CO487 Assignment 2

Daniel Burstyn (20206120)

February 6, 2009

1. **Chosen-plaintext attack on Feistel Cipher**

   a) Each 3-bit value is mapped to another 3-bit value, so since there are 8 different 3-bit numbers, the key is 24 bits, thus the keyspace is 2^{24}.

   b) The ciphertext is **556999bdfd74**. The intermediate steps are: $m_0 = 001122$, $m_1 = 334455$, $m_2 = \text{ac8688}$, $m_3 = 556999$, $m_4 = \text{bdfd74}$.

   c) First notice that the plaintext 2b2b2b2b2b2b is **perfect** for this problem. Each of the first nibbles will be 2, and the second nibbles will be b. Passing those through S_0 and S_1 respectively, we get all 2s. Since all digits are 2s, the swapping has no effect, so lastly xoring the 2xs with 2b2b2b gives us $m_2 = 090909$.

   For the next round, we now pass 090909 through the S-boxes, which luckily gives us 111111. Once again, the swapping has no effect. Now we xor with 2b2b2b to get $m_3 = \text{3a3a3a}$.

   For the last round, we pass 3a3a3a through the S-boxes to get 919191. Notice that $z^* = 0$ since each byte is $\text{3a} = 00111010_2$. Now the swapping takes effect. We know that $m_4 = 981010$ from the oracle, so working backwards, we xor m_4 with m_2 (090909) to get 911919 and reverse the permutations from step (vii) to get to 199191. From this we see that the only the nibbles of the first byte were swapped, implying that $S_k(0) = 100_2 = 4$.

   d) The plaintext is **f9841c9002ef**. I started by splitting up the ciphertext into m_3 and m_4, and then started the decryption process for feistel ladder schemes. Now observe that by the scheme $m_2 = f(m_3) \oplus m_4$. Now, since $m_3 = 012345$, then $z^* = 0$, and from (c), we know that $S_k(0) = 4$. Therefore, we are able to compute $f(m_3)$, and find m_2, which is 4b330a.

   Once again, we are lucky enough that $z^* = 0$, and we are able to compute m_1 by the same process, and get that $m_1 = 9002\text{ef}$.

   Unfortunately z^* is now 5, but now, there are only 8 possibilities for $S_k(5)$, so it is trivial to exhaustively search the options using the given oracle until we retrieve the original ciphertext. By this process we find that $S_k(5) = 3$, and that $m_0 = \text{f9841c}$. Thus the plain text (m_0, m_1) is f9841c9002ef.

2. **Meet-in-the-middle attack on Double-DES**

   a) As we learned in class, the meet-in-the-middle attack on Double-DES involves performing an exhaustive search of the first DES, and storing the encrypted values, and then performing an exhaustive search on the second key by decrypting the ciphertext until we find a match with one of our stored encrypted values from the first step. The real key is found once we have eliminated the false matches by testing them with other plaintext/ciphertext pairs.

Unfortunately the attack outlined in class requires ≈ 980,000 Tbytes of storage, whereas only 1000 Tbytes are available for this problem. To fix this, we can use a time-memory tradeoff to save space.

To do this time-memory tradeoff, we can split the k_1 keyspace into 1024 equal subdivisions, with each one assigned $\frac{1}{1024}$ of the keyspace. Now we can make a slight modification to our approach to the meet-in-the-middle attack to take advantage of this split.

Instead of storing the encryptions of the entire k_1 keyspace, we only store the encryptions of one of the subdivisions. Since the entire keyspace would take about 980,000 TBytes, and each subdivision is 1024 times smaller, we only need 980 TBytes of space to store all the key/ciphertext pairs, which meets the given criteria of 1000 TBytes.

For each subdivision, we can exhaustively decrypt the ciphertext to try and find matches. We can see that like the original method, we will compare each encryption to each decryption, and from that we can conclude that we will similarily find the 2^{48} false keys, and of course the correct key, so this method will work but only require 1000 TBytes of memory.

Lastly, let's estimate the number of DES operations that this will take:

- 2^{56} for doing an encryption for all possible k_1 keys.
- $2^{56} * 1024 = 2^{66}$ since we try all possible k_2 keys for each subdivision (of which there are $1024 = 2^{10}$).
- $2^{48} * 2 = 2^{49}$ since it takes an encryption and decryption to try each false key. So the total number of DES operations is $2^{49} + 2^{56} + 2^{66} \approx 2^{66}$.

b) Our network has 1000 workstations that can do 5,000,000 DES operations per second, giving us $5 * 10^9$ DES operations per second total. From (a) we found that we have to do about 2^{66} DES operations total which is about $7.3787 * 10^{19}$ operations. This equates to about 468 years of execution.

To perform an exhaustive key search on Double-DES, we would try every key in the entire keyspace. Since each key for Double-DES is 112 bits long, then there are 2^{112} keys, and two DES operations per key, so a total of 2^{113} DES operations. At 5,000,000,000 DES operations per second, it would still take $33 * 10^{15}$ years to complete - about 33 trillion millenia.

3. **Hash functions derived from block ciphers**

a) To prove that H is not collision resistant, we will find a collision. Consider the plaintext messages with binary values 1 and 10. Since the hash function will pad the end of the plaintext with messages with zeroes, then both of the $\overline{M}$ will be 100....0 - the same value. Thus it is obvious that they will both hash to the same value and cause a collision.

b) Given any hash h we will try to find a two block preimage of h. If the preimage is two blocks long, then observe that $h = E_{M_2}(E_{M_1}(0))$, and thus $E^{-1}_{M_2}(h) = E_{M_1}(0)$. Now we will attempt a meet in the middle attack by choosing 2^{64} keys out of the possible 2^{128} for M_1 and then pick another random 2^{64} keys to use for M_2. Now we can map all the encryptions of 0 (the IV) using the different M_1s and compare all decryptions of h with M_2. So by the birthday paradox, about 50% of the time there will be a match between an encryption and a decryption and we will have found a preimage. Thus H is **not** preimage resistant.

4. **Hash functions** This is very simple to show: let us *assume* that G is **not** collision resistant. This means that there are two inputs x and x‘ that both hash to the same value. Thus we have that $H(H(x)) = H(H(x‘))$.

So there are two possibilites: Either $H(x) = H(x')$ or not. If they are equal then x and x' are a collision in H. If they are not equal, then $H(x)$ and $H(x')$ must be collisions in H. Thus H is not collision resistant. Thus G must be collision resistant by contrapositive.