# Practice Quiz #2 --- Solutions

This document contains all of the solutions to the old Quiz 2's from 1994 to 1998. The solutions are not in order, to motivate you to solve the problems before looking over the solutions. Some of the provided answers are more verbose than were required, to cover all possible right answers. These are usually noted.

The wide range of question styles is not completely accurate. True/False and multiple choice questions are very likely.

Good luck on the quiz Friday. If you have been missing everything, here are the details given on the quiz announcement:

1. Quiz 2 will be from 2-3pm on Friday, April 16, 1999.

2. The quiz will be held on the third floor of Walker for students with last names from A-R, and in 34-101 for those with names S-Z.

3. The quiz will cover all the material up to and including the April 13th recitation.

4. There will NOT be a quiz review. Instead, the TAs will hold extra evening office hours on Wednesday and Thursday before the quiz. The specific hours will be posted on the web site at the beginning of next week.

5. Yes, the quiz is open-book, open-notes.

**6. One question from the practice quizzes will be on the quiz.**

 Name ______________________________

*.

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.

*. 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.

*. 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 reduction 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 of (800-609.6)/800 = 24%.

*. 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.

*.

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.

$1 / 2^{32} \approx 1 / 4 \cdot 10^9$ is a much better chance than Notsafe's advertised risk, $1 / 10^{12}$!

*. The ratio of the numbers of passwords is 788/268 = 38. 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 38 times as long to crack the new password as it takes to crack the old one."

*. 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.

*.

RSA: 1,000Kbytes / 0.1Kbytes/block × 1,000μs/block = 10 seconds.

DES: 1,000Kbytes / 0.1Kbytes/block × 10μs/block = 0.1 seconds.

*. 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.

*. 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.

```
*.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).
```

*.

    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.

*. Performance doubles every year.
2000 - 1995 = 5 years ==> 5 doublings
So, dividing the answer from part (a) by 25, we now need

        300 * 240/3 workstations, i.e., about 3 million workstations.

*.

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.

*. 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.)

*. 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.

*.

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.

*. Unchecked array bounds allowed buffer overruns.

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

155-digit number ==> 55/3 doublings ==> 300 * 255/3 workstations
This is about 99 million workstations.

*. Unsafe debugging commands were available to untrusted traffic.

*. 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’.

*. “Trusted” machines could bypass authorization protocols.

*. 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.

*. **message = Decrypt(rmessage, $Annette);**

*. They provide addressing within each subnet for an IP address resolution protocol.

*. 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.

*. 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.

*.

**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.

*. 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.

*. 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.

*.

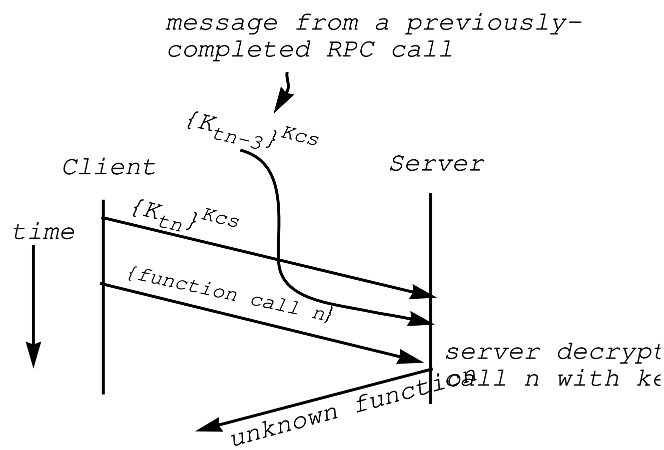
There are two good answers.

1. Louis's protocol provides Alice with **authentication**: the message $\{ \text{Message} \}^T$ 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 $\{ \text{Message} \}^T$ 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.)

***. $Ty**

*. 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.




*. 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, KSpki, 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 Message2: {Bob, KPbob}KSpkpi.

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.

*. Jams the Ethernet briefly.

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

*. None of the above.

*. 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.

*****. Byte (character) framing.

*. 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.
*****. Packet framing.

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

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

*****. Annette was victimized by a replay attack.

*.

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.

*.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.

*. 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.

*.

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.

*****. Even if a chain manufacturer tests each link before assembly, he'd better test the completed chain.

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

*.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.

*****. Assuming Pete is thrust into Solitary remembering no **Key**s, he can't convince inmates that they aren't communicating with a guard.

*.
a) 6
b) 5
c) 4
d) 7
e) 3

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.

*. 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.

*.

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.)

*. 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.

*. 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.

*. 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.

*. Louis speaks for Jim?

*.

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

1. Encryption: Louis encrypts { Message } with Kap1000µs

   (All encryption and decryption in this example uses the public-key scheme, RSA.)

2. Latency: Louis sends { Message }$^{Kap}$ to Alice2000µs

3. Decryption: Alice decrypts { Message }$^{Kap}$1000µs

4. Encryption: Alice encrypts { Reply } with Klp1000µs

5. Latency: Alice sends { Reply }$^{Klp}$ to Louis2000µs

6. Decryption: Louis decrypts { Reply }$^{Klp}$1000µs

Total8000µs

*. 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.)

*. 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.

*. 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.

*. The runtime system may raise a new type of exception as a result of an RPC.

*. 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.)

*. **$Key** takes an intractibly long time to compute from **Key**.

*. 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.

*. **message** was initially sent by Ty.

*. 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.

*.

Again, many possible answers. Not including "buy a working protocol," some are:

1. Use a timestamp or nonces to avoid replay.

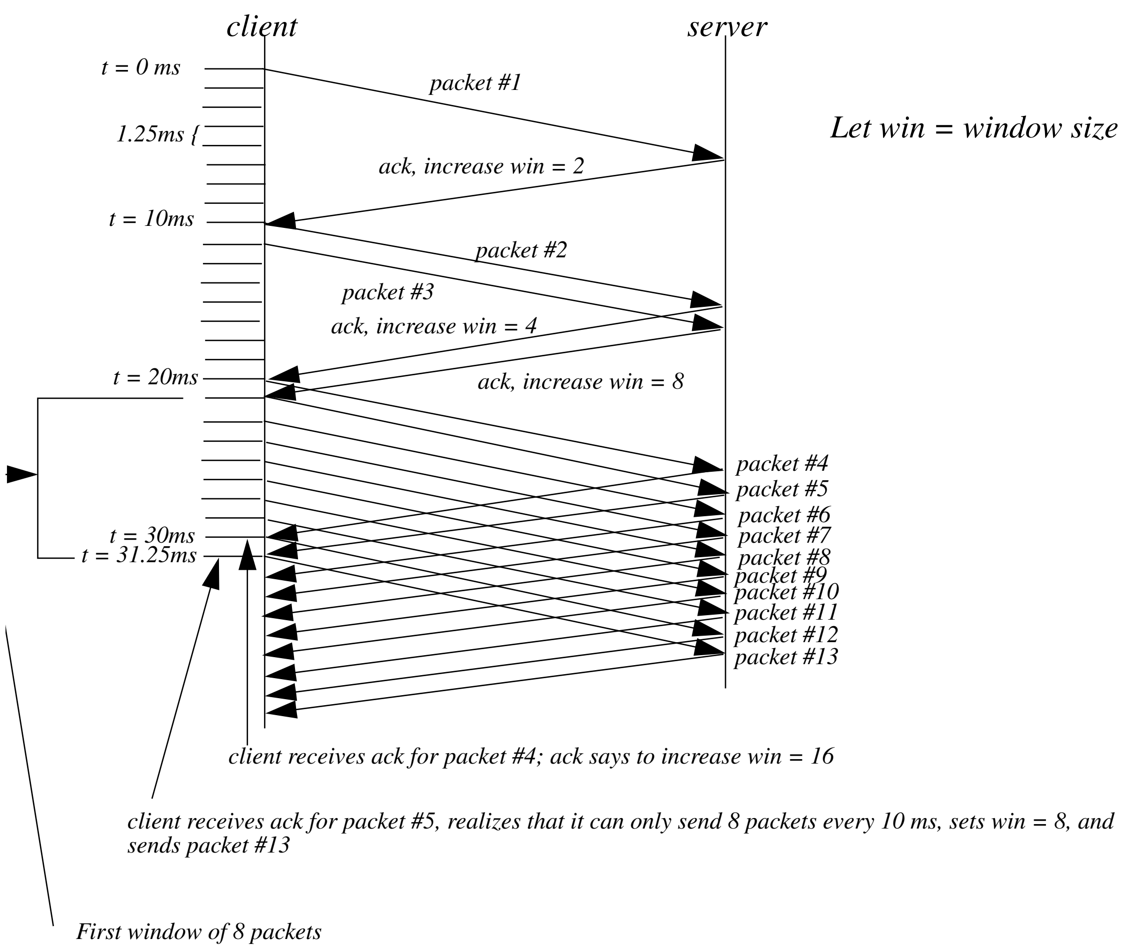
2. Make T much larger than 32 bits to make brute-force guessing harder.

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.

*. **message** was not corrupted in transmission.

*. The ASCII name and logical file number of a file.




 Name ____________________

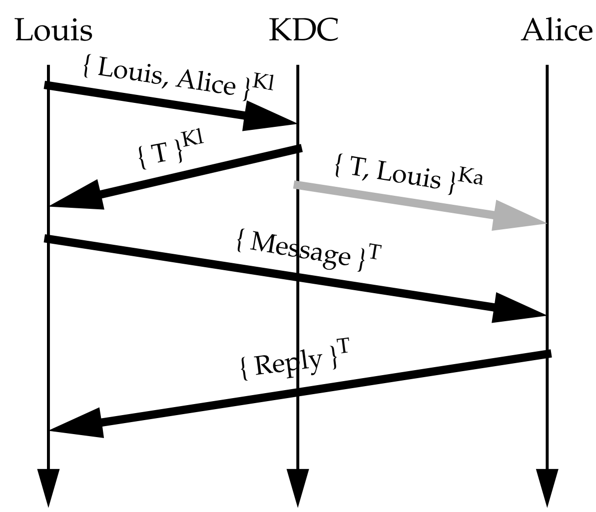
*****. If Pete remembers Ty's uniform number and trusts Ty, an authenticated broadcast message from Ty could be used to remind Pete of other inmates' uniform numbers without danger of deluding Pete.

*. 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.

*. )




It takes 8080μs.

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 Kl10µs

(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 KDC2000µs

3. Decryption: KDC decrypts { Louis, Alice $\}^{Kl}$10µs

4. Encryption: KDC encrypts { T } with Kl10µs

5. Latency: KDC sends { T $\}^{Kl}$ to Louis2000µs

6. Decryption: Louis decrypts { T $\}^{Kl}$10µs

7. Encryption: Louis encrypts { Message } with T10µs

8. Latency: Louis sends { Message $\}^{T}$ to Alice2000µs

9. Decryption: Alice decrypts { Message $\}^{T}$10µs

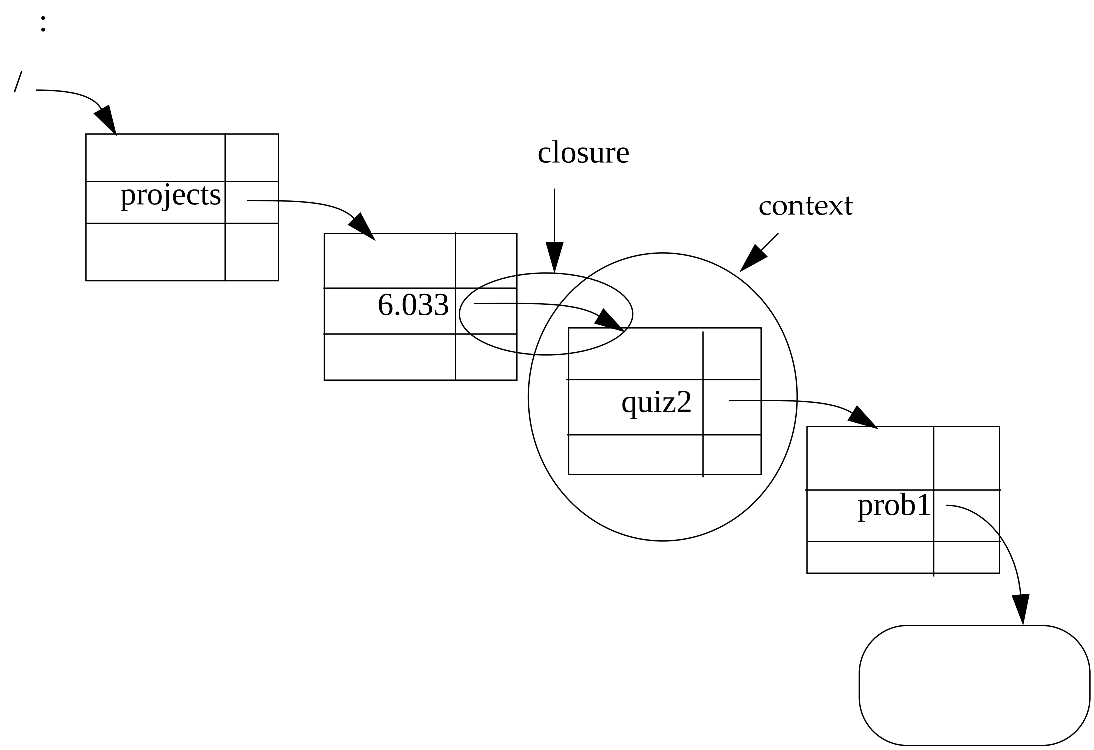
10. Encryption: Alice encrypts { Reply } with T10µs

11. Latency: Alice sends { Reply $\}^{T}$ to Louis2000µs

12. Decryption: Louis decrypts { Reply $\}^{T}$ 10µs

Total 8080µs

Another acceptable answer was 8090µs: 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.




*. 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.