Midterm Exam

CS381-Cryptography

October 30, 2014

Useful Items

$\oplus$ denotes exclusive-or, applied either to individual bits or to sequences of bits. The same operation in Python is denoted ^.

$$2^{10} \approx 10^3 = 1000,$$

and likewise for any (smallish) positive integer k,

$$2^{10 \times k} \approx 10^{3 \times k},$$

so, for instance, 2^{30} is about 10^9, or one billion. (In fact, a 'gigabyte' is not one billion bytes, but 2^{30} bytes.)

Security Benchmarks

- Through use of a custom hardware unit that costs about $10,000 to build, a 56-bit DES key can be recovered by exhaustive search in about a week.
- The record for factoring a product of two primes is a number of 768 decimal digits. The calculation, distributed over hundreds of processors, took two years.
- There are about 3×10^7 (30 million) seconds in a year.

Language statistics There are about 100 printable ASCII characters. In ordinary ASCII text in English (including punctuation and capitalization) the most common character by far is a space, accounting for about 18% of all characters, distantly followed by lower-case 'e', which accounts for about 9% of all characters. Note that this is different from the letter-frequency statistics we used earlier, in which we considered only the 26 letters and ignored punctuation and capitalization.

1 Insecure Encryption in a very old version of Microsoft Word

The earliest version of encryption in Microsoft Word (present in Windows 95) was to have the user select a password of printable characters, and then simply XOR repeated copies of the password with the plaintext. For instance, if the password was

```
hEre_15a*p@5SwD
```

which has 16 characters, the the first byte of ciphertext will be the first character of plaintext XORed with 'h', the second byte of ciphertext will be the second character of plaintext XOR'd with 'E', *etc.* Once the seventeenth charcacter of plaintext is reached, we return to the first character of the password and XOR with 'h', etc. In general, if the three strings are viewed as lists of bytes, we have, in Python:

```
ciphertext[i]=plaintext[i]^password[i % 16]
```

Assume all passwords have exactly 16 characters.

(a) How many different passwords are there? Is it feasible to carry out an exhaustive-search attack that recovers the plaintext from ciphertext by trying out all possible keys? There are three possible answers here: (i) it can be done on my laptop by a program that runs in under an hour; (ii) it can be done with specialized hardware and/or a widely distributed network of collaborating computers in under a year; (iii) it cannot be done with available computing power in under a century. Pick the best answer and justify with a rough calculation.

Solution. With an estimate of 100 printable characters there are

$$100^{16} = 10^{32} \approx 2^{32\times 3.3} > 2^{100}$$

keys as a very rough approximation. The best answer is *(iii)*: Using the DES cracker as a benchmark, we would need 2^{44} times the computing power. 2^{44} is about 16 trillion (one million million), so under the very implausible scenario that one million copies of the machine are available, you would need 16 million weeks or about three hundred twenty thousand years. (It is true that a single encryption for this cipher is just an XOR, and thus much faster than a DES encryption, but this is still not going to make the attack plausible.)

Common errors: Mild : using 256^{16} because you didn't restrict the password bytes to printable characters. Fantasy math version 1: writing something like 16^{100}, or getting 100^{16} correctly and then 'simplifying' it using magical laws of exponents of your own invention. Fantasy math version 2: getting something like the right magnitude for the number of keys, and then saying it wouldn't take all that long to count that high.

(b) Assume that the original plaintext is at least several thousand characters long, in ASCII characters, consisting of ordinary English sentences with spacing, punctuation,

and the like. You possess the encrypted version of the file. Describe an efficient procedure for recovering the *first character of the password*. (The same procedure, applied fifteen additional times, should recover the entire password, which can then be used to decrypt the file.)

Solution. Collect the first, 17th, 33rd, 49th, etc. bytes of plaintext. These are the bytes that were encrypted using the first key character. Find the most commonly occurring byte. XOR with the ASCII code for the space character. The result is the ASCII code for the first key character. This will hold unless you have some *very* weird text that does not use the space character in the conventional way, but that would not accord with the description in the problem.

You may have noticed the resemblance to the Vigenére cipher, but here the cryptanalysis is even easier: you don't have to worry about determining the length of the key because I gave it to you, and the statistics are so extremely skewed by the presence of the space character that you don't have to do anything fancy with the statistics. Most students recognized that this resembled the Vigenre cipher, but then tried to do much more work than was really required, for instance XORing the selected ciphertext bytes with every one of the 100 characters, and/or doing a fancy frequency analysis. (This was usually graded correct.) Some students did a brain dump about Vigenére cryptanalysis, but did not connect it with the problem at hand.

(c) Now suppose that documents are *not* ordinary English, but rather random, uniformly distributed sequences of characters. You have a brief plaintext document `a.doc`, its encryption `a_enc.doc`, and the encryption `b_enc.doc` of a long second document. Describe an efficient procedure for recovering the plaintext `b.doc`. (Assume the user, like most users, employed the same password for each document he encrypted.)

Solution. XOR the first sixteen bytes of `a_enc.doc` with the first sixteen bytes of `a.doc`. This is the key, which you can now use to decrypt the entire second document.

Many students recognized that this is the case of the one-time pad being used two times, and thus suggested computing the entire XOR of `a_enc.doc` and `a.doc`. This was graded as correct, but it's just about twice as much work as necessary, since given the structure of this cipher we only need to use 16 bytes of known plaintext.

2 Block Cipher in Cipher Feedback Mode

The accompanying diagram shows a block cipher mode of operation called *cipher feedback mode* (CFB). Like most of the other modes we have seen, there is an initial *IV* block that is generated at random and sent in the clear. Like CTR mode, this mode uses a block cipher as a stream cipher, XORing the plaintext with the keystream. Unlike CTR mode, the keystream cannot be generated completely from the IV and the key, since each keystream block depends on the preceding plaintext blocks.

Here are equations and a diagram describing the encryption algorithm. The inputs are the plaintext blocks $P_1, P_2, \ldots$, along with the IV block and the key K. E is the block encryption function. The outputs are $C_0, C_1, C_2, \ldots$, where $C_0 = IV$.

$$C_0 = IV,$$
$$C_i = P_i \oplus E(K, C_{i-1}),$$

if $i > 0$.

(a) Write equations for the decryption function, giving the plaintext blocks $P_1, P_2, \ldots$ in terms of $C_0, C_1, \ldots$.

Solution.

$$P_i = C_i \oplus E(K, C_{i-1}),$$

for $i > 0$. Notice that you don't need the block decryption function. (This is what happens as well with other modes, like CTR, in which the block cipher is used as a stream cipher.)

A few students were desperate to work the decryption function into this problem, and it is indeed possible to write an equation involving D, for instance

$$C_{i-1} = D(K, P_i \oplus C_i).$$

But this is not what was asked for, and is unhelpful when the task at hand is to determine the plaintext blocks from the given ciphertext.

(b) Suppose this encryption mode is used with a block cipher with a four-bit block size. The shared secret key is K, and the block encryption function E_K is given in Table 1. You receive a message

$$0000 \quad 1111 \quad 0000.$$

Determine the plaintext message. (Remember that the first block of the received message is the IV, so that there will be two plaintext blocks, not 3.)

Solution. We have

$$P_1 = C_1 \oplus E(K, C_0) = 1111 \oplus E(K, 0000) = 1111 \oplus 0011 = 1100.$$

$$P_2 = C_2 \oplus E(K, C_1) = 0000 \oplus E(K, 1111) = 0000 \oplus 1110 = 1110.$$

(c) This problem concerns what happens when you re-use the initialization vector in CFB mode. You receive two new messages that were encrypted with a different key K' (so that Table 1 is no longer relevant). The messages are

$$0000 \quad 1111 \quad 0000$$

$$0000 \quad 0000 \quad 1111.$$

Let P_1P_2 and $P_1'P_2'$ denote the corresponding plaintext messages. Describe *all* the information you can obtain about the blocks P_i and P_i' from this description.

As above,

$$P_1 = 1111 \oplus E(K', 0000)$$

$$P_2 = 0000 \oplus E(K', 1111) = E(K', 1111).$$

$$P_1' = 0000 \oplus E(K', 0000) = E(K', 0000)$$

$$P_2' = 1111 \oplus E(K', 0000).$$

This gives

$$P_2' = P_1.$$

$$P_2' = P_1' \oplus 1111,$$

in other words, P_1' differs from $P_1 = P_2'$ in every bit. What about P_2? It looks like we get no information, because we do not know what $E(K', 1111)$ is, but we do know that it is different from $E(K', 0000) = P_1'$. So we also have

$$P_2 \neq P_1'.$$

And this is as far as we can go, since any choice of P_i, P_i' consistent with the three displayed equations above is possible. One way to look at this problem is that in the absence of any information, there are $16^4 = 65536$ possibilities for the four blocks, but we are able to whittle this down to $16 \times 15 = 240$ possibilities because of the regularities in the ciphertext messages.

3 Number Theory and RSA

(a) Compute

$$101^{48,000,000,000,023} \bmod 35$$

by hand. (HINT: Use Fermat's Theorem and the Chinese Remainder Theorem. When you apply the CRT, you need not use the explicit formula for the solution of the two congruences; the numbers are so small that you can solve the system by inspection.)

Solution.

$$101 \equiv 1 \pmod 5,$$

so

$$101^{48,000,000,000,023} \equiv 1^{48,000,000,000,023} = 1 \pmod 5.$$

$$101 \equiv 3 \pmod 7,$$

so by Fermat's Theorem,

$$\begin{aligned} 101^{48,000,000,000,023} &\equiv (3^6)^{8,000,000,000,003} \cdot 3^5 \\ &\equiv 1^{8,000,000,000,003} \cdot 3^5 \\ &= 3^5 \\ &= 3 \cdot (3^2)^2 \\ &\equiv 3 \cdot 2^2 \\ &= 12 \\ &\equiv 5 \pmod 7. \end{aligned}$$

So we are looking for the unique solution to

$$\begin{aligned} x &\equiv 1 \pmod 5 \\ x &\equiv 5 \pmod 7. \end{aligned}$$

We can do this by inspection, checking 6,11,16, *etc.,* until we get a result congruent to 5 modulo 7. The correct answer is **26.** (Alternatively, you could apply the formula giving the explicit solution to this pair of congruences, but that is actually more work.) Typically, students failed to take advantage of all the tools that make it possible to solve this problem without ever having to write down a large number. (And even with that, some still managed to solve it correctly!)

(b) A public RSA key has encryption exponent $e = 3$ and modulus $N = 55$. Show all the steps in the calculation of the decryption exponent d.

Solution. The decryption key is

$$3^{-1} \bmod (10 \cdot 4) = 3^{-1} \bmod 40,$$

since since $55 = 5 \times 11$. You really could do this by simple trial and error, observing $27 \times 3 = 81 \equiv 1 \pmod{40}$. If you apply the extended Euclid algorithm, there is only one division:

$$40 = 13 \times 3 + 1,$$

so

$$3^{-1} \bmod \equiv -13 \equiv 27.$$

In either case we get the result **27**.

A common error was computing $3^{-1} \bmod 55$ instead of $3^{-1} \bmod 40$. Another was giving the answer as 13.

(c) Using your result from (b), determine how many multiplications and reductions mod 55 are required to decrypt a ciphertext C. (You do not have to decrypt anything, just understand the algorithm.)

Solution. To decrypt a ciphertext C you need to compute

$$C^{27} = C^{16} \cdot C^8 \cdot C^2 \cdot C \bmod 55.$$

We need four multiplications (squarings) and reductions to compute $C^k \bmod 40$ for $k = 2, 4, 8, 16$, and three more multiplications and reductions to multiply these four values together. So there are 7 multiplications and 7 reductions in all.

Some students just missed the business about repeated squaring entirely. Others knew that repeated squaring was involved but gave a generic answer rather than one specific to the parameters of this problem. A common, and much less serious error, was to neglect to count the three additional multiplications at the end.

And someone came up with the following ingenious solution: Compute $C^{27} \bmod 11$, which is the same as $C^6 \bmod 11$, and $C^{27} \bmod 5$, which is the same as $C^3 \bmod 5$, and then apply the Chinese Remainder Theorem.

(d) A friend who is curious about cryptography asks you why symmetric AES keys are only 128 bits long, while the decryption exponent and modulus for RSA keys are about 2048 bits long. Give a succinct answer.

Solution. Key length for AES is chosen to be resistant to brute-force attack, but the RSA modulus has to be large enough to resist *factoring,* which requires a much larger number. (128-bit integers can be factored efficiently, although not with the naïve algorithm of repeated trial division.)

Serves me right for asking an essay question. I got a lot of brain dumps with correct but irrelevant information (the fact that RSA is used in practice to encrypt keys, that there are various attacks against small RSA exponents or messages) as well as the occasional downright falsehood.

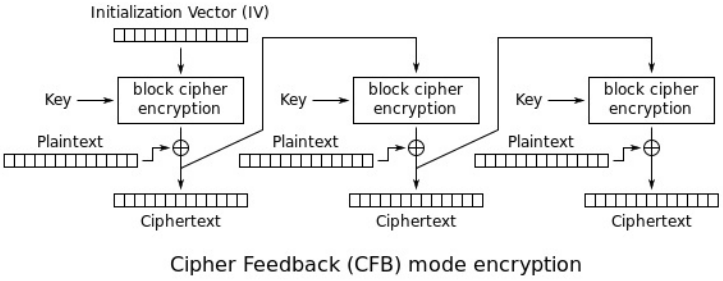

Figure 1: *Cipher Feedback Mode Encryption*

M	E(K,M)
0000	0011
0001	0001
0010	0111
0011	1100
0100	1001
0101	1111
0110	0000
0111	1010
1000	1101
1001	0010
1010	0101
1011	0110
1100	0100
1101	1000
1110	1011
1111	1110

Table 1: *Block encryption function for the block cipher in Problem 2, with respect to some fixed key K.*