COLLEGE OF COMPUTER AND INFORMATION SCIENCE

NORTHEASTERN UNIVERSITY

CSG252: CRYPTOGRAPHY AND COMMUNICATION SECURITY - FALL 2005

PROBLEM SET 1 - SOLUTIONS

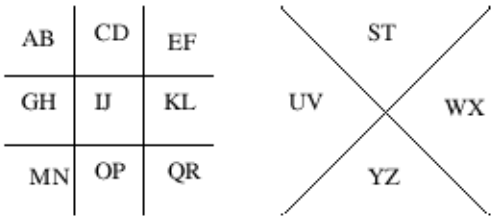
PREPARED BY TIM MORGAN

Problem 1 [10 points]:

(Solution contributed by Justin Sheehy.)

I recall using the "pigpen" scheme that was one of the many codes introduced to the Boy Scouts of America. It's a fairly straightforward code. The key is the alphabet in two grids:

(Sorry for the sloppy drawing, I did it quickly in xfig.)



You represent each letter by drawing the part of the diagram that surrounds it, adding a dot in the middle if it is the second letter in the box.

Example:

The letters do not need to be in the obvious alphabetical order in the "pigpen" diagram. In fact, the set of their positions in the diagram is effectively the key.

Use of this cipher is very traditional. The key diagram must be securely shared, and encryption and decryption are trivial clerical exercises for anyone in possession of the key.

Problem 2 [20 points]:

(a)

- 4457 mod 17 = $(262 \cdot 17 + 3)$ mod 17 = 3
- (-4457) mod 17 = $(-262 \cdot 17 - 3)$ mod 17 = -3 mod 17 = 14
- 3^{64} mod 17 = $(3^2)^{2^5}$ mod 17 = 9^{2^5} mod 17 = $(81 \bmod 17)^{2^4}$ mod 17

  = 13^{2^4} mod 17 = 16^{2^3} mod 17 = 1^{2^2} mod 17 = 1 mod 17
- 3^{65} mod 17 = $3^{2^6} \cdot 3^1$ mod 17 = $1 \cdot 3$ mod 17 = 3 mod 17

(b) *(Borrowed from last year's solution by Atilay Yilmaz)*

Assuming that b_1 is the most significant bit we can use the following algorithm*:*

power(a, b, c) :

```
p = 1;
b[1...n] = b1b2...bn
for i = 1 to n
        p = (p·p) mod c;
        if b[i] = 1 then
                p = (p·a) mod c;
  return p;
```

Problem 3(first) [30 points]:

(a)

Plaintext:

I may not be able to grow flowers, but my garden produces just as many dead leaves, old overshoes, pieces of rope, and bushels of dead grass as anybody's, and today I bought a wheelbarrow to help in clearing it up. I have always loved and respected the wheelbarrow. It is the one wheeled vehicle of which I am perfect master.

(b)

Key: CRYPTO

Plaintext:

I learned how to calculate the amount of paper needed for a room when I was at school. You multiply the square footage of the walls by the cubic contents of the floor and ceiling combined, and double it. You then allow half the total for openings such as windows and doors. Then you allow the other half for matching the pattern. Then you double the whole thing again to give a margin of error, and then you order the paper.

(d)

Cipher: VIGENERE

Key: THEORY

Plaintext:

I grew up among slow talkers, men in particular, who dropped words a few at a time like beans in a hill, and when I got to Minneapolis where people took a Lake Wobegon comma to mean the end of a story, I couldn't speak a whole sentence in company and was considered not too bright. So I enrolled in a speech course taught by Orville Sand, the founder of reflexive relaxology, a self-hypnotic technique that enabled a person to speak up to three hundred words per minute.

Problem 3(second) [20 points]:

(Borrowed from last year's solution by Atilay Yilmaz)

1. First, let' s recall the proof for n=1 (from class):

   For every prime *p* and integer *e*, $\phi(p^e) = (p^e - p^{e-1})$

   The set of elements that are below $p^e$and not relatively prime with p^e is: $\{p, 2\cdot p, ..., p^2, 2\cdot p^2, 3\cdot p^2, ..., p\cdot p^{e-1}\}$

   These are all the numbers that can be divided by p. The cardinality of this set is: p^{e-1}, therefore the number of elements below $p^e$and relatively prime with $p^e$is $p^e - p^{e-1}$.

2. Now, let's show that if *m* and *p* are relatively prime, then:
   $\phi(m \cdot p^e) = \phi(m) \cdot \phi(p^e)$.
   $\phi(m \cdot p^e)$=

   $|\{$all numbers$\leq (m \cdot p^e)\}|$

   - $|\{$all numbers$\leq (m \cdot p^e)$and not relatively prime to $m\}|$

   - $|\{$all numbers$\leq (m \cdot p^e)$and not relatively prime to $p^e\}|$

   + $|\{$all numbers$\leq (m \cdot p^e)$ and not relatively prime to m and not relatively prime to $p^e\}|$

   $= (m\cdot p^e) - (p^e - p^{e-1})\cdot m - (m - \phi(m))\cdot p^e + (p^e - p^{e-1})(m - \phi(m))$

   $= \phi(p^e)\cdot\phi(m)$

   (Here $|A|$ denotes the cardinality of set A.)

3. Now we can show the final result by induction.

   a. $\phi(m) = \prod_{i=1}^{n}(p_i^{(e_i)} - p_i^{(e_i-1)})$ obviously holds for *n = 1* (from the first step).

   b. Let's assume it holds for *n* and show that it also holds for *n+1*.

   If $m = \prod_{i=1}^{n+1} p_i^{(e_i)}$, let us denote by $m_1 = \prod_{i=1}^{n} p_i^{(e_i)}$.
   From the induction hypothesis, we have:
   $\phi(m_1) = \prod_{i=1}^{n}(p_i^{(e_i)} - p_i^{(e_i-1)})$
   we also have that:
   $\phi(m) = \phi(m_1 p_{n+1}^{(e_{n+1})}) = \phi(m_1)\phi(p_{n+1}^{(e_{n+1})})$
   Therefore:
   $\phi(m) = \prod_{i=1}^{n+1}(p_i^{(e_i)} - p_i^{(e_i-1)})$.

Problem 4 [20 points]:

1.

The binary representation of "test" is:

```
"t" = 0x74 = 0000 0000 0111 0100
"e" = 0x65 = 0000 0000 0110 0101
"s" = 0x73 = 0000 0000 0111 0011
"t" = 0x74 = 0000 0000 0111 0100
```

So the elements of the hash would be computed as:

```
shift15('t',1) = 0000 0000 1110 1000
shift15('e',2) = 0000 0001 1001 0100
shift15('s',3) = 0000 0011 1001 1000
shift15('t',4) = 0000 0111 0100 0000
             k = 0000 0000 0000 0100
       0xCE4B = 1100 1110 0100 1011
```

And the resulting XOR of these is:

```
xls_hash("test") = 1100 1011 1110 1011 = 0xCBEB
```

2.

a. The hash is in little-endian order as 0XEBCB

b. The hash appears to follow the bytes 0x0200 and precede a byte 0x55 (or 'U' in ASCII).

3.

Two passwords which happen to hash to the same value as "test" are:

"Pq2U"
"@o3P"

4.

The scheme is insecure for several reasons. First of all, anyone with a hex editor could alter data in the file directly, which bypasses the protection. Secondly, the hash itself is insecure, since it is easy to find pre-images using only pen and paper. Finally, even if pre-images were not easy to derive directly, the size of the hash allows one to brute-force it pretty quickly.

5. [10 points extra credit]

The easiest way to accomplish this is to search for the binary strings directly before and after the hash, and replace the hash with a known one for each sheet.