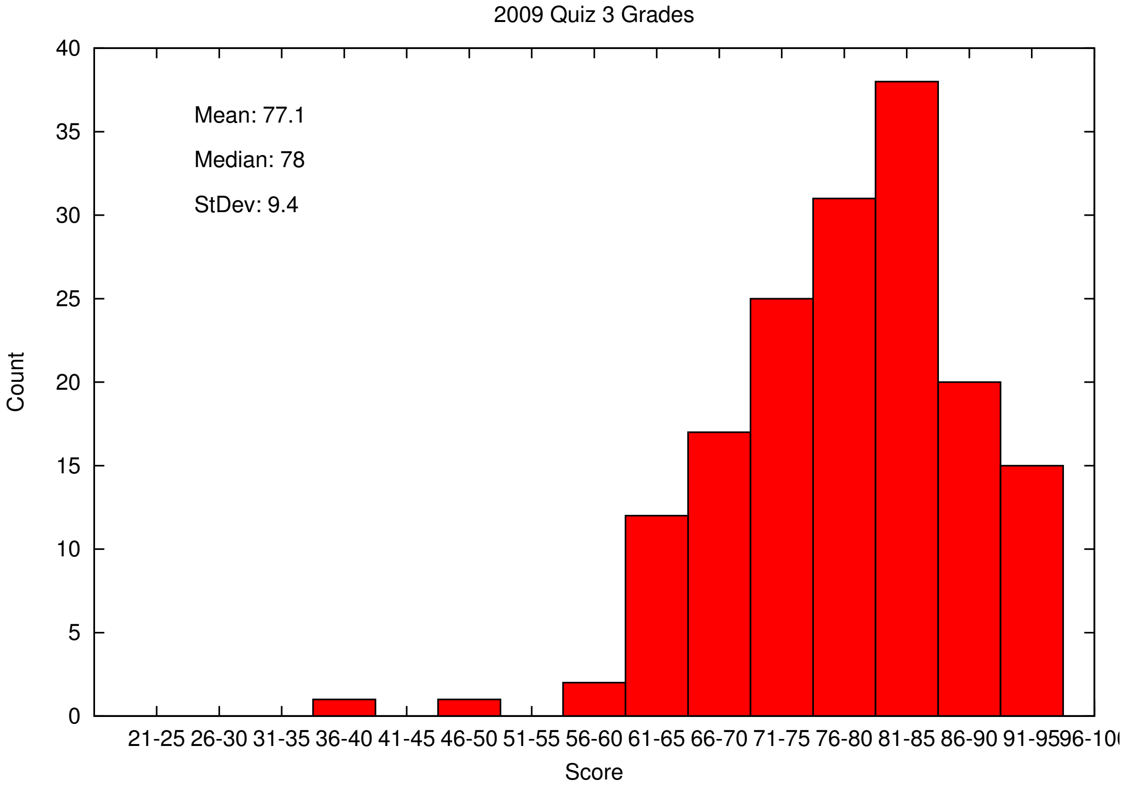
Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.033 Computer Systems Engineering: Spring 2009

Quiz III Solutions



I Reading Questions

1. [4 points]: Based on the description of the Witty worm in "Exploiting Underlying Structure for Detailed Reconstruction of an Internet-Scale Event", by Kumar, Paxson and Weaver (reading #18), which of the following are true?

(Circle True or False for each choice.)

A. True / False Bugs in the worm's design made Witty's behavior harder to analyze.

Answer: False. Bugs made it easier to analyze.

B. True / False To remain effective at detecting worms, it is important for network telescopes to keep their IP address ranges secret.

Answer: True. Otherwise worms that scan the IP address space could leave out known telescope ranges.

2. [4 points]: Which of the following hints appear in Butler Lampson's "Hints for Computer System Design" paper (reading #20), possibly in different words? Mark each True if it appears in the paper, and False if it does not.

(Circle True or False for each choice.)

A. True / False Keep secrets in an implementation, hiding from clients aspects that might change.

Answer: True. Section 2.4 of the paper.

B. True / False Implementations are more important than interfaces, because implementations determine performance.

Answer: False.

C. True / False On coding: don't get it right, get it written; you can always fix it later.

Answer: False.

D. True / False Keep caches small: when in doubt, flush it out.

Answer: False.

3. [8 points]: Based on the paper "Why Cryptosystems Fail", by Ross Anderson (reading #17), which of the following are true?

(Circle True or False for each choice.)

A. True / False The paper argues that the traditional threat model for cryptosystems is wrong.

Answer: True.

B. True / False The paper argues that secure systems cannot be designed the same way safety critical systems are.

Answer: False. The paper argues that these can be designed the same way, making specific reference to models for designing planes and trains.

C. True / False ATM security breaches require that the thief determine both your account number and PIN.

Answer: False. Many of the examples in the paper involve other kinds of attacks at various stages of the ATM's processing.

D. True / False For one bank's ATM network to provide access for a user from another bank, both banks must know the PIN key corresponding to the user.

Answer: False. The PIN key is kept local to the bank.

4. [8 points]: Based on the description of System R in the paper "The Recovery Manager of the System R Database Manager" by Gray, McJones, et al. (reading #21), which of the following are true?

(Circle True or False for each choice.)

A. True / False RAM buffering of disk I/O helps ensure atomicity.

Answer: False. Buffering makes atomicity more complex.

B. True / False Shadow copies, without a log, are sufficient to ensure atomicity in the presence of concurrent transactions that both update the same file.

Answer: False. System R requires the incremental log to ensure transaction consistency after a crash.

C. True / False A transaction is guaranteed to survive a crash once its log entry is written to memory.

Answer: False. The log entry is not stable until written to disk.

D. True / False Uncommitted transactions may have issued writes *before* the last checkpoint. Therefore checkpoints may include incomplete transactions.

Answer: True.

5. [8 points]: Based on the paper “Beyond Stack Smashing: Recent Advances in Exploiting Buffer Overruns”, by Pincus and Baker (reading #16), which of the following are true?

(Circle True or False for each choice.)

A. True / False By not allowing writes outside of the bounds of objects, Java eliminates all risk of attacks based on stack smashing, assuming that the VM and any native libraries are bug free.

Answer: True.

B. True / False Setting the permissions on the stack memory to prevent execution of code would foil attacks based on “return into libc”.

Answer: False. The “return into libc” attack executes preexisting functions (i.e., `system()`), which do not reside on the stack.

C. True / False Making the stack begin at a memory location chosen randomly at runtime would foil the original stack smashing exploit.

Answer: True.

D. True / False Using function pointers presents additional opportunities for arc injection.

Answer: True.

6. [8 points]: Based on the description of ObjectStore in the paper “The ObjectStore Database System” by Lamb, Landis, et al. (reading #23), state whether each of the following is true or false.

(Circle True or False for each choice.)

A. True / False If an existing program, with its own implementation of lists and sets, wants to use ObjectStore to make its data persistent, it must switch to ObjectStore’s list and set collections.

Answer: False.

B. True / False ObjectStore needs to know the location of all pointers in all persistent data structures.

Answer: True.

C. True / False The caching protocol assumes the programmer will obtain a lock before modifying a persistent object.

Answer: False.

D. True / False The locking protocol always allows applications to execute concurrently, as long as they are not accessing the same object.

Answer: False. Locking is performed on a page granularity in ObjectStore, hence no two locked objects in the same page may be accessed concurrently.

7. [8 points]: Based on the description of Porcupine in the paper "Manageability, Availability and Performance in Porcupine: A Highly Scalable Internet Mail Service" by Saito, Bershad and Levy, state whether each of the following is true or false.

(Circle True or False for each choice.)

A. True / False If all of the servers storing mailbox fragments for some user are down, the system will not be able to accept new mail for that user.

Answer: False. The system may write incoming mail to new fragments on alternate servers.

B. True / False Assume you have a large-scale Porcupine deployment, there are more concurrent users than servers, and all users have similar usage patterns. Storing more mailbox fragments for each user would reduce throughput.

Answer: True. Reading and writing to additional servers will require more disk accesses, and more queries when fetching mail for a user.

C. True / False A user that fetches but does not delete their mail from a Porcupine server twice in a row can see different messages, even if no new messages are received.

Answer: True. Messages are propagated "lazily" in Porcupine, and may take some time to be communicated between servers.

D. True / False If one user's mailbox fragment list is causing too much load on one server, Porcupine can move just that user's mailbox fragment list to another server.

Answer: False. Fragment lists are mapped to servers at a coarser granularity than individual users, and hence users cannot be moved individually.

II BLOP

Ben Bitdiddle is building a distributed gambling system called Ben's Land Of Poker (BLOP). In BLOP, users play hands of poker (cards) against each other. Each user is given two *private* cards that the other users can't see. Three additional *public* cards (which can be seen by all users) are revealed one-by-one. Users place bets in four rounds of betting, one after users receive their two cards, and one after each public card is revealed. Bets are in dollars, are > 0, and are not more than a user's remaining balance. At the end of the fourth round of betting, the user with the best hand (according to the rules of poker) wins.

Each user in BLOP has an account with a balance that is stored on one of BLOP's servers. Different users playing in a hand may have their accounts hosted on different servers. Between hands, users can add money to an account with a credit card. BLOP credits a user's account when that user wins a hand. BLOP withdraws from a user's account whenever the user places a bet. During a given hand, one server is appointed a *leader* that is responsible for running the hand: it draws the cards, transfers money from users' accounts to a central *pot* that contains the money bet so far, and sends data to the clients.

During a hand, clients talk only to the leader. The leader sends information about public and private cards to the clients, who connect from their own desktop machines, and also updates the balances of accounts stored on the disk of the non-leader servers (the *subordinates*) and the balance of the pot stored locally on the leader's disk.

The pseudocode used by the leader is as follows:

```
run_hand:
  // (1) beginning of hand
  curPot = 0
  write(pot,0)  // store value of pot on disk

  for each client c:
    // handMsg tells clients about their cards
    sendRPC handMsg(privateCards[c]) to c

  for (round in [0..3])
    // collectBets does one round of betting with clients,
    // returning each of their bets
    bets = collectBets(clients)

    for each client c:
      // servers is an array that stores the subordinate
      //    server for each client's account data
      sendRPC deductMsg(c, bets[c]) to servers[c]
      curPot = curPot + bets[c]
      write (pot, curPot)  // update value of pot on disk
      if (round != 3) // last round is just for betting
         sendRPC handMsg(publicCards[round]) to c

  winner = computeWinner(hands)
  sendRPC deductMsg(winner, -curPot) to servers[winner]
  // (2) end of hand
```

The code to process deductMsg on each of the servers looks as follows:

```
deductMsg(account, amt):
  prevBal = read(account)
  if (prevBal > amt):
     write(account, prevBal - amt)
  else
     write(account, 0)
```

Assume that the network uses a reliable (exactly once) RPC protocol to between the leader and the subordinate servers, such that the leader waits to receive an acknowledgment to each sendRPC request before proceeding. Also assume that write operations are atomic—that is, they either complete or do not complete, and after they complete, balances are on disk.

Initially, Ben's servers run a hand without using any transactions, logging, or special fault tolerance. If the leader does not receive an acknowledgment to an RPC within two minutes, it tells the clients the hand is aborted but takes no other recovery action. If the leader crashes, the clients eventually detect this and notify the users that the hand has aborted. Initially, the leader performs no special action to recover after a crash.

During a hand, each client is given up to two minutes to place a bet. If they do not respond within two minutes (either because they left the hand, or their machine crashed), they forfeit the hand and lose any money they may have bet (play continues for the other clients in the hand.)

8. [6 points]: Which of the following could go wrong if one of the subordinate servers crashes in the middle of a hand, assuming only one hand runs at a time:

(Circle True or False for each choice.)

A. True / False After the leader aborts the hand, and the failed subordinate restarts, it is possible for the sum of all of the on-disk balances of the users in the hand to be greater than when the hand started.

Answer: False. The leader only deducts from user accounts during the hands.

B. True / False After the leader aborts the hand, and the failed subordinate restarts, it is possible for the sum of all of the on-disk balances of the users in the hand to be less than when the hand started.

Answer: True.

Alyssa P. Hacker tells Ben that he should use transactions and two-phase commit in his implementation of BLOP. He modifies BLOP so that reads and writes of the pot and of user accounts on the subordinates are done as a part of a transaction coordinated with two-phase commit and logs. All log writes go directly to an on-disk log. Ben's scheme operates as follows:

- Prior to beginning a hand (before the comment labeled (1)), the leader writes a start of transaction (SOT) log entry and sends each subordinate a BEGIN message. Each subordinate logs an SOT log entry.
- Prior to any update to the pot, the leader writes an UPDATE log entry. Prior to any update to a user account balance, subordinates write an UPDATE log entry.
- At the end of a hand (at the comment labeled (2)), the leader sends each subordinate a PREPARE message for the transaction. If the subordinate is participating in the transaction, it logs a PREPARED log entry and sends a YES vote to the leader. If the subordinate is not participating in the transaction (because, for example, it crashed and aborted the transaction before preparing), it sends a NO vote.
- If all subordinates vote YES, the leader logs a COMMIT log entry and sends a COMMIT message to each of the subordinates. Subordinates log a COMMIT record and send an ACK message.
- Otherwise, the leader logs an ABORT log entry and sends a ABORT message to each of the subordinates. Subordinates log an ABORT record, roll back the transaction, and send an ACK message.

Assume Alyssa's additions to Ben's code correctly implement two-phase commit, and that the system uses the standard two-phase commit and log-based recovery protocols for handling and detecting both leader and subordinate failures and recovery. Both two-phase commit and log-based recovery were discussed in lecture. Two-phase commit is described in Section 9.6.3 of the course notes, and log-based recovery is described in Section 9.3.3 and 9.3.4 of the course notes.

Ben also modifies his implementation so that if one of the subordinates doesn't respond to a `deductMsg` RPC, the leader initiates transaction abort.

9. [9 points]:

Which of the following statements about the fault tolerance properties of Ben's BLOP system with two-phase commit are true?

(Circle True or False for each choice.)

A. True / False If a subordinate crashes after the leader has logged a COMMIT, and then the subordinate completes recovery, and the leader notifies all subordinates of the outcome of the transaction, it is possible for the sum of all of the balances of the users in the hand to be less than when the hand started.

Answer: False. All the subordinates will COMMIT, they will all complete their updates as directed by `run_hand`, and `run_hand` ensures that the sum of the balances ends up unchanged.

B. True / False If the leader crashes before it has logged a COMMIT and then completes recovery and notifies all subordinates of the outcome of the transaction, the sum of all of the balances of the users in the hand is guaranteed to be equal to the sum of their balances when the hand started.

Answer: True. In both the COMMIT and ABORT cases, the sum remains the same.

C. True / False If the leader crashes after one the subordinates has logged a PREPARE, it is OK for that non-leader to commit the transaction, since the transaction must have completed on the subordinate.

Answer: False. It is only safe for a subordinate to commit if *all* subordinates vote YES.

10. [4 points]: Ben runs his system with 2 subordinates and 1 separate leader. Suppose that the mean time to failure of a subordinate in Ben's system is 1000 minutes, and the time for a subordinate to recover is 1 minute, and that failures of nodes are independent. Assuming that each hand uses both subordinates, and that the leader doesn't fail, the availability of Ben's system is approximately:

(Circle the BEST answer)

A. 499/500

B. 999/1000

C. 999/2000

D. 1/1000

Answer: 499/500. To a first approximation one or the other subordinate will be unavailable for 2 minutes out of 1000.

To increase the fault-tolerance of the system, Ben decides to add replication, where there are are two replicas of each subordinate.

Ben's friend Dana Bass suggests an implementation where one replica of each subordinate is appointed the *master*. The leader sends messages only to masters, and each master sends the balance of any accounts it hosts that were updated in a transaction to the other *worker* replica, after it receives the COMMIT message for that transaction. Masters do not wait for an acknowledgment from their worker before beginning to process the next transaction.

When a master fails, its worker can take over for it, becoming the master. When the failed replica recovers, it simply copies the balance of all bank accounts from the new master and becomes the worker. Dana's implementation does nothing special to deal with the case where a COMMIT completes on a master and the master fails before sending the transaction to the worker, which can result in the worker taking over without learning about the most recent committed transaction.

11. [9 points]: Which of the following statements about this approach are true, assuming that failures of masters and workers are independent, and that the leader node never fails:

(Circle True or False for each choice.)

A. True / False Dana's approach improves the availability (that is, the probability that some subordinate responds to `deductMsg` for a given client's account) versus a non-replicated system, as long as the worker node can take over for a failed master in less than the time it takes for the master to restart.

Answer: True.

B. True / False Dana's implementation ensures single-copy serializability, since a user can never see results of hands that reveal that the system is replicated.

Answer: False. For example, suppose a master fails after replying to a COMMIT for a balance deduction but before sending the updated balance to the worker. The worker will then take over as master with a balance that is too high. This could not have happened in the non-replicated system.

C. True / False Suppose Dana modifies her approach to have three replicas for each subordinate (two workers and a master.) Compared to the the approach with two replicas per subordinate, this three node approach has lower availability since the probability that one of the three replicas crashes is higher than the probability that one of two replicas crashes in the original version.

Answer: False. The system has higher availability, since it is available if any one of the three replicas is alive.

III BitPot

In order to back up your laptop's files, you sign up with BitPot. BitPot is an Internet-based storage service. They offer an RPC interface through which you can read and write named files. BitPot gives each of their customers an identification number (cid, an integer). The RPC interface looks like:

```
putfile(cid, filename, content)
getfile(cid, filename) -> content
```

`putfile()` and `getfile()` send their arguments over a network connection to the BitPot server, and wait for a reply.

BitPot provides a separate file namespace for each `cid`; for example, `getfile(1, "x")` and `getfile(2, "x")` will retrieve different data. Neither BitPot nor `getfile()` / `putfile()` do anything special to provide security. Here is what the BitPot server's RPC handlers do:

```
putfile_handler(cid, filename, content):
  name1 = "/customers/" + cid + "/" + filename
  write content to file name1 on the BitPot server's disk
  return a success indication

getfile_handler(cid, filename):
  name1 = "/customers/" + cid + "/" + filename
  if file name1 exists on the BitPot server's disk:
    content = read file name1
    return content
  else:
    return a failure indication
```

You are worried about the security of your files: that other people (perhaps even malicious BitPot employees) might be able to read or modify your backup files without your permission.

For all of the following questions, attackers have limited powers, including only the following:

- Observe any packet traveling through the network;
- Modify any packet traveling through the network;
- Send a packet with any content, including copies (perhaps modified) of packets observed on the network;
- Perform limited amounts of computation (but not enough to break cryptographic primitives);
- Read and write the contents of the BitPot server's disk (for attackers that are BitPot employees);
- Observe or modify the behavior of the BitPot server's software (for attackers that are BitPot employees);

Attackers have no powers not listed above. For example, an attacker cannot guess a cryptographic key; cannot guess the content of the files on your laptop; cannot observe or modify computations on your laptop; and cannot exploit buffer overruns or other bugs on your laptop or BitPot's servers (such as manipulating path names used to read and write files).

You should assume that there are no failures (except to the extent that the attacker's powers allow the attacker to do things that might be construed as failures).

Scheme One

You decide to encrypt each file you send to BitPot with a key that only you know, using a shared-secret cipher (see section 11.4.2 “Properties of `ENCRYPT` and `DECRYPT`” in the course notes). When you want to back up a file to BitPot, you call `backup1()`:

```
backup1(filename):
  plaintext = read contents of filename from your laptop's disk
  ciphertext = ENCRYPT(plaintext, K)
  putfile(cid, filename, ciphertext)
```

and when you need to retrieve a file from BitPot, you call `retrieve1()`:

```
retrieve1(filename):
  ciphertext = getfile(cid, filename)
  plaintext = DECRYPT(ciphertext, K)
  print plaintext
```

`cid` is your BitPot customer ID and `K` is your cipher key.

Only you and your laptop know `K`. `ENCRYPT` and `DECRYPT` withstand all the attacks mentioned in 11.4.2.

12. [8 points]: Which of the following are true about Scheme One?
(Circle True or False for each choice.)

A. True / False `retrieve1(f)` will return exactly the same data that your most recent completed call to `backup1(f)` for the same `f` read from your laptop’s disk, despite anything an attacker might do.

Answer: False. For example, if an attacker modifies the contents your data on BitPot’s disks, `retrieve1()` will produce something other than the original content of your file.

B. True / False Eavesdroppers watching packets on the network may see ciphertext but are very unlikely to be able to figure out the plaintext content of your files.

Answer: True.

C. True / False BitPot’s employees may see ciphertext but are very unlikely to be able to figure out the plaintext content of your files.

Answer: True.

D. True / False If someone modifies one of the files BitPot stores for you, `retrieve1()` is guaranteed to print random data (or to signal an error).

Answer: False. For example, suppose you back up two files to BitPot, f1 and f2. If a malicious BitPot employee copies /customers/cid/f1 to /customers/cid/f2, then `retrieve1(f2)` will yield the backed-up content of f1.

Scheme Two

Your friend Belyssa says you need to use authentication, using `SIGN` and `VERIFY` as described in section 11.3.4 of the course notes. She's not sure quite how best to do this, and suggests the following plan.

Belyssa's plan operates at the RPC layer, beneath `putfile()` / `getfile()`. The client (your laptop) and the server (BitPot) each have a shared-secret signing key (`Kc` and `Ks`, respectively). Each `SIGN`s each RPC message it sends, and `VERIFY`s each RPC message it receives. Each of them ignores any received message that doesn't verify. For simplicity, assume there is only one client and only one server, so that the server doesn't have to manage a table of per-client keys. The client and server both know both `Kc` and `Ks`.

```
// client putfile() and getfile() send requests like this:
send_request(msg):
  T = SIGN(msg, Kc)
  send {msg, T} to BitPot

// the server calls this with each incoming network message:
receive_request(msg, T):
  if VERIFY(msg, T, Kc) == ACCEPT:
    result = call putfile_handler() or getfile_handler()
    send_reply(result)
  else:
    // ignore the request

send_reply(msg):
  T = SIGN(msg, Ks)
  send {msg, T} to client

// the client calls this with each incoming network message:
receive_reply(msg, T):
  if VERIFY(msg, T, Ks) == ACCEPT:
    process msg (i.e. tell getfile() or putfile() about the reply)
  else:
    // ignore the reply
```

Only your laptop and BitPot's server know `Kc` and `Ks`. `SIGN` and `VERIFY` withstand all the attacks mentioned in 11.3.4.

You execute the following procedure on your laptop using Scheme Two:

```
test():
  write "aaaa" to file xx
  backup1(xx)
  write "bbbb" to file yy
  backup1(yy)
  write "cccc" to file yy
  backup1(yy)
  retrieve1(yy)
```

That is, you back up file `xx`, then you back up two different versions of `yy`, then you retrieve `yy`. `test()` will print a value (from the call to `retrieve1()`).

13. [7 points]: Which of the following values is it possible for `test` to print?
(Circle ALL that apply)

A. `aaaa`

Answer: `aaaa` is possible. A BitPot employee could copy the backed up `xx` to the backed up `yy`.

B. `bbbb`

Answer: `bbbb` is possible. A BitPot employee could save the old backed up `yy` and copy it to the backed up `yy` after the final `backup1(yy)`.

C. `cccc`

Answer: `cccc` is possible.

D. `(^.^) insert witty message here (^.^)`

Answer: The properties of SIGN/VERIFY and ENCRYPT/DECRYPT allow this possibility for attackers who are BitPot employees. However, with practical implementations of ENCRYPT/DECRYPT, it might be difficult for an attacker to produce this result.

Scheme Three

Your other friend, Allen, is uneasy about the properties of Belyssa's scheme. He proposes eliminating Belyssa's changes, and instead `SIGN`ing and `VERIFY`ing only in the backup and retrieve procedures.

Allen's new backup (called `backup2()`) includes `SIGN`'s authentication tag in the "content" it sends to BitPot, and Allen's new retrieve extracts and `VERIFY`s the tag from the data sent by BitPot.

```
backup2(filename):
  plaintext = read contents of filename from your laptop's disk
  ciphertext = ENCRYPT(plaintext, K)
  T = SIGN(ciphertext, Kx)
  what = {ciphertext, T}
  putfile(cid, filename, what)

retrieve2(filename):
  what = getfile(cid, filename)
  {ciphertext, T} = what
  if VERIFY(ciphertext, T, Kx) == ACCEPT:
    plaintext = DECRYPT(ciphertext, K)
    print plaintext
  else:
    // ignore the getfile reply
```

`K` is the shared-secret cipher key from Scheme One, which only you and your laptop know. `Kx` is a shared-secret signing key known only to you and your laptop.

You execute the following procedure on your laptop using Scheme Three. (This is the same procedure as for the previous question, but uses `backup2()` and `retrieve2()`).

```
test2():
  write "aaaa" to file xx
  backup2(xx)
  write "bbbb" to file yy
  backup2(yy)
  write "cccc" to file yy
  backup2(yy)
  retrieve2(yy)
```

14. [7 points]: Which of the following values is it possible for `test2` to print?
(Circle ALL that apply)

A. `aaaa`

Answer: `aaaa` is possible; the example from the previous question works.

B. `bbbb`

Answer: `bbbb` is possible; the example from the previous question works.

C. `cccc`

Answer: `cccc` is possible.

D. `(^.^) insert witty message here (^.^)`

Answer: not possible. The VERIFY in `retrieve2()` will reject any content not previously signed by the laptop.

IV Systems Design Experience

There are known knowns.
There are things we know
that we know.

There are known unknowns.
That is to say
There are things that we now know
we don't know.

But there are also unknown unknowns,
There are things we do not know
we don't know.

15. [2 points]: The aforementioned quote is highly applicable to the design of large computer systems. According to the guest lecture on May 11, who first uttered this sage advice?

(Circle the BEST answer)

A. Albert Einstein

B. Butler Lampson

C. Mahatma Gandhi

D. Donald Rumsfeld

Answer: Donald Rumsfeld.

End of Quiz III