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

Handout 31
November 15, 2001

Fall 2001 Midterm Solutions

1. This midterm is due on *Thursday, November 1, 2001* at the beginning of class. Late midterms will *not* be accepted. There are six (6) problems and a total of 100 points.
2. You are to work **alone** on this midterm. You may **not** talk to anyone about questions on this midterm, except for the 6.857 TAs and Prof. Rivest. Collaboration or copying are **not** allowed and will not be tolerated.
3. This exam is open book. You may use any printed or Web resource, provided that you credit the sources properly.
4. Type up your solutions, use separate sheets per problem (subparts of a problem can be on the same page), and put your name and problem number on each page. You will receive 1 point for each problem in which you follow these directions correctly.
5. 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.

Problem Q-1. File Systems and Hashing [19 pts]

In this problem we examine a simple read-only file system which has a single directory mapping filenames to database keys[1]. There are no inodes or access control.

Filename	Database key
foo	0x29085324
bar	0x50932152
⋮	⋮

Figure 1: The directory maps filenames to database keys.

Figure 1 shows how a directory might appear. An auxiliary database provided by a publisher uses a hash table (with chaining) to map database keys to file content. To read the file "foo" in the above example, the file system first maps the filename to the database key 0x29085324. The system then asks the database for the value associated with this key. A publisher creates this database and chooses the database keys.

Ben Bitdiddle has modified this file system to provide for integrity protection of file contents. In his file system, a database key is instead the cryptographic hash of the value pointed to. If the file "foo" contains the text "March20", then the database key is hash ("March20"). If one trusts that

[1] By a "key" we mean a key/value pair as in an associative array or hash table

the directory itself has integrity, then a file system can not only locate file data, but also verify the integrity of the file data by recomputing and checking the hash. Furthermore, such a file system saves storage by identifying files that match exactly. If several files have the same content, then the hashes will also match. As a result, the database must store only one key/value pair for all the matching files.

Ben wanted to use SHA-1 as the hash function, but Ben's management required that database keys have minimal length without endangering the security of file integrity. Help Ben select an appropriate hash function output size that balances the database key size with the strength of integrity protection. How might you parameterize the length of the hash output with the strength of integrity protection and the sizes of file system structures? Limit your essay to one page. You can assume there exists a CR hash function for any given output size.

Solution:

We are aware of two types of integrity problems, one malicious and one bad luck. A malicious publisher or intermediary can trick a user into accepting bogus content by finding a collision between any two files. To protect against this adversary, one must make the hash output size large enough to prevent a computationally-bounded adversary from finding a collision.

A non-malicious problem appears when two legitimate files hash to the same value. In this case, the file system may return wrong answers. Content will simply disappear. The expected number of files helps greatly in determining an adequate hash output size. Here the hash output size must only be good enough such that other factors are much more likely to produce wrong answers. For instance, IBM explains that alpha particles and cosmic rays can induce bit flipping soft errors inside registers[2]. As long as the probability of a cosmic ray or some other event damaging the hardware is less than the probability of a collision between any two files from a given number of files, then you are for all practical purposes safe. Failure is not preventable, but you can make it relatively harmless in the grand scheme of things.

To solve both these problems, one would set the hash output length as the maximum of the two cases.

In grading, we looked for well-written essays that gave a reasonable definition of "strength of integrity protection" and "file system structures." For instance, the strength could be characterized as the probability that an adversary who can make at most 2^n queries of the hash function will not likely find a collision.

Because the problem set said to assume a CR hash function existed for any output size, only the number of files were relevant to integrity. The actual size of the files are not relevant because the definition of CR does not care about file sizes. In reality, the size may have some effect on integrity (we are not aware of any though).

We also expected students to reference the birthday paradox to explain how an n-bit hash can fall to an adversary who can make $2^{n/2}$ hash queries.

Another cool thing to do is apply Moore's law so that your choice of a hash output size remains secure as expected computing power increases. One could also take into account the number of hashes the adversary can perform in a given time period.

[2] `http://www.edtn.com/news/june11/061198topstory.html`,
`http://www-3.ibm.com/chips/micronews/vol5_no2/swietek.html`

A few students misunderstood the definition of collision resistance. CR does *not* prevent collisions; they must exist by the pigeonhole principle. Rather, CR means finding a collision is difficult for a computationally-bounded adversary.

What about the motivation for this problem? People often ask about the SFS Read-only File System, "What if two files accidentally hash to the same database key? How do you deal with this type of corruption?" Our standard answer is that the chance of a cosmic ray flipping a bit in a register during database creation is greater than having an accidental collision in SHA-1. For more information about SFS and the SFS Read-only File System, see `http://www.fs.net/`.

Here is a well-written essay from Timo Burkard:

There are numerous ways in which corruptions can occur. One possibility is the database not retrieving the requested file, but some other file. Another source of error would be the database returning a corrupted file. In the latter case, the corruption can have two sources: some hardware/software error that altered the file in a random manner, or an adversary, who designed an altered file with the intention that it will have the same hash key. Our goal is to come up with a hash function so that if the check based on the recomputation of the hash sum succeeds, we can assume with a reasonably high probably that we have in fact the correct file.

I will outline how the length of the hash output has to be chosen in all of these cases.

First, let's discuss the most severe case where an adversary constructs an altered file that will have the same hash value. In the scheme described in the problem set, we will use the output of a CR hash function as the database key. Since CR implies weak CR, it is infeasible for an adversary to replace a specific file with some corrupted file without the hash check catching this corruption, assuming that the hash output is sufficiently long. Specifically, if the hash output consists of n bits, an adversary would have to construct 2^{n-1} files in expectation before he finds encounters a file that has the same hash value.

Therefore, we have to choose n large enough so that it takes the adversary very long until he can find such a file in expectation. The level of security can therefore be measured by the *number of files* that an adversary has to create until he ends up with a file that has the same checksum in expectation. Let's call that value s. The hash output should then have $\lceil \log_2(2s) \rceil$ bits, independent of the size of the file system.

Therefore, if we want to be reasonably sure that one specific file is not corrupted, we should choose a hash output of length $\lceil \log_2(2s) \rceil$, where s is the measure of security as defined above. A reasonable value would be 46, which corresponds to an adversary who could try 1 million random files per second having to keep on generating files for 1 year in expectation until he would find a match.

In the case that we want to protect our file system from any file being corrupted by an adversary, and we again assume that the level of security is measured in terms of the *number of random files* that an adversary would have to create until it would match any of the files in the directory, the hash output should have the size $\lceil \log_2(2s * f) \rceil$, where f is the number of entries in the file system. Notice that as f grows larger, it takes an adversary longer to check if a single random file matches, because he has to compare the hash value of that file to all f entries in the directory. However, if we assume that calculating the hash of a random file is extremely expensive compared to comparing the checksums of f files, it is reasonable to measure the effort that it takes the adversary to cause a corruption simply in terms of the number of hash values that he has to compute.

Second, let's consider the case of the database returning a corrupted file with a random corruption. In this case, a good measure of the strength of the integrity protection would be the maximum probability p that a corruption in a file goes undetected. Since we assume that the corruption is random, if the length of the output is n, the probability of a corrupted file having the same hash value as the original file is 2^{-n}. Therefore, if we want to the probability of a corruption in a file not to be noticed to be at most p, we should pick the length of the hash output to be $\lceil -\log_2 p \rceil$, independent of the size of the file system structures.

Third, let's consider the case of the database always returning a file that is not corrupted, but the database might return a different file stored in the directory instead of the one actually requested. Again, we measure the strength of integrity protection by the maximum probability that a corruption is not detected. Since the database returning a random file of all the database entries is equivalent to the database returning any random file, we can apply the same reasoning as in the previous case, and again, we should pick the length of the hash output to be $\lceil \log_2 p \rceil$, independent of the size of the file system structures.

Problem Q-2. Numbing Theory [20 pts]

Below we borrow from the excellent solution turned in by Cristian Cadar.

Ben Bitdiddle has proposed the following encryption scheme for his client, Perry Noyd. Perry has a public key consisting of a large prime p, a generator g modulo p, and a value g^a modulo p; where the secret key a is a randomly chosen value modulo $p-1$.

Ben proposes that Perry's friend Amy encrypt a message $m = m_1 m_2 \ldots m_k$ for Perry (where each m_i is a bit) as follows. For each bit m_i, Amy first picks a value b at random modulo $p-1$. She then sends Perry the pair (g^b, g^{ab+m_i}) of values modulo p. Ben says to Perry that this scheme is secure, based on the usual Diffie-Hellman assumption, thus justifying its inefficiency.

1. Ben has suggested to Perry that a should be chosen so that $\gcd(a, p-1) = 1$. Ben claims that this ensures that $g^a \bmod p$ is also a generator. Argue that Ben's claim is correct. Limit your answer to this subproblem to one page.

   Solution:

   In this part, we prove that if g is a generator mod p and $a \in \{0, 1, ..., p-2\}$ such that $\gcd(a, p-1) = 1 \bmod p$ then g^a is also a generator mod p.

   We know that g is a generator mod p iff $g^{\frac{p-1}{q}} \neq 1 \bmod p, \forall q|(p-1)$. Similarly, g^a is a generator mod p iff $(g^a)^{\frac{p-1}{q}} \neq 1 \bmod p, \forall q|(p-1) \iff g^{\frac{a}{q}(p-1)} \neq 1 \bmod p, \forall q|(p-1)$.

   But since g is a generator $\Rightarrow (g^x \equiv 1 \iff x \equiv p-1)$. So, for any $q|(p-1)$, $g^{\frac{a}{q}(p-1)} \equiv 1 \iff \frac{a}{q}(p-1) \equiv p-1$. But from $\gcd(a, p-1) = 1$ and $q|(p-1) \Rightarrow q \not| a$ and so $\frac{a}{q}(p-1) \not\equiv p-1$, and so $(g^a)^{\frac{p-1}{q}} \not\equiv 1, \forall q|(p-1)$ and so g^a is a generator mod p and our proof is finished.

2. Explain how Perry can decrypt Amy's message. Limit your answer to this subproblem to one page.

   Solution:

   Perry can easily decrypt Amy's message. Let's consider a pair $(g^b, g^{(ab+m_i)})$ that Amy sends to Perry. Since Perry knows g^b and also knows a he can immediately compute $(g^b)^a = g^{ab}$.

After that, he computes the inverse modulo p of g^{ab} (by using Extended Euclid's algorithm for example) and multiplies Amy's message by this number. He obtains:

$$g^{(ab+m_i)} \cdot \left(g^{ab}\right)^{-1} = g^{ab} \cdot g^{m_i} \cdot \left(g^{ab}\right)^{-1} = g^{m_i} = \begin{cases} 1, & \text{if } m_i = 0 \\ g, & \text{if } m_i = 1 \end{cases}$$

If the result of his computation is 1, Perry will know that m_i was a 0. Similarly if the result is g, he will know that m_i was a 1. In order to decrypt the message sent by Amy, Perry will repeat the procedure above for every bit m_i.

3. Explain why Ben's proposed encryption is not very secure; it leaks information about m to an eavesdropper. Limit your answer to this subproblem to one page. (Hint: consider quadratic residuosity)

Solution:

We allowed students to assume that a was odd from the first part. But the problem is still solvable when a is even. One can check whether the public value g^a is a QR to determine if a is even or odd. Solutions that explained how to retrieve part of m got partial credit.

Ben's scheme is not secure. We will prove here that an attacker can easily reconstruct the message m.

First of all, an attacker knows that a is odd, since $\gcd(a, p-1) = 1$ (and $p-1$ is even).

We will solve this problem by using the concept of a quadratic residue (QR). A number x is a QR modulo p iff there is a y modulo p such that $x = y^2 \bmod p$. To determine if x is a QR mod p, we use the theorem that says that x is a QR mod p iff $x^{\frac{p-1}{2}} = 1 \bmod p$ where p is prime.

Let's consider the case where $x = g^v$, where g is a generator mod p. If g^v is a QR, then there is a y modulo p such that $g^v = y^2 \bmod p$. But since g is a generator modulo p, we can write y as $y = g^w \bmod p$. So, if and only if g^v is a QR modulo p, than there exists a w such that $g^v = g^{2w} \bmod p$, so we know that there exists w such that $v = 2w$, namely v is even.

Let's now consider a particular pair $(g^b, g^{(ab+m_i)})$ sent by Amy, pair that is intercepted by the attacker. Then, by applying the procedure described above, the attacker can find the parities of b and $ab + m_i$. As we said above, the attacker also knows that a is odd.

Then, he will find himself in one of the following situations:

$$
\begin{aligned}
ab + m_i \text{ is even and b is even } &\Rightarrow m_i = 0 \\
ab + m_i \text{ is even and b is odd } &\Rightarrow m_i = 1 \\
ab + m_i \text{ is odd and b is even } &\Rightarrow m_i = 1 \\
ab + m_i \text{ is odd and b is odd } &\Rightarrow m_i = 0
\end{aligned}
$$

Please note that here we are using the fact that $ab + m_i$ is computed modulo $p - 1$ and thus, if we know the parities of a and b, we can find the parity of ab modulo $p - 1$ (since $p - 1$ is even). Thus, by just finding the parities of b and $ab + m_i$, an eavesdropper can reconstruct the entire message m.

Problem Q-3. Count Transitivania [20 pts]

In the country of Transitivania, each citizen is required to vote and each citizen is considered eligible for the office of President. Voting is interpreted "transitively", so that if A votes for B, and B votes for C, then it is as if A voted for C directly (and so on, if C has voted for someone else; the chains get followed to the end). A citizen who wishes to be President votes for himself, stopping a chain. If a nontrivial "cycle" develops (e.g., A votes for B, who votes for C, who votes for B) then all the relevant votes are discarded.

1. Sketch an approach towards implementing the Transitivanian voting system that maintains voter privacy as well as you can. Limit your answer to this subproblem to one page. (You may answer however you like, but you might consider as a hint the homomorphic voting scheme described in class.) You need not implement "candidate privacy" (keeping secret who has voted for themselves).

   Solution:

   The simplest approach to this problem is to use a variation on the homomorphic encryption scheme presented in class, which can be used as a subroutine.

   Denote the set of voters as $V = \{V_1, \ldots, V_n\}$. The first round of voting proceeds as usual for homomorphic encryption, and the (decrypted) tallies $t_1^{(1)}$, ..., $t_n^{(1)}$ are computed. (The encrypted tallies are decrypted by the election authorities.)

   A second round of voting then occurs, where the original ballot of voter V_i is replicated $t_i^{(1)}$ times instead of once. (If $t_i^{(1)} = 0$ then voter i's original ballot is not used at all here.) A second round of tallies $t_1^{(2)}$, ..., $t_n^{(2)}$ are computed as before.

   This proceeds for $n - 1$ rounds, as the longest chain has at most $n - 1$ edges.

   The authorities determine the winner using the final tallies $t_1^{(n-1)}$, ..., $t_n^{(n-1)}$ to be the voter with the highest final tally who has voted for himself. (The authorities need to decrypt an entry of some of the original ballots to determine which potential winners have voted for themselves.)

   No individual voter's ballot needs to be completely decrypted for this to work. The operations of the election officials are publicly verifiable, as a consequence of the public verifiability of the underlying homomorphic encryption scheme, assuming that the tallies from each round are published.

The privacy of individual voters is compromised only to the extent that linear algebra can be used on the tallies from each round to solve for the votes of an individual voter. If this is a concern, the tallies could be kept secret (but then we lose public verifiability).

Other approaches are possible.

2. What do you think of "transitive voting"? What are the advantages and disadvantages of transitive voting? Limit your discussion of this subproblem to one page.

   Solution:

   There are many pros and cons for transitive voting. Here are some of each.

   Pros.

   - A voter can choose a "representative" or "proxy" to vote for him. The proxy could be a political party representative, a relative, a colleague, a news analyst or a political pundit.
   - Transitive voting may encourage smaller political parties to form.
   - Requiring everyone to vote is probably in principle a good idea.

   Cons

   - The scheme doesn't distinguish between the cases where A wants B to be president and A wants B to be his proxy to vote for president. These are rather different, and should be distinguished. (Bob may let his wife proxy for him, but wouldn't want his wife to be president.)
   - A voter doesn't know ahead of time where his vote will end up; his candidate may vote for someone else.
   - Large numbers of votes may be lost due to cycles.
   - If a candidate doesn't vote for himself, all of the votes for him go somewhere other than where the voters intended.
   - It may encourage vote-buying of votes of proxies who have large numbers of votes.
   - It is unrealistic to expect everyone to be able to vote, and a missing vote can cause large problems in this scheme.
   - Transitive voting seems a bit too complicated for real life.

Problem Q-4. In the know [20 pts]

In many password-based login systems, the server maintains the hash of a password, $h(p)$, and accepts a login if the user is able to produce a preimage, p, for this hash value.

In one of the zero-knowledge protocols that we have studied, a Prover is able to prove to a Verifier that she knows the discrete log, a, for some public value $y = g^a \bmod p$.

Describe the differences between a password, an interactive zero-knowledge proof of knowledge, and a non-interactive zero-knowledge proof of knowledge. Be sure to distinguish between the first and third items. Limit your discussion to one page.

Solution: Most answers pointed out that while all three solutions prove a theorem to the verifier, the zero-knowledge (ZK) and noninteractive zero-knowledge (NIZK) protocols only reveal the truth of the theorem and nothing else. The password scheme, unfortunately, reveals the preimage to the verifier, which might allow a verifier to latter impersonate the prover.

Even so, a remarkable similarity between NIZK proofs and Passwords is that both can be replayed by the verifier. Therefore, even though NIZK proofs do not reveal the secret, they are just as prone to impersonation attacks as passwords. Other key points for the answer are:

1. ZK and NIZK proofs are more computationally involved than passwords.
2. NIZK and Passwords are one-round protocols, whereas the flavor of ZK that we have discussed is at least three rounds (more rounds are required for better soundness).
3. NIZK and Passwords both make integral use of hash functions. For NIZK, we need collision resistance; for passwords we need one-wayness.

Problem Q-5. Block Cipher [20 pts]

Ben Bitdiddle has designed a simple and fast block cipher algorithm. A diagram of his cipher is presented below. The cipher encrypts blocks of n bits and requires a key of size $3n$ bits. The key is split into three n-bit subkeys, k_1, k_2, k_3. The input block is first XORed with k_1, then added to k_2 and multiplied by k_3. The addition and multiplication operations are performed $(\text{mod} 2^n)$. It is important to note that k_3 must be odd in order to be able to decrypt properly. In order to guarantee this, the low order bit of k_3 is first ORed with 1.

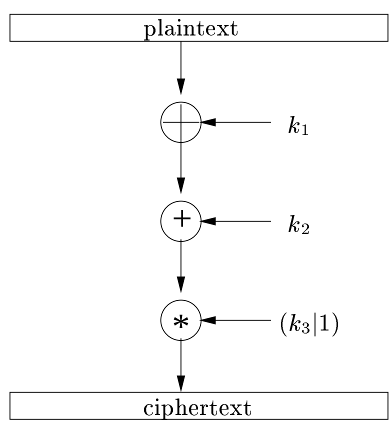

Figure 2: Ben's block cipher

Decryption works by performing the operations in essentially reverse order. In the first step, multiply the ciphertext block by $(k_3|1)^{-1}$ (that is, find the inverse of $(k_3|1) \bmod 2^n$ and multiply by this value), and then subtract k_2. Finally, XOR the resulting value with k_1 to get the plaintext.

Is this block cipher secure against a chosen-message attack? That is, if you are allowed to ask Ben for encryptions of arbitrary messages, are you able to mount an attack that recovers the secret key bits? If so, present the attack. If not, justify. Limit your response to one page.

Solution: Tara Andrews contributes the following slick solution: This block cipher is not secure against a chosen-message attack. I claim that given the encryptions of three messages $0, 1$, and $2^n - 1$, I can recover the effective public key. (I will not necessarily know the last bit of $k3$, but this does not matter for the purposes of the cipher.)

The value of $k3$ can be narrowed down to two possibilities by comparing the output of $C(0)$ and $C(1)$. Since $k1 \oplus 0$ is always $k1$, and $k1 \oplus 1$ is always $k1 \pm 1$, I know that the value of the cipher will have changed by ± 1 before it is multiplied by $k3|1$. Therefore $(k3|1)$ is $\pm(C(0) - C(1))(\pmod 2)^n)$. Yay linearity.

Now I find the multiplicative inverse, $m3$, of our two $k3|1$ values, and multiply each of $C(0)$ and $C(2^n - 1)$ by $m3$. By a property of binary representation, I know that $k1 \oplus (2^n - 1)$ is equal to $(2^n - 1) - k1$. Hence, the values $C(0)$ and $C(2^n - 1)$ provides two equations in two unknowns:

$$
\begin{aligned}
k_1 + k_2 &= C(0)/m_3 \pmod{2^n} \\
(2^n - 1) - k_1 + k2 &= C(2^n - 1)/m_3 \pmod{2^n}
\end{aligned}
$$

I now have two sets of equations that can be solved for $k1$ and $k2$. I can use our two sets of values $(k1_i, k2_i, k3_i)$ to find my encryption $C_i(1)$. Whichever one matches the actual $C(1)$ is the winner.

Problem Q-6. Academic honesty [1 pt]

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:

This problem was self-explanatory. Students who followed the directions got the point.