Problem Set 2

This problem set is due *online,* at `https://courses.csail.mit.edu/6.857/` on *Friday, March 4* by **11:59 PM**. Please note that no late submissions will be accepted.

You are to work on this problem set with your assigned group of three or four people. You should have received an email with your group assignment for this problem set. If not, please email `6.857-tas@mit.edu`. Be sure that all group members can explain the solutions. See Handout 1 (*Course Information*) for our policy on collaboration.

Homework must be submitted electronically! Each problem answer must appear on a separate page. Mark the top of each page with your group member names, the course number (6.857), the problem set number and question, and the date. We have provided templates for LaTeX and Microsoft Word on the course website (see the *Resources* page).

Grading: All problems are worth 10 points.

With the authors' permission, we will distribute our favorite solution to each problem as the "official" solution—this is your chance to become famous! If you do not wish for your homework to be used as an official solution, or if you wish that it only be used anonymously, please note this in your profile on the homework submission website.

Both of these problems have online components. The formal answers required are just clear writeups of your solutions, but you are strongly encouraged to implement your work and check them against the online components to make sure your answers work.

Problem 2-1. Cryptanalysis of RC4 with leakage

Note that this is an *open problem*, as far as we know – we don't know what the best solution is, or if it's solvable for all k. Please do the best you can and document your work, and we'll grade reasonably.

The stream cipher RC4 was described in class, and a detailed description can be found on the Internet. It has a 256-byte state $S[0\,..\,255]$, initialized from a random key, and array indices i and j initialized to zero. It then generates a stream of pseudo-random bytes, updating i, j, and S at each step, as follows:

```
repeat:
  i = i + 1     mod 256
  j = j + S[i]  mod 256
  swap S[i] and S[j]
  output S[S[i] + S[j]  mod 256]
```

As briefly mentioned in lecture, while it may seem that an array lookup $S[i]$ takes constant time, on modern processors this may not be true because of caching effects. Suppose that, either due to such a timing attack or some other side-channel attack (electromagnetic leakage?), you have the ability to intercept the high-order $k = 4$ bits of j at the end of each iteration through the loop. For example, if $S[1] = 47 = 00101111$ in binary when the loop begins (i.e., immediately after the key scheduling phase finishes), then at the end of the first loop, $i = 1$, $j = S[1]$, and you see the leaked high-order four bits 0010.

Given a long stream of RC4 output, with the corresponding leaked high-order four bits of j, can you reconstruct the final secret RC4 state, either in whole or in part?

Feel free to consider variations in which the top $k = 5$, 6, 7, or 8 bits are output, if that helps. (Eight bits should make this problem quite easy, so try to make some progress for $k < 8$. If you make good progress, can you solve for $k = 1$ bit of j output each time?)

On the course website's resources section, you can find 8 pairs of RC4 output bytes and high-order bits of j, for each value 1 – 8 of k. The low-order bits of j are just zeroed, so for the example above, you'd see 00100000 binary. For each of these, we've run RC4 for 2^{20} steps (1 megabyte). You should test your attack on these files.

Problem 2-2. The Advanced Bit-Diddling Standard

Last year Ben Bitdiddle proposed a block cipher that he thought was highly secure. He would have gotten away with it too, if it weren't for those meddling 6.857 students[1].

Ben has decided to make a few changes to his algorithm. As a first step, he decides to use the Feistel network design around his algorithm. He also eliminates one of the permutations q to avoid issues with weak keys (he hopes).

Here is a complete description of the resulting algorithm. The key is of the form (p, S) where p is a permutation of the integers $0 \,..\, 63$, and S is an S-box mapping every byte $(0 \,..\, 256)$ to some other byte. Note that in comparison to Ben's original algorithm, p is half the size, and S does not necessarily need to be invertible, because of the Feistel construction. The *Advanced Bit-Diddling Standard* (code-named "Bitdael") works as follows:

1. Split the input 128-bit message block M into two halves, $L[0 \,..\, 63] = M[0 \,..\, 63]$ and $R[0 \,..\, 63] = M[64 \,..\, 127]$.
2. Permute the bits in R according to p, i.e., $R_p[i] = R[p[i]]$.
3. Interpreting each group of 8 bits in R_p as a byte, substitute each byte $b_0 = R_p[0 \,..\, 7] \ldots b_7 = R_p[56 \,..\, 63]$ with S, yielding the string $R_S = S[b_0]|| \ldots ||S[b_7]$.
4. XOR the result into L, yielding a new value $L \oplus R_S$.
5. Set L to the current R and R to the current $L \oplus R_S$.
6. If there are more rounds remaining, continue from step 2. Otherwise return $L||R$ as the output of the cipher.

Ben thinks that 2 rounds is probably weak, but 3 rounds should definitely be secure, and more rounds would be too slow and not worth it. Give a chosen-plaintext key recovery attack on Ben's scheme for 2 rounds and (if you can) also for 3 rounds. That is, some remote party has a key, and you can send message blocks to that remote party and receive ciphertext blocks for them; your job is to determine what the key must be.

(Notes: in this cipher the table S may not be one-to-one. There are also many keys "equivalent" to a given key; changing p and S in corresponding ways can leave the cipher unchanged. So it is OK to find any key equivalent to the hidden key.)

We have prepared a webserver linked from the course website that will play the part of this party and let you encrypt text the Advanced Bit-Diddling Standard with a variable number of rounds. Instructions on how to use it (and a sample client library in Python) are also listed on the web; you can generate a key and encrypt up to 1000 plaintexts of your choice with that key, and then you're allowed to check your guess of what the key is. See how many rounds you can break.

[1] `http://courses.csail.mit.edu/6.857/2010/handouts/H03-ps2.pdf`