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

Handout 29
November 13, 2001

Problem Set 7

This problem set is due on *Tuesday, November 20, 2001* at the beginning of class. Note the abnormal due date. Late homeworks will *not* be accepted. You are to work on this problem set in groups of three or four people.

Mark the top of each sheet with your names, 6.857, the problem set number and question, and the date. **Type up your solutions** and be clear. Each problem should begin on a new sheet of paper. That is, you will turn in each problem on a separate pile of paper. **Cite** your sources of information.

Problem 7-1. Virus!

You have been hired by MegaScan, the leading provider of virus scanners, to write a special purpose subroutine to detect occurrences of the "UoyEvolI" virus, a particularly nasty polymorphic virus. Because of the way this virus inserts itself into the infected program, it isn't detected by the usual procedures (it doesn't execute when the infected program starts, but only some time later). Your subroutine will be run on all files to detect the UoyEvolI virus.

The UoyEvolI virus uses encryption to achieve polymorphism; the body of the virus is encrypted by exclusive-oring with a byte sequence S derived from an eight-byte secret key K that changes from instance to instance in a random way. The sequence S is derived by merely repeating over and over the given key K.

How would you approach this problem? What experiments would you run?

Problem 7-2. The DMCA Section 1201: Paracopyright

Niels Ferguson refused to release his break of HDCP[1] because of the Digital Millennium Copyright Act (DMCA). The Electronic Frontier Foundation (EFF) argues that abuse of the DMCA is forcing Niels to withhold research. If he publishes his break, he could be arrested when visiting the U.S. On the other hand, the American Association of Publishers (AAP) says that Niels is over-reacting.

Your job is to write two one-page essays. In the first essay, you are the senior intellectual property attorney for the EFF. Argue why Niels is not over-reacting to the DMCA. In the second essay, you are the vice president of legal and governmental affairs for the AAP. You are in favor of the DMCA. Argue why Niels is over-reacting.

Both the AAP and EFF representatives agree that in order to comply with the WIPO treaty, the U.S. government *must* implement adequate laws to protect against circumvention of copy protection technologies. Both representatives agree that the DMCA *had* good intentions. Beyond that, the representatives begin to disagree....

Each essay should give two clear points supporting the thesis. Essays should include an introductory paragraph, a concluding paragraph, and paragraphs to support your points. Each essay must fit on one page.

You'll probably have a harder time arguing the point of the AAP. However, there are legitimate points to both sides. You cannot simply disagree with the assignment[2].

[1] `http://www.macfergus.com/niels/dmca/cia.html`
[2] Debaters call this type of assignment "going swing."

Problem 7-3. It's time to rock around the clock

After 30 years of cracking ciphers for the NSA, Louis Reasoner Sr. decided to fly the coop to the private sector. Louis designed and implemented the "Egg" block cipher. Amazingly, the NSA permitted Louis to publish this algorithm[3]. Since Louis slept through the AES submission deadline, he was left with no choice but to exclusively license the Egg cipher to the Sunny Side Partners holding company which sub-licenses Egg to a dozen companies specializing in tamper-resistant smart cards.

The Egg cipher encrypts and decrypts messages as follows:

```
scramble (plaintext, key[])
// cost = 0
c = plaintext
for i = 1 to R
    c <- c XOR key[i]
    c <- f (c)              // cost += #ones in c
    c <- Sbox [c]
c <- c XOR key[R+1]
ciphertext <- c
// return cost too

unscramble (ciphertext, key[])
// cost == 0
p = ciphertext
p <- p XOR key[R+1]    // pre-whitening of Egg
for i = R downto 1
    p <- Sbox'[p]
    p <- f'(p)              // cost += #ones in p
    p <- p XOR key[i]
plaintext <- p
// return cost too
```

`f()` returns the dot product of an invertible matrix with its input in GF(2). `f'()` is the inverse. Sbox is a permutation. Sbox' is the inverse. Each round key is 64 bits. Ciphertexts and plaintexts are each 64 bits.

Alyssa P. Hacker, just laid off from the Egg project, muttered to Louis on her way out about implementing the `f()` and `f'()` functions to have running times that do not vary with the input. She also warned Louis about counting his chickens before they hatch, but that's another story. This subroutine for the `f()` and `f'()` functions was found scribbled on Alyssa's cube:

[3] The NSA has prior restraint on publications of former NSA employees.

```
// Precomputed ``weight'' of a byte.  If the number of one bits in
// the byte is odd, array entry is 1.  0 otherwise.  Idea from VRA.
private static final short weight[]={
    0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0,
    1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1,
    1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1,
    0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0,
    1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1,
    0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0,
    0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0,
    1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1,
    1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1,
    0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0,
    0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0,
    1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1,
    0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0,
    1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1,
    1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1,
    0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0
};
// Ybhvf vf n onfxrg pnfr sbe qrfvtavat gur rtt pvcure
```

Alas, Louis decided that the code to implement Alyssa's function was offensive and laid her Egg cipher implementation to waste in a basket. He opted for a function that runs in time linear in the number of one bits in the input.

One of the sub-licensees decided to use Louis' code in their impenetrable, tamper-resistant Egg Shell (TM) smart card. A person using the Egg Shell (TM) smart card can easily observe the running time of encryptions and decryptions respectively of chosen plaintexts and ciphertexts.

(a) Crack Egg

What is wrong with Louis' decision to implement the `f()` and `f'()` functions his way? Is this a rotten cipher?

(b) Coding time!

Implement a timing attack in the `eggTimer` class given in the sample source code on the 6.857 lecture references page. Your eggTimer class must be written in Java and return the key embedded in the eggSmartCard (with a very high probability) within a couple minutes on a reasonably modern machine. You must also write a `main` method. Your program should print the key in capitalized hexadecimal such that it is of this form:

```
0x71, 0x69, 0xCA, 0x35, 0xD1, 0x5E, 0xB3, 0x43,
0x9A, 0xCF, 0x15, 0x49, 0x1B, 0xCF, 0x0E, 0x1E,
0x1D, 0x22, 0x24, 0x37, 0xEC, 0x79, 0xC8, 0x24,
0xD1, 0x22, 0x42, 0x73, 0xCE, 0x97, 0x8C, 0x42
```

We will link your code against a 3-round version (256-bit key) of Egg. During our test, your code will have access to the methods exposed by the eggSmartCard Java class. You can time encryptions and decryptions, but there is no method to return the key itself. You can borrow code from the sample eggSmartCard.java file into your eggTimer.java file.

We will test your code by compiling our eggSmartCard class and then in the same directory running your code:

```
javac eggTimer.java
java eggTimer
```

Email your code with the subject "No egg on my face" to `6.857-staff@mit.edu` by the deadline. Also include a printout of the code with your turned in problem set. We will test your code on a new key, not the key given in the sample eggSmartCard.java file.

Since this problem involves a bit of probability, we give you several hints for an R-round version of the Egg cipher:

1. It's not necessary to use the values of encryptions or decryptions to find the key. Using just the timings, you can find the key...
2. You may assume the time for a single encryption is a random value between 0 and $64R$, inclusive.
3. The mean time for a single encryption is $32R$.
4. The variance of the time for a single encryption is $16R$.
5. The variance of the sum of a set of independent random variables is the sum of their individual variances.
6. You may assume that the inputs to the $f()$ and $f'()$ functions are independent (same for the outputs), if the plaintext encryption input is randomly chosen (same for ciphertext input for decryption).
7. A single random bit has mean $\frac{1}{2}$ and variance $\frac{1}{4}$.
8. The standard deviation of a random variable is the square root of its variance.
9. If you sample a random variable with mean μ and standard deviation σN times and take the average, the average has mean μ and standard deviation $\frac{\sigma}{\sqrt{N}}$.
10. Can you check a guess of a single key bit somehow?
11. You may (or may not) find Appendix C of Cormen/Leiserson/Rivest/Stein helpful on Probability and Counting. (This was Chapter 6 of the first edition by Cormen, Leiserson, and Rivest.)

That's all, folks! No more problem sets. We hope this last problem was eggucational and over easy.