, Baybank can be sure that

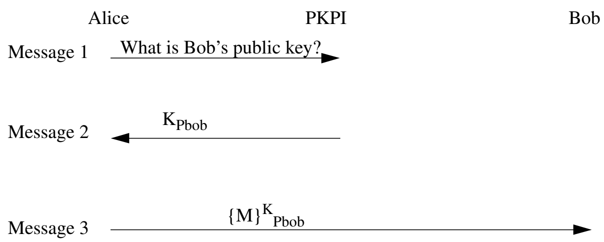
Ben says (Transfer $1,000,000 to Alyssa)

which is what Baybank needs to know before it will perform the transfer.
Thus Baybank should attempt to establish the truth of choice (a).

Name: ____________________________________

5. *(25 points)* Ben Bitdiddle and Louis Reasoner have founded a startup company, named Public Key Publication, Inc. (PKPI), whose business is distributing public keys. Their idea is that people who have a key-pair for use with a public-key system need a way of letting other people know the public key of their key-pair. Ben and Louis are not interested in creating keys, but just in acting as a public key distributor.

Ben and Louis have designed the following protocol, in which Alice sends a private message to Bob. They need your help in debugging the protocol. K_{Pxyz} is the public key of principal xyz.



Messages 1 and 2 constitute the PKPI protocol; message 3 is the beginning of Alice's protocol with Bob and is not under the control of PKPI; message 3 is shown here only to place the PKPI protocol in context.

5a. *(5 points)* Louis believes that Eve, the passive eavesdropper, will find that she cannot learn anything by overhearing the PKPI protocol in use. Give an argument that supports Louis' position, or an example demonstrating that Louis is mistaken.

Louis is almost correct. Neither Message 1 nor Message 2 contains any secrets, so passive eavesdropping won't reveal anything. Eve may, however learn something from traffic analysis: that Alice has asked for Bob's public key. If Alice had hoped to keep that fact secret, this was not a very good protocol to use.

5b. *(10 points)* Louis originally hoped that Lucifer, the active attacker, wouldn't be able to cause any problems, either, but since hearing the 6.033 lecture on the topic he is not sure. Give an example of an active attack that demonstrates that Louis needs to revise the PKPI protocol to protect against Lucifer.

Lucifer can actively attack the protocol in several ways:
1) He can intercept and replace Message 1 with, "What is Lucifer's public key?" which will cause Message 3 to be encoded with Lucifer's public key. He will then be able to decrypt Message 3.
2) He can intercept and replace Message 2 with Lucifer's public key instead of Bob's public key, which will cause Message 3 to be encoded with Lucifer's public key. He will then be able to decrypt Message 3.

5c. *(10 points)* Ben suggests that the protocol could be improved by changing Message 2. What should Message 2 contain, and how should it be enciphered, so that Alice can be confident that no one but Bob can decipher message 3?

The protocol lacks two things:
1) Authentication: Alice needs proof that the message she is receiving is from PKPI and not from Lucifer. This can be accomplished by encoding Message 2 in the secret key of PKPI, K_{Spki}*, which anyone can decrypt but only PKPI could have constructed.*
2) Connection with Bob's key: Alice needs proof that the key she is receiving in Message 2 is Bob's, not Lucifer's. This proof can be accomplished by including the name "Bob" inside the now-encrypted Message 2: $\{Bob, K_{Pbob}\}^{K_{Spkpi}}$.

Note that if public keys never change, replays aren't of concern--even if Lucifer intercepts and replaces Message 2 with a month-old copy of Message 2, as long as that stale message was encrypted by PKPI and contains Bob's name it is OK.

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

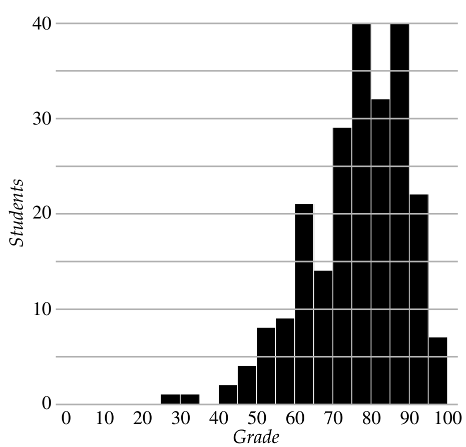
6.033—Computer System Engineering Handout 28—April 23, 1996

Quiz 2 Solutions

These solutions contain four problems; we have written our answers in the space provided.

One note: Our solutions are not brief. We have aimed for clarity and completeness (listing almost every possible answer) rather than brevity. Your answers may have been, and should have been, much shorter. We expect only one answer from you, while you should expect all the correct answers from us.

Grade distribution



Name: *Mrs. B. O. T. Envelope-Calculation*

1. Ben Bitdiddle has been thinking about RPC. He remembers that one of the problems with RPC is the difficulty of passing pointers: since pointers are really just addresses, if the server dereferences a client pointer, it'll get some value from *its* address space, not what the client intended. Ben hopes to get some insight by applying his new-found 6.033 naming knowledge to the problem.

1a. *(15 points)* Match up each concept on the left with the corresponding naming concept on the right. You won't use some of the naming concepts.

Here is one consistent interpretation:

(6) address space	(1) implicit context
(7) pointer	(2) object
(2) memory value	(3) multiple contexts
(4) pair of address space and pointer	(4) closure
(1) dereferencing a pointer in the current address space	(5) search path
	(6) context
	(7) name

Alternatively, an address space may be interpreted as a closure. To see this, consider a pointer as a name, and the page table structure as a context. Then the address space may act as a mechanism binding names to their context: a closure.

Inspired by his naming insights, Ben decides to pass closures instead of bare pointers in his RPC system. Louis Reasoner, excited by Ben's discovery, decides to change *all* end-to-end protocols used on MIT's networks along the same lines.

1b. *(5 points)* Argue for or against Louis's changes, using 6.033 arguments.

We dislike Louis's changes. Not all applications need to pass pointers, so requiring all end-to-end protocols to support this service will incur needless performance penalties. Instead, only those applications that require safe pointer passing should provide it as part of their end-to-end protocol. This is an example of an end-to-end argument.

2. In a mobile computing environment, locating a user in the face of user movement is an important problem. Consider a system in which each user is assigned a permanent home location, which can be found simply by looking the user up in a name service.

Alyssa P. Protocol-Hacker is designing an end-to-end protocol for locating these mobile hosts. Her first protocol is very simple: every time a user moves, store a forwarding pointer at the old host, pointing to the new host. This creates a chain of forwarding pointers with the permanent home location at the beginning and the mobile host at the end. Messages meant for the mobile host are sent to the home location, which forwards them along the chain until they reach the mobile host itself. (The chain is truncated when a mobile host returns to a previously visited location.)

Alyssa notices that performance generally gets worse each time she moves her mobile host! "This must be because of the long chains of forwarding pointers," she muses.

Help Alyssa design a more efficient protocol which avoids long chains. Assume that the end-to-end protocol you design is built on top of a network protocol which can reorder messages, but doesn't lose or duplicate them.

2a. *(8 points)* Alyssa's first try works like this. Each time a mobile host moves, it sends a message to its home location indicating its new location. The home location maintains a pointer to the new location. With this protocol, there are no chains at all. Nodes other than the home node do not maintain forwarding information.

When this protocol is implemented, Alyssa notices that messages regularly get lost when she moves from one location to another. Explain briefly why or give an example.

If a message is in transit to mobile host A at the time when A moves from location X to location Y, that message will arrive at X and find that A has left. Since there is no information at X indicating where A went, the message will be lost. Similarly, any message that the home node forwards after A has moved but before A's rerouting message gets to home will go to location X and be lost. All messages that home forwards to A from about one network transit time before the move until about one network transit time after the move will probably be lost.

A more disastrous scenario, though less likely and therefore probably not the cause of messages "regularly" getting lost, happens when a mobile host A moves from X to Y, and then on to a new location Z. The rerouting message that A sends from Y may be delayed in transit long enough that the rerouting message it sends from Z gets home first. In that case, when the rerouting message sent from Y finally arrives, home will forward all future messages to the wrong place; A will be completely cut off, at least until its next move. The briefer the visit of A to Y, the greater the chance of this scenario happenening.

2b. *(8 points)* Alyssa is disappointed with her first attempt, and decides to start from scratch. In her new scheme, no forwarding pointers are maintained anywhere, not even at the home node. Say a message destined for a mobile host A arrives at a node N. If N can directly communicate with A, then N sends the message to A, and we're done. Otherwise, N broadcasts a search request for A to all the other fixed nodes in the network. If A is near a different fixed node N', then N' responds to the search request. On receiving this response, N forwards the message for A to N'.

Will messages get lost with this protocol, even if A moves before the message gets to N'? Briefly explain your answer.

No and Yes *(a typical 6.033 answer!).*

No, messages will not get lost: *Suppose mobile host A moves away from N' just after N' responds to the search request N sent. N will forward the original message to N', but when it arrives, N' will follow the standard procedure: If it is in contact with A it will pass the message along to A; however, since it is not, it* broadcasts another search request. *Assuming A is always in contact with some node, the original message will thus follow A around. If A ever rests long enough for the message to catch up with it, it will then be delivered.*

Yes, messages will get lost: *Search broadcasts have a transit time, and any given broadcast doesn't get to all nodes at exactly the same time. Suppose node N broadcasts a search request, and while the broadcast is in transit, A moves from node X to node Y. The following time sequence creates a problem:*

1. *N broadcasts "Where is A?";*
2. *Y receives broadcast, ignores it becase A is at X;*
3. *A moves from X to Y;*
4. *X receives broadcast, ignores it because A is now at Y;*
5. *N concludes from the lack of response that A has retired from active service and trashes the original message.*

And the following briefer example of the above scenario, in which Y is actually N, can lose messages even if broadcasts are delivered everywhere simultaneously. (We assume that mobile hosts don't actively notify locations when they arrive; instead, the locations try to locate them when messages for them are received.)

1. *N broadcasts "Where is A?";*
2. *A moves from X to N;*
3. *X receives broadcast, ignores it because A is now at N;*
4. *N concludes from the lack of response that A has retired from active service and trashes the original message.*

2c. *(8 points)* Unfortunately the network doesn't support broadcast messages efficiently, so Alyssa goes back to the keyboard and tries again. Her third protocol works like this. Each time a mobile host moves, say from N to N', a forwarding pointer is stored at N pointing to N'. Every so often, the mobile host sends a message to its permanent home node with its current location. Then, the home node propagates a message down the forwarding chain, asking the intermediate nodes to delete their forwarding state. Can Alyssa ever lose messages with this protocol? Explain briefly. (Hint: think about the properties of the underlying network.)

There are lots of ways to lose messages with this protocol. Here are three of them (numbered for future reference).

P1. Since the underlying network can reorder messages, the "delete forwarding pointers" message may overtake an ordinary message already in transit along the forwarding chain. Then, when the ordinary message gets to its next forwarding point it will find that the forwarding information has already been deleted.

P2. If mobile host A creates a long forwarding chain, sends an "I'm over here at node X" message home, then immediately moves to a location Y that is earlier in the chain and then on to somewhere else, the "delete forwarding pointers" message will come along and delete the current pointer in Y, leaving A unable to receive anything.

P3. A version of the disastrous scenario of part 2a also applies. A moves from X to Y, sends a message home, then moves to Z and sends another message home. The second message home gets there first, causing the chain out to Y (including the forwarding pointer from Y to Z) to be deleted. Then the first message gets home, telling home to send everything to Y.

2d. *(8 points)* What additional steps can the home node take to ensure that the scheme in question c never loses messages? Describe briefly.

Here are a number of different solutions.

S1. Since the latest "I'm here" message from mobile host A to its home short-circuits any forwarding pointers in other nodes, the home node could just omit sending the message that deletes old forwarding pointers. That would solve all three problems of part 2c, which is good enough. However, it would leave dead forwarding information lying around, using up space, with no way to know when it can be reclaimed (call that problem P4).

S2. If the home node knew the maximum transit time of the network, it could delay the sending of the "delete forwarding pointers" message for that length of time. This approach would take care of problem P1 and problem P4 and, with a clever home node, might help reduce the likelihood of problem P3. Unfortunately, it will increase the likelihood of problem P2.

S3. The home node could include sequence numbers in all forwarded packets. Then, when it gets a rerouting message from a mobile host, it can immediately begin forwarding new packets directly to the new location, and it can also send a message to the mobile host telling it the sequence number of the last packet sent via the old route, and asking for an explicit acknowledge when all packets up to the sequence number have arrived. (Note that a timestamp isn't good enough here; we need a way of being sure that **all** *intervening packets have been delivered.) When that acknowledgement returns to home, send the "delete forwarding pointers" message down the old chain. Problems P1 and P4 are solved. Problems P2 and P3 persist.*

S4. Instead of sending a "delete forwarding pointers" message, the home node could send a "change forwarding pointers" message, alerting everyone along the old forwarding chain to adjust their forwarding pointer to A's new location. That would eliminate problems P1 and P2, but problems P3 and P4 persist.

S5. The home node could include its own address in every forwarded message, and all other forwarders could follow a new rule: if a message is undeliverable as originally addressed, send it back to the home node. The home node can then rescue and reroute any messages that would have been lost under the other methods. In addition, when it gets a rerouting message from a mobile node, it can immediately send a "delete forwarding information" request down the old route, confident that any dallying packets will be returned to it for rerouting. This approach takes care of problems P1 and P4, and some instances of P2. It also means that when problem P3 (or the bad instances of P2) arise, any packets that the home tries to forward to the mobile will loop indefinitely between home and the last place home thought the mobile was connected; if the mobile ever checks in again, these packets will eventually be delivered. Whether this is a satisfactory state of affairs is debatable.

S6. The mobile host could count the number of rerouting messages it sends home, sending a sequence number with each message. These sequence numbers impose an order on rerouting messages, so problem P3 is solved. The last sequence number the mobile sent is attached to each forwarding pointer it creates. When the home node receives a rerouting message with sequence number S, where S is greater than the last sequence number home saw from that mobile host, it sends out a message "delete forwarding information with sequence number smaller than S." This solves problem P2. Combine this method with solution S5 (undeliverable messages are sent back to the home node) and we have solved all four problems, P1 through P4!

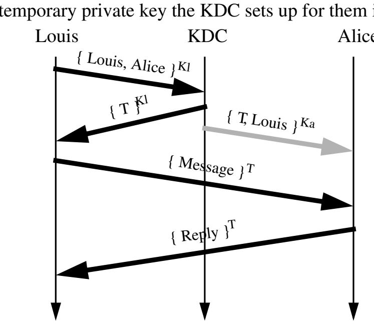
Note that none of the suggested solutions involve adding an acknowledge/timeout/retransmit scheme to the forwarders. That approach would have the side effect of producing duplicates in a network that is certified never to produce duplicates, so it would require modifying the end-to-end protocol implementations to suppress these duplicates.

3. Louis B. Unreasonable has decided that the NSA has gotten their hands on PGP and probably made it weak or insecure. He therefore wants to write his own safer and more secure encryption system. Before doing so, he first wants to understand the most popular schemes of today, RSA and DES. RSA is a public-key system where encryption and decryption each take 1,000 ms per block (a block is 0.1Kbyte or less). DES is a private-key system with a single key; encrypting or decrypting with DES only costs 10 ms per block. (Only consider passive attackers for this problem.)

3a. *(6 points)* How long does it take to encrypt a file that is 1,000Kbytes in size, assuming you are using RSA? How long does it take to encrypt that same file using DES?

RSA: 1,000Kbytes / 0.1Kbytes/block ¥ 1,000ms/block = 10 seconds.
DES: 1,000Kbytes / 0.1Kbytes/block ¥ 10ms/block = 0.1 seconds.

3b. *(6 points)* Louis was planning on using public-key encryption, but the costs you calculated for him have changed his mind. Now, his system will have central key distribution centers (KDCs) using only private-key encryption. Here's how the protocol works when Louis sends a message to Alice. (Louis's private key is Kl; Alice's private key is Ka; and the temporary private key the KDC sets up for them is T.)



How long will it take for this exchange to complete, that is, from the time Louis starts encrypting the first message to the KDC to the time Louis is done decrypting Alice's reply? *Only consider encryption times, decryption times, and network latency.* Assume the network latency is 2,000 ms per message. Assume also that each message fits in one block.

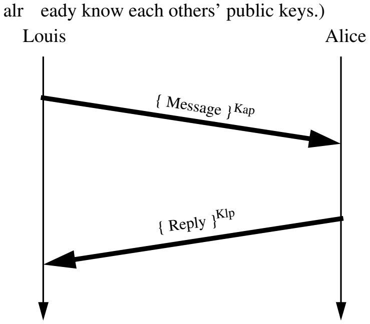
It takes 8080ms.

The critical message path consists of the black messages on the diagram above. The message we've greyed out, the KDC's response to Alice { T, Louis }^{Ka}, is **not** *part of the critical path. As the diagram suggests, the KDC sends the temporary key to Louis before it sends the temporary key to Alice; Louis can begin working on his message before Alice receives hers.*

1. Encryption: Louis encrypts { Louis, Alice } with Kl	*10ms*
(All encryption and decryption in this example use the private-key scheme DES. Also note that all messages fit in one block.)	
2. Latency: Louis sends { Louis, Alice }^{Kl} to KDC	*2000ms*
3. Decryption: KDC decrypts { Louis, Alice }^{Kl}	*10ms*
4. Encryption: KDC encrypts { T } with Kl	*10ms*
5. Latency: KDC sends { T }^{Kl} to Louis	*2000ms*
6. Decryption: Louis decrypts { T }^{Kl}	*10ms*
7. Encryption: Louis encrypts { Message } with T	*10ms*
8. Latency: Louis sends { Message }^{T} to Alice	*2000ms*
9. Decryption: Alice decrypts { Message }^{T}	*10ms*
10. Encryption: Alice encrypts { Reply } with T	*10ms*
11. Latency: Alice sends { Reply }^{T} to Louis	*2000ms*
12. Decryption: Louis decrypts { Reply }^{T}	*10ms*
Total	*8080ms*

Another acceptable answer was 8090ms: the KDC might encrypt T with both keys before sending either message. This would add a step 4.5 to the list: KDC encrypts { T, Louis } with Ka.

3c. *(6 points)* Alyssa P. Hacker thinks that Louis's reasoning is all wrong, particularly when messages and replies are small. She proposes a public-key encryption system; with her protocol, the same message exchange would look like this. (She assumes that Louis and Alice already know each others' public keys.)



How long will it take for this message exchange to complete with Alyssa's protocol (encryption times, decryption times, and network latencies only)? Again, the message and reply each fit in a block.

With this protocol, the message exchange takes 8000ms: less than Louis's optimized protocol, even though expensive public-key encryption is used!

1. Encryption: Louis encrypts { Message } with Kap 1000ms
(All encryption and decryption in this example uses the public-key scheme, RSA.)
2. Latency: Louis sends { Message }^{Kap} to Alice 2000ms
3. Decryption: Alice decrypts { Message }^{Kap} 1000ms
4. Encryption: Alice encrypts { Reply } with Klp 1000ms
5. Latency: Alice sends { Reply }^{Klp} to Louis 2000ms
6. Decryption: Louis decrypts { Reply }^{Klp} 1000ms
Total 8000ms

3d. *(6 points)* What guarantee does Louis's protocol provide for Alice that Alyssa's protocol doesn't? Assume that Louis keeps T secret.

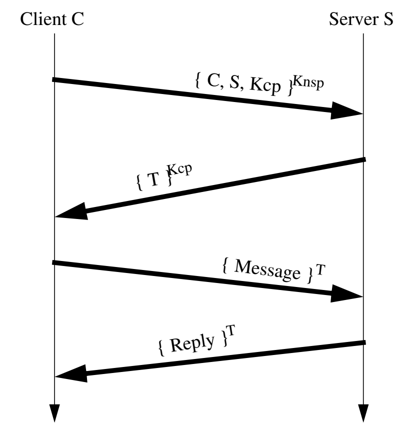
There are two good answers.

1. Louis's protocol provides Alice with **authentication***: the message { Message }^{T} is guaranteed to come from Louis, since the KDC only sent T to Alice and Louis, and Louis didn't give it away. (We also assume the KDC is secure.) Since there are no active attackers, the message { Message }^{T} could not be a replay.*

2. Since there are no active attackers, authentication is not important! No one will ever pretend to be Louis anyway. Therefore, Louis's protocol provides **no** *guarantees whch Alyssa's doesn't. (It may be easier to determine the source of a message with Louis's protocol, since the KDC tells you explicitly, but this is not really a guarantee.)*

4. A startup company with an insanely high stock price, the Notsafe Communications Corporation, is designing a protocol for safe and private transactions over the Internet. Rather than rely on centralized security servers like Kerberos, the Notsafe designers are using public-key cryptography to implement their protocol. However, only the initial handshake uses public-key, since the protocol will be used for high-volume data transfers, where public-key cryptography is just too expensive. The handshake will set up a private key for communication between the parties, which then can continue for as much as a day or more.

The Notsafe protocol has a well-known public key, K_{nsp}, which is used to establish connections. In addition, servers who want to talk the Notsafe protocol must use a Notsafe server program, which includes the secret key, K_{nss}. The protocol is graphically depicted below.



In the first message, the client sends a handshake including C's public key, encrypted with K_{nsp}: {C, S, K_{cp}}^{Knsp}. The server decrypts this with K_{nss} and sends a response back to the client including a 32-bit private key T for the rest of the communication, encrypted with K_{cp}. Then, the client decrypts T with K_{cs} and sends messages and receives replies encrypted with T.

4a. *(2 points)* Notsafe's marketing department declares that "the risk of an intruder reading your sensitive data with Notsafe's products is less than one in one trillion." Mrs. Bach O. T. Envelope-Calculation immediately exclaims, "Poppycock!" Given the key that is being used, what might she have been thinking?

The private key T has only 32 bits, so there are exactly 2^{32} possible values for T. Therefore, an intruder making a random guess at T has a one in 2^{32} chance of guessing right.
$1 / 2^{32} \approx 1 / 4\ 10^9$ is a much better chance than Notsafe's advertised risk, $1 / 10^{12}$!

4b. *(9 points)* List 3 fundamentally different problems with Notsafe's protocol.

There are **many** *things wrong with Notsafe's protocol. Valid answers included:*
1. Notsafe's secret key, Knss, is the same for all servers, and is stored in the server binary. An intruder can just purchase a server from Notsafe. She can then disassemble the binary to get Knss, or she could even feed the unmodified Notsafe binary messages she sniffed from the wire; it will dutifully decrypt them. Compromising Knss in this way breaks the entire protocol.
2. Replay attacks are possible (either on messages, or on the key T that is being sent encrypted).
3. There is no authentication being performed. C doesn't really know he is talking to S as all servers share the same public key. Also, since C's supposed public key Kcp is included in the first message (which anyone can spoof as Knsp is well-known), someone can pose as C by sending their own public key in place of Kcp. Because there is no authentication on either end, Lucifer could even sit in the middle, pretending to be the server to the client, and pretending to be the client to the server! This would allow Lucifer to read every message and reply sent, or even make alterations.
4. An active attacker can replace the encrypted session key $\{ T \}^{Kcp}$ *with random data. When the client decrypts this, it will obtain something that looks like a temporary session key; however, C will not be able to communicate with S with this garbage key, causing denial of service.*

4c. *(5 points)* List one possible problem that could arise from a very poor implementation of the Notsafe server (*not* a fundamental problem with the protocol itself).

Here is one problem.
T is a randomly generated session key. If the pseudorandom number generator is weak, T becomes predictable. (In the worst case, the same T may be used over and over. T may also be easy to guess in other ways; if the seed for the random number generator is based on the current time, for example, which is essentially the same everywhere. The real Netscape Communications Corporation had a protocol which broke for this very reason.)

4d. *(8 points)* Introduce two changes to the Notsafe protocol which improve its security and explain them briefly. *You are not expected to create a completely secure protocol*.

Again, many possible answers. Not including "buy a working protocol," some are:
1. Use a timestamp or nonces to avoid replay.
2. Make T much larger than 32 bits to make brute-force guessing harder.
3. Don't have all servers use the same secret key. Instead, let every server set up its own public/secret key pairs; clients then have to obtain the right public key via certifying authorities.
4. Add authenticaion. This is not simple to implement. One possible method requires **everyone** *to have a well-known public key; therefore, a separate distribution method is needed for the keys. The server no longer receives Kcp with the first message, and must instead get Kcp itself through secure channels. Also, nonces can be used to ensure each message is part of the authenticated chain, and not a replay.*

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:

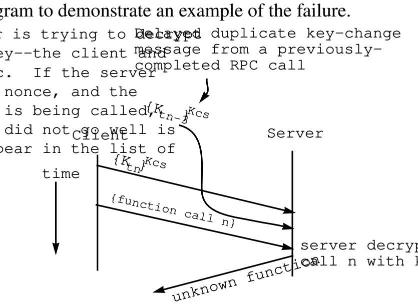
client -> server: {K$_{tn}$}$_{Kcs}$
client -> server: {nonce, function, arguments}$_{Ktn}$
server -> client: {nonce, response}$_{Ktn}$

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.

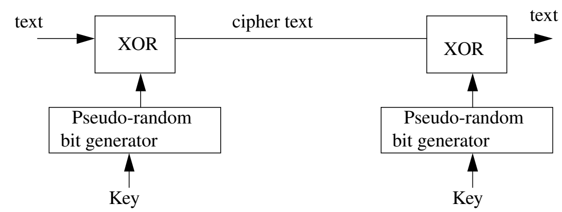
X__Flow-control, since Blast will be unreliable and Send-and-wait is slower.

___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 corr ect 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 messa ge 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.

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 Æ 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 Æ 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 Æ 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 messa ge between C and S, because she can decrypt them using key K'.

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: ______________________________