CO487 Assignment 1

Daniel Burstyn (20206120)

January 23, 2009

1. **Viginère Cipher**
   The secret key used to encrypt Page20 was **HAZELNUT**.

   The first approach I used to try and decrypt the text was to look at various frequency distributions of letters based on the possible lengths of the secret key. However, due to the small amount of characters, and the complexity of frequency analysis, I chose to use a different approach.

   I investigated the reapeated three letter blocks to see if I could determine the key length. Given that the size of the key is relatively small, and that we have a fair amount of ciphertext, we can conclude that a large proportion of the repeated blocks are due to the same word being encrypted by the same part of the key, and not just coincidences. Since the distance between the repeated blocks was always a multiple of 8, the key most likely had length 8.

   Knowing the length of the key, it was quite simple to retrieve a list of words in english, extract all 8-letter words with no repeated letters, and perform an exhaustive key search.

2. **Hill Cipher**

   (a) To decrypt ciphertext c, one only needs to compute $m = D(c) = (c - b)A^{-1}$ modulo 2.

   (b) In a chosen-plaintext attack, the attacker can start by choosing:
   $m = [0, 0, ..., 0]$ which gives $c = b$.
   Now with b, we can determine A using n more plaintexts. From the encryption function, we can easily see that if m is all 0s except for a 1 in the ith column, then c is the ith row of A. So, we can reconstruct A using all of these n plaintexts.

   (c) First observe that by the properties of exclusive or, if $a \oplus b = c$ then flipping the ith bit of a will also flip the ith bit of c.
   Now, consider the decryption routine outlined in (a): $D(c) = (c - b)A^{-1} \bmod 2$, which is equivalent to $D(c) = (c \oplus b)A^{-1}$ because of the modulo 2.
   If we take c, and flip it's first bit, then $c \oplus b$ will also have it's first bit flipped. Now let's investigate what effect this will have on the matrix multiplication:
   Suppose we have $d = [d_1, d_2, ..., d_n]$ and $A = \begin{bmatrix} A_{11} & \dots & A_{1n} \\ \vdots & \ddots & \\ A_{n1} & & A_{nn} \end{bmatrix}$, and $dA = [e_1, e_2, \dots, e_n]$
   By the properties of matrix multiplication, $e_1 = d_1A_{11} + d_2A_{21} + \ldots + d_nA_{n_1}$ and in general $e_i = \sum_{j=1}^{n} d_iA_{ji}$.
   From this we can see that if we were to flip d_i, then the new result would be e with bits flipped in the positions where the ith **row** of A has 1s.

   Thus by those two observations, we can conclude that by flipping a bit in the ciphertext, will result in flipping a distinct set of bits of the plaintext. Now, producing the real plaintext becomes quite simple:

Given the ciphertext $c = [c_1, c_2, \ldots, c_n]$ the three chosen ciphertexts should be:

$$c' = [\overline{c_1}, c_2, \ldots, c_n]$$
$$c'' = [c_1, \overline{c_2}, \ldots, c_n]$$
$$c''' = [\overline{c_1}, \overline{c_2}, \ldots, c_n]$$

With m', m'', and m''' as the respective plaintexts. Let U be the set of bits flipped in m', and V be the set of bits flipped in m''. The bits flipped in m''' should then be $U \cup V - U \cap V$. Observe that $m' \oplus m'''$ will be m except with V bits flipped (since it will reflip the bits in $U \cap V$. With $m' \oplus m'''$ and m'' both having only V bits flipped, xoring them will yield the original plaintext m.

In conclusion, in order to find the original plaintext m, the **three** chosen ciphertexts should be $c' = [\overline{c_1}, c_2, \ldots, c_n]$, $c'' = [c_1, \overline{c_2}, \ldots, c_n]$, and $c''' = [\overline{c_1}, \overline{c_2}, \ldots, c_n]$, yielding m', m'', and m'''. m is simply retrieved by computing $m' \oplus m'' \oplus m'''$.

3. **Weakness in RC4**

(a) To prove this I will walk step by step through the key generating function while showing the contents of S, and the values of i and j. First let us assume that $S[1] = x \neq 2$, and for convenience I have merged the last two steps of the function into one.

Step	Command	i	j	$S[1]$	$S[2]$	$S[x]$	Output
0	begin	0	0	x	0	?	
1	$i \leftarrow (i+1) \bmod 256$	1	0	x	0	?	
2	$j \leftarrow (S[i]+j) \bmod 256$	1	x	x	0	?	
3	Swap$(S[i], S[j])$	1	x	?	0	x	
4	Output$(S[i]+S[j])$	1	x	?	0	x	$S[?+x]$
5	$i \leftarrow (i+1) \bmod 256$	2	x	?	0	x	
6	$j \leftarrow (S[i]+j) \bmod 256$	2	x	?	0	x	
7	Swap$(S[i], S[j])$	2	x	?	x	0	
8	Output$(S[i]+S[j])$	2	x	?	x	0	$S[x] = 0$

Therefore, with the given parameters, the second bit of the keystream will always be 0.

(b) First, assume that the key scheduling algorithm produces a uniformly random S. That is, the probability of any $S[i]$ being a certian number is the same as it being any other number. I estimated that this is at least approximately true using a large sample of keys. Similiarily, assume that except for the second byte, all other bytes have equal probability of being 0, and that all other numbers have a uniform distribution. This was also verified using a large sample of keys. Although an estimation, it is a safe assumtion to make given that we are only attempting to **approximate** the probability of the second keystream byte being 0.

From our assumptions, it is fairly simple to calculate the probability of the second byte of the keystream being 0:
p[second byte is 0] $= (p[S[2] = 0] * p[S[1] \neq 2]) + (p[S[2] \neq 0] * p[S[x] = 0])$
Where $x = S[2] + S[S[1] + S[2]]$ (this is from stepping through the key generating algorithm like in (a))

The first part: $p[S[2] = 0] * p[S[1] \neq 2]$ comes from (a). Breaking it up we get $p[S[2] = 0] = \frac{1}{256}$ since the key scheduling algorithm is uniformly random, and $p[S[1] \neq 2] = \frac{255}{256}$ for the same reason.

The second part comes from how 0 would be the output if $S[2]$ was not 0. Breaking it up we get $p[S[2] \neq 0] = \frac{255}{256}$ again because the key scheduling algorithm is uniformly random. $p[S[x] = 0] \approx \frac{1}{256}$ because all other bytes have equal probability of being 0.
Putting that all together, we get p[second byte is 0] $\approx \frac{1}{256} * \frac{255}{256} + \frac{255}{256} * \frac{1}{256} \approx \frac{1}{128}$
This is about twice as probable as any other value, since all other values have equal ($\approx \frac{1}{256}$) chance of being the second keystream byte.

(c) Since we know that the second byte of the keystream is biased towards 0, if we gathered the second bytes of all of the ciphertexts and found the byte that occurred the most often, it would most likely have been encrypted with 0. Given the probabilities from (b), we can expect to see twice as many of these bytes than any other, which is "a good chance" of being right. Since that byte would have been encrypted with 0 (via $\oplus$) that byte is the plaintext byte that we seek.

4. **Feistel ciphers**

(a) Decryption also takes h rounds:
Round 1: $(m_h, m_{h+1}) \rightarrow (m_{h-1}, m_h)$, where $m_{h-1} = m_{h+1} \oplus f_h(m_h)$.
Round 2: $(m_{h-1}, m_h) \rightarrow (m_{h-2}, m_{h-1})$, where $m_{h-2} = m_h \oplus f_{h-1}(m_{h-1})$.
.....
Round h: $(m_1, m_2) \rightarrow (m_0, m_1)$, where $m_0 = m_2 \oplus f_1(m_1)$.

(b) If we have a plaintext/ciphertext pair (m, c), where $m = (m_0, m_1), c = (m_2, m_3)$ it is very easy to break this encryption and find the secret key. Let's look at the result of the encryption process through the feistel ladder:
Round 1: $(m_0, m_1) \rightarrow (m_1, m_2)$, where $m_2 = m_0 \oplus f_1(m_1) = m_0 \oplus m_1 \oplus k_1$
Round 2: $(m_1, m_2) \rightarrow c = (m_2, m_3)$, where $m_3 = m_1 \oplus f_2(m_2) = m_1 \oplus m_2 \oplus k_2 = m_1 \oplus m_0 \oplus m_1 \oplus k_1 \oplus k_2 = m_0 \oplus k_1 \oplus k_2$
So, the values we have are: m_0, m_1, $m_2 = m_0 \oplus m_1 \oplus k_1$, and $m_3 = m_0 \oplus k_1 \oplus k_2$. From these it is trivial to retrieve k. First, $k_1 = m_2 \oplus m_0 \oplus m_1$, and then $k_2 = m_3 \oplus m_0 \oplus k_1$. Putting k_1 and k_2 together we get k.

(c) Similar to (b), we'll walk through the Feistel ladder to see what the result of the encryption is. Again, one round of encryption has the behaviour: $(m_{i-1}, m_i) \rightarrow (m_i, m_{i+1})$, with $m_{i+1} = m_{i-1} \oplus m_i \oplus k_i$.

Round	$\boldsymbol{m_i}$	$\boldsymbol{m_{i+1}}$	
Begin	m_0	m_1	
Round 1	m_1	$m_0 \oplus m_1 \oplus k_1$	
Round 2	$m_0 \oplus m_1 \oplus k_1$	$m_0 \oplus k_1 \oplus k_2$	m_1 cancels out
Round 3	$m_0 \oplus k_1 \oplus k_2$	$m_1 \oplus k_2 \oplus k_3$	m_0 cancels out

So our givens are m_0, m_1, $m_0 \oplus k_1 \oplus k_2$, and $m_1 \oplus k_2 \oplus k_3$. In order to achieve the goal, we only need $k_1 \oplus k_2$ which is easily obtained from $(m_0 \oplus k_1 \oplus k_2) \oplus m_0$, which were both given. Now, assume that k_1 is the bitstring $b_1 b_2 \ldots b_{128}$, which means that k_2 is $b_{128} b_1 b_2 \ldots$. This means that $k_1 \oplus k_2$, which we have, is the bitstring $(b_1 \oplus b_{128})(b_1 \oplus b_2)(b_2 \oplus b_3) \ldots (b_i \oplus b_{i+1}) \ldots (b_{127} \oplus b_{128})$.
If we knew $b_1 = 0$, then we could find b_2, since $b_1 \oplus b_2$ is the second bit of $k_1 \oplus k_2$. Similarily, we could then find b_3 since we have b_2, and the third bit of $k_1 \oplus k_2$. We could repeat this process to retrieve the entirety of k. Therefore, k can be determined to be one of two values based on whether $b_1 = 0$ or $b_1 = 1$.