Massachusetts Institute of Technology
6.857: Network and Computer Security
Professor Ronald L. Rivest

Handout 32
November 12, 2002

Fall 2002 Midterm Solutions

1. There are six (6) problems and a total of 100 points. This midterm is due on *Thursday, October 31, 2002* at the beginning of class. Late midterms will *not* be accepted. If you can't hand the midterm in or have a friend hand it in, then drop it off with the TA or course secretary BEFORE the deadline.

2. **Policy:** The midterm is open book, but *no collaboration is allowed!* You may not communicate with any person (except for the course staff) about any aspect of the midterm until after the hand-in deadline, even if you have already handed in your midterm. Violation of this policy will be dealt with severely.

   You may use your notes from this course, class handouts, textbooks, and resources online. Give citations for any material (other than your lecture notes and class handouts) that you use. Keep in mind that citations are not always correct or sufficient for justification. It is best to rely on your own reasoning and the material presented in class.

3. **Format:** Type up your solutions, staple each problem separately (subparts of a problem can be stapled together in proper order), and put your name and problem number on each page. Use at least an 11pt font size and 1 inch margins. You have a week for the midterm, so there is no excuse for sloppiness. You will receive 1 point for each problem in which you follow these directions correctly.

4. **Page limit:** Each problem has a specified a page limit. Pages beyond the specified limit will be ignored. Do not feel obliged to use the entire allotment; many problems have concise answers.

5. **Bugs, etc.:** Eventual bugs in the midterm or any hint the staff may feel inclined to give will be reported to you through the `6.857-students-public@mit.edu` mailing list. Check your e-mail regularly during the examination period! If you have not received e-mail on this list yet, then you are probably not on it. Tell the TAs ASAP.

Problem Q-1. I don't like spam [20 pts]

The solution to this problem (parts a, b, c) is adapted from Kenny Chi Kong Fong's submission. We included Jim Paris's submission for part d.

In this problem, limit your write up to three pages total. Additional pages will be ignored.

Email has become nowadays a wonderful means of communication: it is widely accessible, simple to use, virtually cost-free, and you can reach people at the other end of the world almost instantaneously.

But there is a flip side to this coin. Email has become as well a fantastic way of sending unsolicited advertising or marketing to people. Post a message in a newsgroup about digital rectal thermometers, and companies selling health equipment may send you information about the latest health devices that are completely useless to you. Enter your email address in an online raffle, and you may end up receiving an endless series of emails inviting you to enter even more of them.

As a consequence, fighting spam has become an ever-increasing need. Hoping to lay his hand on a golden opportunity, Ben Bitdiddle decides to enter the anti-spam business.

(a) Describe the objectives of an anti-spam personal agent. What would you expect an anti-spam personal agent to do for you? How would you rate the effectiveness of an anti-spam personal agent?

In your discussion, remember that the agent is fallible: there can be false positives (a non-spam email is detected as spam) and false negatives (a spam email is not detected as such).

Solution: The objectives of an anti-spam personal agent are:

- The agent shall filter emails that the user does not want to receive (i.e. it filters unwanted emails based on user's preference).
- The agent shall provide a user-friendly interface and be easy to use.
- The agent shall not delete any emails that it filters off to guard against false positives (i.e. a non-spam email that is detected as spam). It shall provide a robust backup system and recovery mechanism to allow a user to restore a false-positive email back to the inbox.
- The agent shall allow the user to forward any false-negative email (i.e. a spam email not detected as such) to the bulk/spam box, or even to delete it.
- The agent shall try to minimize the number of false positives and false negatives. While a high ratio of false negatives is rather bothering (having to delete a spam email), a positive ratio of false positives is extremely annoying (missing a possibly important email), and should be avoided. Therefore a global error rate should weight the ratio of false positives much more than that of false negatives.
- The agent shall be customizable, so that the user can instruct the agent to filter emails based on different criteria, such as "filter by sender", "filter by recipients", "filter by subject", "blocking attachment", etc.
- The agent shall provide spam reporting mechanism to fight back against spam. For example, it can send an unsubscribe notice to the spam sender, or an abuse report to the ISP administrator.
- The user shall be able to set the agent to run invisibly to the user for convenience and privacy.

- The agent shall be able to alert the user when it filters an email, but the user shall be able to turn the alert option off.
- The agent shall be able to support multiple email accounts.
- Installation of the agent shall be easy and simple, and the use of the agent shall not require new investments in hardware or software.
- The agent program shall use as little CPU time and bandwidth as possible.

Although many current anti-spam agents rely on AI techniques to smartly filter the emails based on content or other properties of spam email or spam senders, Ben decides to focus on a computational approach: the sender has to complete a certain amount of work for his email to get into the recipient's inbox.

Recall the hash-cash technique discussed in lecture: upon reception of an email, the recipient sends a challenge (x, z) (where z is a given b-bit long string) to which the sender needs to reply with r such that $h(x||r)$ starts with z, where h is a one-way hash function and $||$ denotes concatenation. To make the protocol non-interactive, one can set x to be the concatenation $(m||e||d)$ of the message m, the recipient's email address e and the current date/time d.

Instead of basing the work on the difficulty of finding a partial inversion, Ben Bitdiddle decides to base his scheme on the difficulty of finding a partial *collision*: the sender now needs to find $r \neq r'$ such that $h(m||e||d||r)$ and $h(m||e||d||r')$ have the same b leading bits.

(b) Evaluate the amount of work needed by the sender to meet the challenge in Ben's scheme, in terms of processing time and storage space. Compare it to the hash-cash technique.

How do the time and memory requirements grow with the number of recipients?

You must specify any assumptions you are making.

Solution: To meet the challenge in Ben's scheme, the sender needs to find r and r' such that $r \neq r'$, and $h(m||e||d||r)$ and $h(m||e||d||r')$ have the same b leading bits. The sender achieves this by hashing $(m||e||d||r)$ with different r's until a collision on the first b leading bits occurs. He needs to maintain a table to store the pairs (r, h_r) that have been computed so far, where h_r is the first b leading bits of $h(m||e||d||r)$. An efficient way to implement this is to have the table mentioned above as a hash table indexed by the second entry (i.e. h_r), associated with a hash function g. Every time when the sender tries a new r, he computes $g(h_r)$ and inspects the $g(h_r)^{th}$ entry of the hash table. If this entry contains a pair $(r', h_{r'})$ such that $r \neq r'$ and $h_r = h_{r'}$, then a collision is found. Otherwise, the sender inserts the pair (r, h_r) in that entry and tries another r. The reason for using a hash table is to ensure that the pairs (r, h_r) can be inserted into the table in constant time (in contrast to sorting the h_r's in the table).

By the birthday paradox, the sender needs to try an expected number of $\sqrt{2^b}$ r's to obtain such a partial collision. Therefore, the amount of work needed by the sender to meet the challenge is given by $\sqrt{2^b}$ hash evaluations in terms of processing time and $2b\sqrt{2^b}$ bits in terms of storage space (assuming that he tries b-bit r's, a pair (r, h_r) is $2b$-bit long).

In the hash-cash technique, the sender needs to find an r such that $h_r = z$. Since the sender just needs to compute h_r for different r's and compares h_r with a fixed z,

no storage is required. As discussed in class, he needs to try an expected number of 2^b r's to find an r such that $h_r = z$, since z is b-bit long. We conclude that the hash-cash technique requires about 2^b hash evaluations but no storage space. In terms of processing time, the variant of hash-cash is much faster than the original hash-cash technique, but such an improvement in running time is obtained by sacrificing a large amount of storage space.

The time requirement grows proportionally with the number of recipients for both the hash-cash variant and the original hash-cash because the e-mail address e is different for each recipient. However, the storage requirement does not grow with the number of recipients because we can reuse space. With n recipients, the variant of hash-cash requires $n\sqrt{2^b}$ hash evaluations and $2b\sqrt{2^b}$ bits of storage, while the original hash-cash technique requires $n2^b$ hash evaluations but negligible storage space.

(c) Discuss the efficiency of this scheme: what would be an appropriate value for b? can the sender perform any pre-computation?

You can assume for this question that a typical *personal* computer can compute around one million hash evaluations per second.

Solution: From part (b), we know that the processing time for each recipient is approximately $\sqrt{2^b}$ hash evaluations. As discussed in class, a typical spam sender cannot tolerate a per-recipient processing time of 5 seconds as he sends the same spam email to millions of recipients. Assuming that he uses his personal computer to send spam emails and his personal computer can compute around one million hash evaluations per second (i.e. about 2^{20} hash evaluations per second), he would have a per-recipient processing time of at least 5 seconds if the following is satisfied:

$$\frac{\sqrt{2^b}}{2^{20}} \geq 5$$

$$2^{b/2} \geq 5 \cdot 2^{20} > 2^{22}$$

$$b/2 > 22$$

$$b > 44$$

With $b = 45$, the corresponding space requirement (in bytes) is:

$$2b\sqrt{2^b}/8 = 90 \cdot \sqrt{2^{45}}/8 \approx 67\text{MB}$$

We conclude that an appropriate value for b is 45.

Unfortunately, this value for b can only guard against spam senders that do not perform any pre-computation. Since this scheme is non-interactive, the sender can always perform pre-computations. He just needs to choose a date/time d well beyond the current time, and then performs pre-computation to find r and r' that yield the partial collision for each recipient long before he sends all the spam emails collectively at that date/time d. However, the time needed for pre-computing the r and r' is the same as the time needed to compute them during the course of sending the spam emails. Thus, unless the sender really has lots of free time, a value of 45 for b is pretty effective in guarding against not-so-patient spam senders.

(d) What are the advantages and drawbacks of this scheme compared to the more traditional approaches, based on heuristics? Remember to consider the different issues in actually implementing this scheme (deployment, ease of use, customization, ...).

Following is a minor revision of the solution submitted by Jim Paris.

Solution: Advantages of a computational solution to the spam problem:

- Spam can be written to trick heuristics, whereas there's no way to avoid the necessary computations in this model.
- A cost is imposed on the sender, meaning that sending spam is no longer a free medium for advertisers.
- Does not require custom heuristics for a specific application, user, or language.
- Can be implemented uniformly and be standardized without jeopardizing effectiveness.
- In the interactive scheme, recipients could still use heuristics and request different amounts of computation depending on the result.

Disadvantages:

- Having to wait for computation is annoying and wastes time of senders of legitimate e-mails.
- Time for e-mail to get delivered depends on system speed and load.
- Legitimate owners of mailing lists are not distinguished from spammers and must also expend significant computation.
- Speed of machines used to send e-mails varies easily by 10^3: my old Palm running at 4MHz hardly compares to a dual Pentium 4 at 3GHz, and so what takes 1 minute for one computer could easily take 16 hours on another.
- Solutions to above problem just transfer burden to service providers and other e-mail gateways.
- Computing collisions is trivially parallelizable for separate e-mails, and cheap parallel hardware could be built and sold to speed up the computations, rendering it a mere annoyance and one-time cost to spammers.
- Most devices don't have the memory available to use Ben's scheme (e.g. old computers, cell phones, PDAs, etc.)
- All e-mail software would need to be rewritten, since the computation must be done directly on the sender's computer.
- Last but not least, the computational approach needs a wide (if not total) adoption in order to work: indeed, since the recipient automatically discards any email with no proper "proof of work", it will thus discard all emails sent with non-compliant software.

Problem Q-2. Security of embedded devices [20 pts]

In this problem, limit your write up to three pages total. Additional pages will be ignored.

As technology progresses, we will likely see embedded computers in almost everything, from toasters to food packaging (perhaps in the form of RFID chips).

One way to manage these devices securely is to let every such device have an "owner" that can give it commands. For example, your PDA might be the "owner" for your toaster. Commands from other parties (including previous owners) are to be ignored by the device.

(a) Explain how such a device might be programmed to recognize commands from its owner, and only its owner. Show how this can be done with both classical (symmetric private-key) technology and with public-key technology.

Solution: Before designing a security protocol, it is important to assess the requirements. A reasonable security model will require integrity protection and replay prevention of commands.
For a security protocol to make any sense, all assumptions must be explicitly stated. For the asymmetric scheme, we accepted the following assumptions when explicitly stated: the new and previous owner have a secure, mutually authenticated link or the owners trust a third-party certificate authority. Note that a previous owner cannot simply accept a public key of a new owner from thin-air. The public key is not known to speak for any particular identity.
The symmetric protocol requires similar assumptions. You either need a secure link between principals or a trusted third party to introduce principals to each other.
A hybrid approach seems to work well. If principals have a secure link not likely susceptible to eavesdropping or insertion attacks (e.g., a well-shielded cable or buttons), then trust in a third party is not necessary. If such an environment is not available, one can fall back to the trusted third party.
With this in mind, the following is one reasonable asymmetric scheme where principals have public/private key pairs (PK,SK). For owner A to send a command CMD to device C, A sends the message $m, \sigma(h(m))$ where $h()$ is a CR hash function such as SHA-1, $\sigma(x) = \text{sign}_{\text{SK}_\text{A}}(x)$, and $m = (r, \text{CMD})$ where r prevents replays. The device must either be initialized with the first owner's public key, or it must trust a third-party certificate authority to establish trust. The r value could be a monotonically increasing nonce or a timestamp. If a nonce, the device will have to remember the last nonce seen. If a timestamp, the principals will need synchronized clocks and/or a replay cache. Upon receiving a command, the device must verify that it's not a replay and that the signature is valid.
For a symmetric scheme, one could design something similar to Kerberos if you want a trusted third party. Or you could state an assumption that principals can establish a secure, mutually authenticated link (e.g., a cable or physical buttons). Each owner has a unique symmetric key for each device owned. $\text{MAC}_{K_{\text{A,C}}}$ is the symmetric key shared by A and C. For owner A to send a command CMD to device C, A sends the message $m, \sigma(m)$ where $\sigma(m) = \text{MAC}_{K_{\text{A,C}}}(m)$ and $m = (r, \text{CMD})$. The device is initialized with the first owner's shared key, or by a trusted third

party. The r value can either be a monotonically increasing nonce or a timestamp as in the asymmetric case. The device recomputes the MAC and verifies freshness.

(b) Describe a protocol explaining how "transfer of ownership" can be accomplished, under the public-key framework of your answer in part (a). That is, how can Alice instruct the device that Bob is now the new owner? (Remember that once this is done, the device should no longer respond to commands from Alice.)

Solution: To transfer ownership of device C from owner A to owner B, owner B sends its public key to A. This must be done over a secure link (e.g., a cable) or B and A must have a CA to introduce them to each other. Otherwise, A cannot trust that B's public key actually speaks for B. A then executes the command "CMD = XFER, B, $\mathrm{PK_B}$." This causes the device to replace $\mathrm{PK_A}$ with $\mathrm{PK_B}$, effectively revoking the old owner and enabling the new owner.

It's important that A checks that B indeed is B. Otherwise, an adversary could pretend to be B during the request for ownership change. Most submitted solutions overlooked this PKI problem. Remember, digital signatures do not solve all the problems of MACs. You still need either a secure link or a trusted party like a CA. Using a plaintext-aware cryptosystem is even better.

(c) Describe a protocol explaining how "transfer of ownership" can be accomplished, under the classical (symmetric-key) framework of your answer in part (a), similarly.

Solution: Several students used the Diffie-Hellman key exchange. Unfortunately, this is not part of the set of tools for "symmetric" key cryptography. DH uses asymmetric cryptography. Moreover, DH is subject to man-in-the-middle attacks. The previous owner could perform this attack. A Public-Key Infrastructure (PKI) could prevent the man-in-the-middle attack, but then you must trust a third party. The following protocol works if you explicitly state two assumptions: The new and old owners establish a secure link (prevent eavesdroppers and impersonators during this stage). You must also explicitly state that the new owner and device can establish a secure connection free from impersonation and eavesdropping by the old owner. Otherwise you need a trusted third party similar to Kerberos.

The new owner B sends over a secure link to the old owner A the message (B, H) where $H = h(\mathrm{K_{B,C}})$ where h is a secure hash function and $\mathrm{K_{B,C}}$ is the symmetric key that the device will eventually receive. This works because we assume no one can impersonate B.

The old owner A sends to the device C the command "CMD = XFER1, B, H". This tells the device to only accept the next command over a secure link iff the initiating party knows a pre-image of H. This prevents other parties from becoming the owner. Note, this does not have to take place over a physically secure link if you MAC the message as described earlier.

The new owner B then connects to the device C over a physically secure link (once again, something like a cable or shielded IR). It sends the command "XFER2, $\mathrm{K_{B,C}}$." The device verifies that this key is a pre-image of H. For future commands over physically insecure links, the device only accepts commands MAC'd with this key. This effectively revokes the old owner's access. Only B knows the pre-image.

There are several variants to this problem, but in all cases you must explicitly state

your trust assumptions.

(d) Compare the public-key and private-key approaches to this problem, in terms of efficiency, security, and flexibility.

Solution: There are some factors to consider:

1. Trust establishment: In both schemes, devices must either have pre-established trust in keys over physically secure links or have a trusted third party to introduce principals to each other. In the asymmetric case, you need a PKI. In the symmetric case, you need something like a Kerberos Key Distribution Center (KDC).
2. Performance: Asymmetric protocols tend to be much slower than symmetric protocols.
3. Storage: In the symmetric protocol, each owner-device relationship requires its own key. Therefore, a owner will consume O(n) space where n is the number of devices it owns. In the asymmetric protocol, the owner uses the same private key to authenticate to multiple devices. The key storage space is O(1). Symmetric keys tend to be smaller than asymmetric keys though (think 128-bit symmetric vs. 1024-bit asymmetric).
4. Atomicity: Some of the symmetric schemes submitted were not atomic. Ownership transfer was a two-step process. Most of the asymmetric schemes were atomic, given the public key of the new owner.

One of the more important aspects to recognize is that both the symmetric and asymmetric schemes require trust in either a third party or physically secure communication between devices and owners during transfer commands.

Problem Q-3. Palladium? But I just met 'em. [25 pts]

Following is the original solution submitted by Ajay Sudan.

In this problem, limit your write up to three pages total. Additional pages will be ignored.

For each of the four key components of Palladium (curtained memory, sealed storage, secure input/output, and attestation):

(a) Describe very briefly what the component does.

Solution: From the MIT Palladium presentation abstract available at:
`http://cryptome.org/palladium-mit.htm`
and `http://theory.lcs.mit.edu/theory-seminars/LaMacchia-abs.html`

- Curtained memory. "The ability to wall off and hide pages of main memory so that each 'Palladium' application can be assured that it is not modified or observed by any other application or even the operating system."
- Attestation. "The ability for a piece of code to digitally sign or otherwise attest to a piece of data and further assure the recipient of the signature that the data was constructed by an unforgeable, cryptographically identified software stack."
- Sealed storage. "The ability to securely store information so that a 'Palladium' application or module can mandate that the information be accessible only

to itself or to a set of other trusted components that can be identified in a cryptographically secure manner."

- Secure input and output. "A secure path from the keyboard and mouse to 'Palladium' applications, and a secure path from 'Palladium' applications to an identifiable region of the screen."

(b) Describe what vulnerabilities would be introduced if that component were removed from the design.

Solution:

- Without curtained memory, there would probably be no Palladium. Curtained memory is the cornerstone of Palladium and allows each Palladium application to reside in a "secure bubble" where its memory is secure and cannot be accessed by other applications. In the absence of curtained memory, an application would be able to read/write in another application's memory space. This is a huge vulnerability since proper memory protection for trusted applications is imperative. If an application's memory can be read, then all its secret data is vulnerable and can be stolen or overwritten. Curtained memory differs from current memory protection schemes in that even the operating system cannot observe or modify memory curtained by another application.
- Removing attestation should not create any vulnerabilities directly, but eliminates the user's ability to sign a piece of data as being authentic and coming from that user's Palladium-enabled PC. Some other method of authentication would be required in order to enable applications which rely on digital signatures to prove facts about the user's platform, operating system, etc.
- Sealed storage is similar to curtained memory in that it allows a Palladium application to restrict access to data only to itself. In the absence of sealed storage, a Palladium application would be unable to securely and persistently store sensitive information to disk. Curtained memory protects data in memory, but sealed storage is necessary to protect data on the disk. A Palladium application may need to save state to disk when it is not running, or if all its state does not fit into memory during runtime.
- Without secure input and output, a Palladium-enabled PC would become vulnerable to attacks which sniff on the input channels (keyboard presses, mouse movements, etc.) and attacks which try to sniff the output channels (grab video frame buffers and try to determine what's on the screen). Secure input is needed so that a user can securely communicate with a Palladium application without untrusted applications reading the user's input. Secure output is needed so that a trusted application can display data to the screen without an untrusted application figuring out what is being shown to the user. This is important because the information being shown may be of a sensitive nature.

(c) Describe one or two applications that would probably become infeasible (insufficiently secure) if that component were not present. (Or if you think this component is inessential for all applications, explain why.)

Solution:

- Digital-rights management (DRM) would become infeasible without curtained memory. DRM relies on loading sensitive data into memory for access by a trusted application, and the data must be kept out of reach from an untrusted program. Even if the data is encrypted in memory, at some point the data must be decrypted prior to display. If an untrusted program is able to access this data, then the copyrighted data can be captured and re-distributed.
  Curtained memory also helps "protect software from software" and applies to viruses, worms, etc. If a virus is able to alter data in memory which is supposed to be protected, then a virus will be able to wreak havoc on the system.
- If attestation were to be removed from Palladium, then DRM applications would be made more difficult to implement, and possibly become infeasible. DRM relies on attestation to determine the user's identity and determine what access, if any, they should be granted to sensitive data. DRM applications also need to be able to authenticate the user's software configuration to make sure it is allowed to be given access to the protected data.
- In the absence of sealed storage, data that is to be protected could be read by untrusted applications. Sealed storage is important for many applications, including DRM. If you would like to store a movie or some other rights-managed data to disk, then it is important that it not be readable by another application. Almost all Palladium applications which require data to be stored to disk require sealed storage. One example is a secure e-mail client which provides the user with all the benefits of Palladium (curtained memory, attestation, sealed storage, and secure I/O). Without sealed storage, another application would be able to access all of the e-mail client's data which is stored on disk without being authorized to do so.
- Secure input is needed for applications which must receive data from the user via a secure channel. Any Palladium application which requires the user to enter a password or other sensitive data relies on secure input. If an untrusted application (i.e. a virus) is able to read the user's keyboard input and captures a user's login to an e-commerce or electronic banking site, then the virus could subsequently used the captured data to log in as the user and wreak havoc.
  Secure output is required in any application where information must be displayed to the user securely, without any chance that other programs are "listening" in. DRM applications would rely on secure output, as would applications where private data is being displayed (i.e. web browser displaying financial information, medical records, etc.)

(d) Describe one or two applications that would not be affected by the removal of that component, but which still require the remaining components. (If you think there would be no such applications, explain why.)

Solution:

- Curtained memory. There are no applications that would be unaffected by the removal of curtained memory. As mentioned above, curtained memory is the cornerstone of Palladium, and without it, the whole concept of a trusted platform dissolves. Without curtained memory, another application or the operat-

ing system can observe, and possibly modify, a Palladium application's memory contents. If this is the case, then it is impossible to create any trusted applications since they have no memory space in which to perform computations that are unobservable to other applications.

- Attestation. Removing attestation removes the ability to authenticate a user's software configuration. Many applications are still possible without attestation. A web browser with caching, cookies, etc., would require curtained memory, sealed storage, and secure input/output, but not attestation.
- Sealed storage. The removal of sealed storage would not affect applications which do not need to store any state to disk. A very simple secure web browser which does not cache pages, store cookies, etc., would not be affected by the removal of sealed storage. Another example is some sort of simple client application used to connect to a brokerage firm and view one's accounts online. If no state is stored on disk, then sealed storage is not necessary. These applications would still make use of the other components; curtained memory for preventing another application from observing confidential data, secure input and output to prevent another application from sniffing the user's input or the display, and attestation for providing digital signatures.
- Secure input and output. Secure input is not needed for an application which doesn't require any input from the user. Secure output is not needed for an application that doesn't care if another application analyzes the frame buffer and reads the display. One example of an application that requires the other components would be a Palladium version of Windows Update manager. This application would send a signed attestation to the Microsoft attesting to the system configuration. Some secret session key may be returned by the server which needs to be protected from other applications using curtained memory. The application may store some state to disk and require sealed storage to prevent other apps from reading the data.

Problem Q-4. Universal Translators [25 pts]

In this problem, limit your write up to three pages total. Additional pages will be ignored.

Here is a variant of the ElGamal encryption scheme where the g and y values are swapped during encryption.

Algorithm G: Key generation for user i:

1. Generate a large random prime p and a generator g of the multiplicative group Z_p^* of the integers modulo p.
2. Select a random integer $x_i, 1 \leq x_i \leq p-2$ where $\gcd(x_i, p-1) = 1$, and compute $y_i = g^{x_i} \bmod p$.
3. i's public key is (p, g, y_i); i's private key is x_i.

Algorithm E: Encrypting a message to user i:

1. Obtain i's authentic public key (p, g, y_i).

2. Represent the message as an integer m in the range $\{1, \ldots, p-1\}$.
3. Select a random integer $k, 1 \leq k \leq p-2$.
4. Compute $a = y_i^k \bmod p$ and $b = m \cdot g^k \bmod p$.
5. Send the ciphertext $c = (a, b)$ to i.

Algorithm D: Decrypting a ciphertext $c = (a, b)$ sent to user i:

1. Use the private key x_i to compute $g^k = a^{1/x_i} \bmod p$ where the fraction in the exponent is computed $\bmod(p-1)$.
2. Recover m by computing $b/g^k \bmod p$.

Vericosine, a spin-off company of a Verisine, has decided to go into the ElGamal universal translator business. You can assume all parties share and agree upon securely the same p and g values. In your solution, you must state any other properties you require of p and g that are not specified above.

(a) Warm up

Several of Vericosine's customers use this system for encryption. A bug in the implementation of step 3 in E caused k to be a deterministic function of only the message. Unfortunately, the customers cannot upgrade their encryption (it was apparently built into the SSC hardware of a Palladium chip). Thus, if a sender encrypts the same message to two different recipients, an adversary can detect that the messages are the same.

Seeing a new source of revenue, Vericosine explains that users could send their ciphertexts (over a secure connection) to a re-randomizer before sending the final ciphertext over an untrusted network to the recipient. This could be implemented in software near the chip.

Using the notation above, explain how Vericosine can re-randomize ciphertexts generated by E. That is, construct a function R that takes as input the ciphertext (a, b) to produce a uniformly re-randomized ciphertext (a', b') such that $(a, b) \neq (a', b')$ and $D(R(a, b)) = D(a, b)$. Explain why your re-randomization works. You can assume that the re-randomizer has a source of randomness. The re-randomizer does not have access to any user secrets. Make sure to specify the necessary security properties of anything you use.

> *Solution:* Prof. Rivest solved this problem on the blackboard during lecture. By computing $a' = a \times y_i^r \bmod p$ and $b' = b \times g^r \bmod p$ where r is random, a third-party can effectively re-randomize the ciphertext without knowing the original message, the original k, or any private keys. It's important to choose $r \in_R Z_q^*$ if $p = 2q+1$ and q is Sophie-Germain. Otherwise the distribution of r would not be spread evenly, i.e. some virtual k' numbers would appear more often than others over several random choices.

(b) Translation

Vericosine, tired of the mere re-randomization business, decides to enter the burgeoning ciphertext translation business. Vericosine decides that if it obtains the quotient $t_{j \leftarrow i} = x_i/x_j \bmod (p-1)$, then it can translate ciphertexts originally destined to user i into ciphertexts decryptable by user j.

Construct the translation function $T(a_i, b_i, t_{j \leftarrow i})$ which takes as input a ciphertext for user i and the translation quotient from i to j to produce a new ciphertext (a_j, b_j) which only user j can decrypt to the original message.

Explain why your scheme works. In particular, what property of this variant of ElGamal is necessary for your scheme to work?

Solution: The basic idea is that the translator should exponentiate the a value of the original ciphertext with the translation quotient. To translate a ciphertext (a, b) cipherable by user i into one decipherable by user j, the translator computes $(a^{x_j/x_i} \bmod p, b)$. This is possible since for each secret key x_i, we have that $\gcd(x_i, p-1) = 1$. This problem comes from a paper on proxy cryptography. We encourage you to read "Atomic Proxy Cryptography" by Matt Blaze and Martin Strauss (check Google) for the arguments behind why this method works.

Several students misunderstood the how multiplying "downstairs" affects exponents "upstairs." They assumed that by multiplying downstairs, the exponents upstairs would also multiply. This is not the case. Multiplying $g^a \times g^b$ results in g^{a+b}, not g^{ab}. Exponentiating the quantity g^a to b, however, results in g^{ab}.

(c) Divided between preparations

The Technical Advisory Board of only Vericosine (TABOO-V) proposes two methods of establishing the translation quotient on the Vericosine translation server, T. Below, the $\longrightarrow$ symbol means "sends to":

Preparation Un:

$i \longrightarrow j : x_i$

$j \longrightarrow T : x_i/x_j$

Preparation Deux:

$i \longrightarrow T : x_i$

$j \longrightarrow T : x_j$

(T then computes $x_i/x_j \bmod (p-1)$)

Compare and contrast these two preparations. What are the security ramifications? What are the trust models? Some issues you may consider include resiliency of the system to disclosure of secrets or partial secrets, collusion between various parties, the assumptions about T, the addition of more users, and intrusion tolerance. Use the notion i, j, T, W to represent user i, user j, the translator, and the rest of the world respectively.

Solution: The first scheme trusts that user j will complete the protocol. This gives user j more power than i. User j could simply skip town with i's secret key, leaving user i with nothing but a broken heart and a compromised key. The translator only learns the quotient though.

The second scheme trusts the translator with the secret keys, even though the translator only needs quotients of secret keys. This makes the translator scalable; it only needs one datum per user, rather than per pair, resulting in $O(n)$ versus $O(n^2)$ storage space respectively where there are n users. However, it requires absolute trust in the translator and makes the translator much more tempting to a would-be-attacker.

If there are multiple users, then the system becomes very brittle if a single private key is leaked. Any chain of quotients will topple like dominos if a single private

key in any of the quotients leaks. The translator can determine all the private keys given a single private key.
In both schemes, the "sends to" must also mean "sends to over a mutually authenticated, confidential, integrity-protected link."

(d) Enter the consultant

Gwen Bytediddle[1], a well respected and paid consultant, points out that the scheme could be strengthened by making private keys available to fewer parties. Unfortunately, Gwen was called away to fix a leaky encrypted pipe before she could finish. She left a hint that the two users should first agree upon a random number $r \in Z_{p-1}$.

(1) Using Gwen's hint, construct a "preparation trois" such that Vericosine can translate ciphertexts, but private keys such as x_i and x_j are minimally exposed when all parties are honest.

(2) Explain the new trust model and argue briefly why preparation trois is secure when all parties are honest.

(3) What problems from part (c) remain if parties collude?

Solution: One protocol that works (where r is a random number in Z_q^* that i and j have agreed upon):
Preparation Trois:

$i \longrightarrow T : x_i \times r \bmod p-1$
$j \longrightarrow T : x_j \times r \bmod p-1$
(T then computes $(r \times x_i)/(r \times x_j) \bmod (p-1)$, and the r's cancel)

Here are some of the factors to consider.
Users now have equal power. Users trust that other users will not collude with the translator to determine private keys. Users trust that the translator will not pretend to be a user to compromise a chain of quotients.
The brittleness from part (c) remains. If a single secret key leaks, then the translator can determine all the other secret keys in any connected chain of quotients.

(e) Bonus

For 1 point, explain in one sentence how the two companies can become one.

Solution: $\text{Verisine}^2 + \text{Vericosine}^2 = 1$

[1] A distant relative of Ben Bitdiddle

Problem Q-5. Survey [5 pts]

On a single page, answer both of the following questions:

(a) Describe your favorite material in 6.857. Why is it your favorite?

Solution: The favorite material chosen was rather diverse:

Topic	Popularity	Topic	Popularity
Cookie security	18	Real-world psets	2
Hashing	12	Human factors	2
PK crypto	12	Blind signatures	2
Number theory	9	Crypto attacks	1
Palladium	8	Secret sharing	1
ZK proofs	7	Case studies/failures	1
Primality testing	4	Top 10 vulnerabilities pset	1
Encryption	4	"P=NP"?	1
OTP, Two-time pad	3	"Theory"	1
Voting	3	Buffer overflows	1
PKI	2		

(b) What material or paper not yet presented in class would you most like to discuss in 6.857? Why? Limit your suggestion to a single topic. `http://web.mit.edu/6.857/www/references.html` may offer some help.

Solution: Here's what the class said:

Topic	Popularity	Topic	Popularity
"Real world" security	10	Future of security	1
Differential crypto	5	Peer-to-peer	1
SSH remote login	5	More number theory	1
Viruses	4	Secure distributed computation	1
Hacker methods	4	Computer related risks	1
Anonymous communication	3	.NET	1
Identity-Based Encryption	3	Videogame security	1
Biometrics, user authentication	3	Factoring	1
Usability	2	GM security	1
Social engineering (Anderson)	2	Ring signatures	1
Programming language security	2	Symmetric ciphers	1
Keystroke logging and attacks	2	TCP/IP	1
Kerberos	2	Operating system security	1
Timing attacks	2	Mobile phone encryption	1
Denial of service	2	Watermarks	1
Steganography	2	CD copy protection	1
XBox hacking	2	Covert channel	1
Quantum crypto	2	Crypto and cryptanalysis	1
Policy	1	Alert sites, emergency response	1
PRNGs	1	DNS security	1

If you want to give the most feedback, apply to be a 6.857 TA next year....

Problem Q-6. Academic honesty [5 pts]

Write a couple sentences to testify that you have not collaborated with anyone on this midterm and that you have cited all your sources. Affix your pretty signature to this statement.

Solution: We accepted all reasonable, signed statements that used "I" and not "We".

Signature: