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

Handout 19
November 13, 2003

Problem Set 5 Solutions

Problem 5-1. Watermarking

Problem 5-1 was not supposed to be a difficult problem. Nevertheless, most of the groups did not do well on it. Your TAs were generally dissappointed by the lack of critical thinking that was evidenced on this problem. Many groups gave "estimates" for the items in part (a) that were nothing more than wild guesses. Many groups suggested approaches in part (b) which had been specifically discussed in class as being technically poor.

Grading: Part (a) was worth 5 points. Our plan was to remove one point for each significant technical error, sloppiness, and other problems. But some groups made so many errors that we couldn't hold them to this, otherwise they would have received no credit at all.

Common errors that were seen in part (a) include:

- **Watermarking Shakespeare with Spaces.** Many students didn't bother to survey Shakespeare's plays to find out how many lines there actually are! Some students assumed that the average line length was 80 characters (it isn't) and calculated the number of lines by dividing the size of the play by 80. Other groups guessed wildly. One group Googled for a reference which they then misinterperted — they were off by two orders of magnitude. In fact, depending on the play and how it is formatted, there are between 2000 and 5000 lines in the typical play. You could determine this by using the Unix `wc` command or the "Word Count" feature built in to Microsoft Word (which also reports lines).
- **Watermarking photos by adding 1 to pixel values.** Many groups erroneously asserted that the mark is detectable, because adding 1 to a pixel with a 255 value would cause the pixel to wrap to 0. But this can easily be avoided — because you have the original, you can *decrease* the color value to 254 in this special case (this is what we did in Problem 5-3). In any case, the watermark is easily removed: use a lossy compression scheme (like JPEG) to "smooth out" these small jumps, or re-encode the picture with a slightly smaller color palette. Either way, the picture will retain almost all of its quality, but the mark will disappear.
- **Watermarking movies by dropping frames.** There were wildly divergent opinions as to how many continious shots are in a typical movie. One group actually watched the first 10 minutes of *The Matrix Reloaded* and counted. Another group correctly noted that different directors would have different averages. Most other groups just guessed.

  Many groups asserted that this watermark could not be removed. In fact, it could be removed by randomly dropping 1 or 2 frames at the beginning and end of every sequence. This would slightly affect the quality of the film (the sound would have to be dropped too), but it wouldn't be terrible.
- **Watermarking Java by loop rewriting.** An astonishing number of groups asserted that this would be a difficult watermark to add to a program because it would have to be done manually. In fact, rewriting one kind of loop to the other is a mechnical operation that can be done by a simple program. Compilers do this sort of work all the time.

  Few groups bothered to examine source code to find out how common loops actually are. It turns out that they are not terribly common — one group discovered that there are less than 50 loops in 10,000 lines of the Java code that they had previously written for 6.170.
- **Watermarking PowerPoint by changing ADD instructions to SUB instructions.** A few groups with knowledge of x86 assembler noted that rewriting an ADD instruction as a SUB instruction would require adding an instruction to negate the value of a register; the problem set should have specified `ADDI` and `SUBI` instructions to specify immediate operands. Of course, it is still possible to

watermark the PowerPoint executable in the way we specified; the operation would simply enlarge the code.

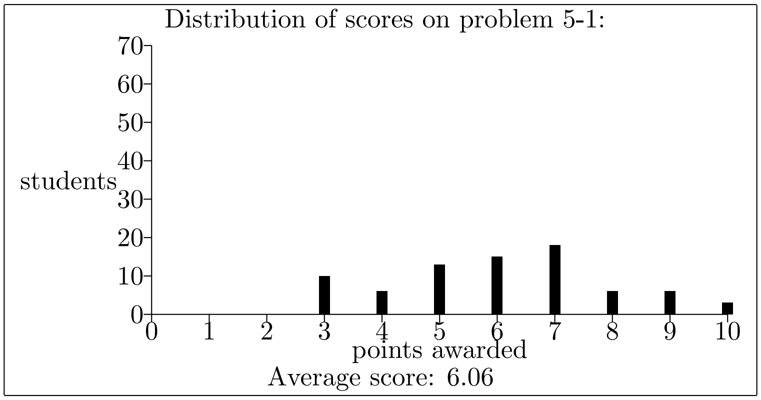
Groups once again gave wildly divergent guesses as to how many ADD and SUB instructions are in the typical program. "We assume that ADD instructions would be fairly common, 1 mark per 10 lines of machine code." Another group said that there would be an ADD instruction for "every few operations, since ADD is a very freqently used assembler operation." Another group argued that 50% of all instructions would be ADD instructions. Only one group actually bothered to examine the assembler output for some programs: they discovered that ADD instructions make up less than 1% of assembler opcodes. (To be fair, different programs will have a different opcode distribution.)

In part (b), three points were removed for the incorrect solution of "sign the file, the embed the signature into the file as a watermark." Once you embed the signature as a watermark, the file has changed and the signature will no longer verify!

A correct solution is to compute a digital signature (under some scheme for which the player knows the *public* key) for the first half of the file and to embed the watermark in the second half. Another solution is to remove the watermark from the file before verifying the signature, but this requires a special kind of watermark.

Some groups suggested using an HMAC to verify the authenticity of the MediaWare songs. This is a bad solution, as it requires embedding the same *secret* key in every MediaWare player. This then becomes a high-value secret key, and somebody will hack it out.

(Incidentally, the scenario envision in part (b) isn't so far-fetched: for example, Microsoft sells its X-Box video game system hardware at a loss, but makes up for it via licensing fees charged to game developers. The X-Box is essentially commodity hardware, but (unless specifically modified) will only play games that have been digitally signed under a specific key. One might imagine a similar regime for music and/or video, in which standard playback hardware is sold or given away at a loss, but is configured to only play media that is approved by the manufacturer.)



Problem 5-2. Zero Knowledge (or is it?)

We were very pleased with the solutions to this problem — almost all groups provided very clearly-written, technically accurate answers. Our only complaints are with two minor (but common) technical errors:

- If Polly does not know the decryption key d, it is difficult to quantify exactly her probability of successfully fooling Vinnie. This probability is *not*, however, as low as 2^{-1024}, as many groups claimed. The reason is that there are factoring algorithms that work faster than random guessing, so in fact it is possible to compute d in something like 2^{333} operations. In any case, a cheating Polly still has a negligible chance of decrypting Vinnie's challenge message, which is why the protocol enjoys such strong soundness.

- Many groups claimed that the hardness of the *discrete logarithm* problem is what prevents a cheating Polly from fooling Vinnie. While it *is* true that if DLP were easy, Polly would be able to cheat, that does *not* necessarily imply that if DLP is hard, Polly cannot cheat. In fact, we need a slightly stronger assumption: that the *RSA problem* (i.e., finding eth roots) is hard.

The following solution was submitted by Alexandr Andoni, Dumitru Daniliuc, Tudor Leu, and Oana Stamatoiu:

(a) If Polly knows d, then she certainly can decrypt the message and send r back to Vinnie. Then, Vinnie sees that he got r back, and concludes that Polly knows d. Thus, if Polly knows d she can prove to Vinnie that she knows it with probability 1, which proves that the protocol is complete.

If Polly does not know d, then she has the task of decrypting a ciphertext ($2^e \pmod{n}$) encrypted with RSA. This is equivalent to breaking RSA (solving the RSA problem), and is believed to be computationally infeasible. Thus, Polly will answer incorrectly with a high probability, which proves that the protocol is sound.

Vinnie does not need to repeat the protocol several times in order to be convinced that Polly knows d, because the probability that Polly does not know d, but still answers correctly is very small.

(b) The protocol is:

$$V \to P : r^e \pmod{n}$$
$$P \to V : r \pmod{n}$$

The protocol is zero-knowledge, because Vinnie could simulate the entire transcript in the following way:

1. Generate a random number r.
2. Compute $k = r^e \pmod{n}$.
3. Output the transcript:

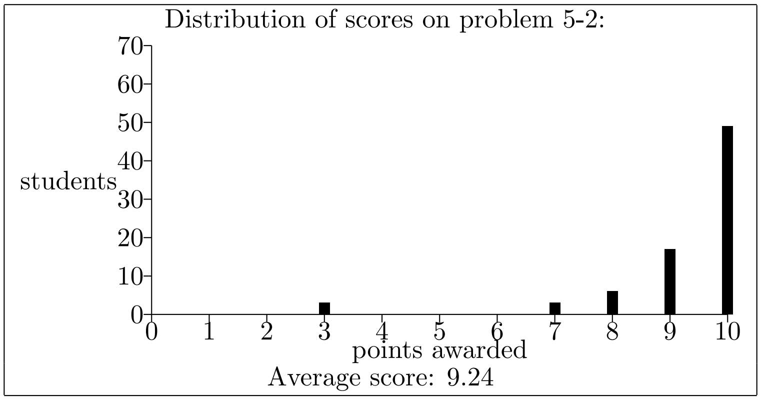
$$V \to P : k$$
$$P \to V : r$$

Since Vinnie can simulate the entire transcript, he can't even convince anybody else that Polly knows the secret key d.

(c) Let's prove that Vinnie can learn something that he cannot discover by himself.

Suppose Vinnie sends 2 to Polly, instead of sending $r^e \pmod{n}$. Then, since the key pair is generated with RSA, Polly will return $2^d \pmod{n}$. This information Vinnie could not learn by himself. In particular, he cannot simulate the entire transcript, because he does not know $2^d \pmod{n}$, and therefore, by definition, the protocol is not zero-knowledge.

Moreover, if Vinnie is eavesdropping on the network, then he can learn all messages sent to Polly — he just needs to ask Polly to decrypt every block of the message. In a similar way, Vinnie could impersonate Polly (by signing any messages in Polly's name, for example).



Problem 5-3. More Watermarking!

We were also very pleased with the solutions to this problem. Groups did a thorough job of analyzing the files and explaining the reasoning behind their attacks. Also, we learned a couple of interesting things about Herbert Yardley, thanks to multiple mini-biographies.

It turns out that the "best" way for a coalition to produce a pirate copy, no matter *what* code is being used (whether Boneh-Shaw, or a more efficient one), is extremely simple: at each marked location, the coalition chooses a fresh set of *three distinct members of the coalition at random.* Then, for the pirate copy, it uses the mark that is in the *majority* of those 3 users' files (i.e., the mark that appears at least twice). Of course, this strategy obeys the Marking Assumption. Notice also that for the Boneh-Shaw code this does exactly the right thing in blocks 1 and c, because only one user has a different mark than the others, and therefore those unique marks will never be used. In blocks $i = 2$ to $c/2$, this strategy produces blocks which have weight proportional to $i(i-1)/c^2$ (i.e., quadratic in i). In other words, the weights grow just slowly enough to prevent anyone from being implicated. In blocks $c/2$ through c, the weights grow more slowly, symmetrically with the first $c/2$ blocks.

The following solution was submitted by Conor Murray, Rui Viana, and Pallavi Naresh:

(a) Answer: either user 2 or 3 is missing

Our first strategy was to analyze the differences between the files on a bit level, i.e., bit by bit. However, in order to make life simpler, we wrote Java code that analyzed half a byte, or four bits, at a time, and wrote the differences to a output file. The first important realization came from the fact the at one position one of the files had a 0, or 0000 in binary, and another had an F, or 1111 in binary. We conjectured that if the marks were made at a bit level, it was highly unlikely that four marks would have been so close together, and therefore the marks must have been made at a byte level.

The next step consisted of modifying our Java code to analyze the files one byte at a time. Further confirmation that our hypothesis held was the fact that there were exactly 500 byte positions where at least one of the 5 files was different.

The observation of 500 hundred differences also suggested that users 0 and 5 were part of the coalition, otherwise we should have seen 400 differences.

We continued our analysis by noticing that in some of the positions where the files were different, America and Norway were alone, i.e., different from all the other countries, and therefore they must have been users 0 and 5, although we could not tell for sure which was which.

In our final step, we realized that at each of the positions in which files differed, one of the following was true:

- America was alone.
- Norway was alone.
- America and Germany were equal and different from the others.
- Norway and Canada were equal and different from the others.

Since we already knew that America and Norway were users 0 and 5 we had two cases left: either user 1 or 4 was not in the coalition, and either user 2 or 3 was not in the coalition. To be able to differentiate between these two cases, we excluded America and Norway from the analysis and counted the number of differences in the remaining files. If it was the case that either 1 or 4 was missing, we should expect to see 200 differences (corresponding to blocks 3 and 4. See the left case on the table bellow). If it was the case the either 2 or 3 was missing, we should expect to see 300 differences (corresponding to blocks 2,3 and 4. See the right case on the table bellow). We ran the code and observed 300 differences, and therefore either user 2 or 3 was not in the coalition.

User	B_1	B_2	B_3	B_4	B_5		User	B_1	B_2	B_3	B_4	B_5
0	x	x	x	x	x		0	x	x	x	x	x
1	-	-	-	-	-		1	0	1	1	1	1
2	0	0	1	1	1		2	-	-	-	-	-
3	0	0	0	1	1		3	0	0	0	1	1
4	0	0	0	0	1		4	0	0	0	0	1
5	x	x	x	x	x		5	x	x	x	x	x

Note that we cannot tell which one is the correct user, because we cannot say for sure when a mark is representing a 1 and when it is representing a 0. All we can do is observe that the mark is there.

(b) Our strategy for this part was going through the tracing algorithm and figuring out what had to be done in order for the algorithm to not accuse any of the colluders.

We started by assuming that user 3 was the one not in the coalition. This can be done because if we assume that user 2 is missing instead, the entire analysis is equivalent if we replace 0's by 1's and vice-versa.

In order to figure out which image corresponded to which user we counted the number of times America and Germany were together (100 times). This fact, together with the observations made in part (a), led us to the following assignments:

- America is User 0
- Germany is User 1
- France is User 2
- Canada is User 4
- Norway is User 5.

Fooling the Tracing Algorithm America and Norway could not be implied by the algorithm. Hence, in every position that one of these countries appeared alone we had to keep the byte from the other files. Thus, $weight(B_1) = 0$ and $weight(B_5) = 100$. Using the fact that $weight(B_1) = 0$, we were able to calculate that $weight(B_2)$ could be as large as 32 and User 1 would still not be implied. With this extra information we concluded that $weight(B_3)$ could be as large as 96 and User 2 would not be implied. However, we could differentiate between B_3 and B_4, therefore we had to deal with $weight(R_4 = B_3 \bigcup B_4)$, instead. Finally, we checked that as long as $weight(B_4) \geq 35$, the algorithm would not imply User 4.

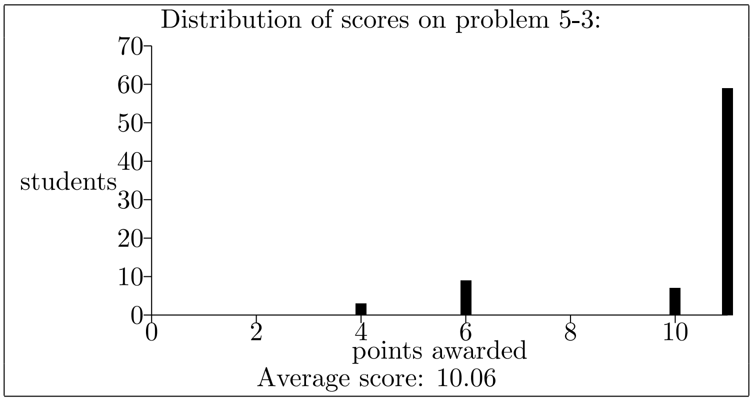
Here is a summary of the weights we assigned:

Block	Weight
B_1	0
B_2	32
R_4	130
B_5	100

The reason we chose $weight(R_4) = 130$ was that we cannot tell what $weight(B_3)$ and $weight(B_4)$ will be, but we want to minimize the chances that $weight(B_3) > 96$ and $weight(B_4) < 35$. Therefore, we chose the approximate average of 96 and 35, or 65, as the expected weight of each of these blocks. This choice minimizes the probability that either of the above conditions will fail (actually we did not care enough to check if the exact value was 64,65 or 66. We were happy enough to be this far from 35 and 96).

Our strategy for constructing a picture with these weights was just to assign 1's until we reached the correct weight and then assign 0's for the rest of the block (same thing for R_4). Note that at this point we could differentiate between a 0 and a 1. A 1 corresponded to the value in the America file and a 0 to the value appearing in the Norway file.

Finally, our picture was successful against the Tracing algorithm offered on the class web-site.


Distribution of scores on problem 5-3:
70
60
50
40
students
30
20
10
0
0
2
4
6
8
10
points awarded
Average score: 10.06