Lecture Notes 6

1 Data Encryption Standard (DES)

The Data Encryption Standard is a block cipher that operates on 64-bit blocks and uses a 56-bit key. It was the standard algorithm for data encryption for over 20 years until it became widely acknowledged that the key length was too short and it was subject to brute force attack. (The new standard used the Rijndael algorithm and is called AES.)

1.1 Feistel Networks

DES is based on a *Feistel network*. This is a general method for building an invertible function from any function f that scrambles bits. It consists of some number of stages. Each stage i maps a pair of 32-bit words (L_i, R_i) to a new pair (L_{i+1}, R_{i+1}). By applying the stages in sequence, a t-stage network maps (L_0, R_0) to (L_t, R_t). The (L_0, R_0) is the plaintext, and (L_t, R_t) is the corresponding ciphertext.

Each stage works as follows:

$$L_{i+1} = R_i \tag{1}$$

$$R_{i+1} = L_i \oplus f(R_i, K_i) \tag{2}$$

Here, K_i is a *subkey*, which is generally derived in some systematic way from the master key k.

The security of a Feistel-based code lies in the construction of the function f and in the method for producing the subkeys K_i, However, the invertibility follows just from properties of $\oplus$.

The inversion problem is to find (L_i, R_i) given (L_{i+1}, R_{i+1}). Equation 1 gives us R_i. Knowing R_i and K_i, we can compute $f(R_i, K_i)$. We can then solve equation 2 to get

$$L_i = R_{i+1} \oplus f(R_i, K_i)$$

DES uses a 16 stage Feistel network. The pair L_0R_0 is constructed from a 64-bit message by a fixed initial permutation IP. The ciphertext output is obtained by applying IP^{-1} to $R_{16}L_{16}$.

The scrambling function $f(R_i, K_i)$ operates on a 32-bit data block and a 48-bit key block. Thus, a total of $48 \times 16 = 768$ key bits are used. They are all derived in a systematic way from the 56-bit primary key and are far from independent of each other.

1.2 The Scrambling Function

The scrambling function $f(R_i, K_i)$ is the heart of DES. It operates on a 32-bit data block and a 48-bit key block. Thus, a total of $48 \times 16 = 768$ key bits are used. They are all derived in a systematic way from the 56-bit master key k and are far from independent of each other. In a little more detail, k is split into two 28-bit pieces C and D. At each stage, C and D are rotated by one or two bit positions. Subkey K_i is then obtained by applying a fixed permutation (transposition) to CD. (See Table 3.4c of the text.)

The scrambling function itself is rather involved. However, at its heart are eight "S-boxes". These are boxes with 6 binary inputs $c_0, x_1, x_2, x_3, x_4, c_1$ and 4 binary outputs y_1, y_2, y_3, y_4. Each computes some fixed function in $\{0,1\}^6 \rightarrow \{0,1\}^4$. Moreover, each S-box has the very special property that for each of the four possible ways of fixing the values of (c_0, c_1) to Boolean constants, the resulting function on the remaining four inputs $x_1, \ldots, x_4$ is a permutation from $\{0,1\}^4 \rightarrow \{0,1\}^4$. Therefore, we can regard an S-box as performing a substitution on four-bit "characters", where the substitution performed depends both on the structure of the particular S-box and on the values of its "control inputs" c_0 and c_1. The eight S-boxes are all different and are specified by their truth tables.

The S-boxes together have a total of 48 input lines. Each of these lines is the output of a corresponding $\oplus$-gate. One input of each of these $\oplus$-gates is connected to a corresponding bit of the 48-bit subkey K_i. (This is the only place that the key enters into DES.) The other input of each $\oplus$-gate is connected to one of the 32 bits of the first argument of f. Since there are 48 $\oplus$-gates and only 32 bits in the first argument to f, some of those bits get used more than once. The mapping of input bits to $\oplus$-gates is called the *expansion permutation* E and is given by Table 3.2(c) in the text. By looking at the table, one sees that the $\oplus$-gates connected to the six inputs $c_0, x_1, x_2, x_3, x_4, c_1$ for S-box 1 are in turn connected to the input bits $32, 1, 2, 3, 4, 5$, respectively. For S-box 2, they go to bits $4, 5, 6, 7, 8, 9$, etc. Thus, inputs bits $1, 4, 5, 8, 9, \ldots 28, 29, 32$ are each used twice, and the remaining input bits are each used once.

Finally, the 32 bits of output from the S-boxes are passed through a fixed permutation P (transposition) that spreads out the output bits. The outputs of a single S-box at one stage of DES become inputs to several different S-boxes at the next stage. This helps provide the desirable "avalanche" effect described in the text.

1.3 Security considerations

We have mentioned previously that DES is vulnerable to a brute force attack because of its small key size of only 56 bits. However, it has turned out to be remarkably resistant to two recently discovered cryptanalysis attacks, differential cryptanalysis and linear cryptanalysis. The former can break DES using "only" 2^{47} chosen ciphertext pairs. The latter works with 2^{43} chosen plaintext pairs. Neither attack is feasible in practice.

DES has now been replaced as a national standard by the new AES (Advanced Encryption Standard), based on the Rijndael algorithm, developed by two Dutch computer scientists. AES supports key sizes of 128, 192, and 256 bits and works on 128-bit blocks. We will say more about it later in the course.

2 Double Encryption and Group Property

A natural way to attempt to increase the security of a cryptosystem is *double encryption*. Each message is encrypted twice using two different keys k_1 and k_2. That is, $c = E_{k_2}(E_{k_1}(m))$. Now, a brute force attack would require trying all possible keys k_1 and all possible keys k_2, thereby doubling the effective key length. In the case of DES, this would result in a key length of 112 which is plenty large enough to make a full enumeration of the key space infeasible.

Unfortunately, double encryption does not increase the security of a cryptosystem as much as one might naively think. The reason is that it allows new kinds of attacks that are more efficient than brute force.

We consider two cases: When the underlying cryptosystem is a group, and when it is not. DES has been shown not to be a group.

2.1 Group property

Double encryption is really a new cryptosystem created by composing two encryption functions. Let $\hat{k} = (k_1, k_2)$ be the key of the double encryption system, and let $\hat{E}_k$ denote the resulting encryption function, that is,

$$\hat{E}_{\hat{k}}(m) = E_{k_2}(E_{k_1}(m))$$

Combining two functions in this way to define a new function is called *functional composition*. We often write $\hat{E}_{\hat{k}} = E_{k_2} \circ E_{k_1}$ to denote the result of composing E_{k_2} with E_{k_1}.

Let $\mathcal{E} = \{E_k() \mid k \in \mathcal{K}\}$ be the set of possible encryption functions of the original cryptosystem, where $\mathcal{K}$ is the key space. It might happen that $\hat{E}_{\hat{k}}() = E_k()$ for some $k \in \mathcal{K}$. That is, there might be a key k of the original cryptosystem such that encrypting any message m with it gives the same ciphertext as encrypting m first with k_1 and then with k_2. If this is the case for *every* $k_1, k_2 \in \mathcal{K}$, then the original cryptosystem is said to be *closed under functional composition*, and we say that it is a *group*.

When the cryptosystem is a group, double encryption adds no security against a brute force attack. Even though the key length has doubled, the number of distinct encryption functions has not increased (and might actually have decreased). Since every double encryption is same as a single encryption, the double encryption system will fall to a brute force attack on the original cryptosystem.

But it's even worse. If a cryptosystem system is a group, then it is subject to a known plaintext "birthday" attack, which reduces the number of keys that must be tried to roughly the square root of what a brute force attack needs. For example, if the original key length was 56, then only about $\sqrt{2^{56}} = 2^{28}$ keys need be tried.

Briefly, here's how it works: Let (m, c) be a known plaintext-ciphertext pair. Choose 2^{28} random keys k_1 and encrypt m using each. Choose another 2^{28} random keys k_2 and decrypt c using each. Look for a match between these two sets. Assuming reasonable randomness properties of the encryption function, there is a significant probability of finding k_1 and k_2 such that $E_{k_1}(m) = D_{k_2}(c)$. But this implies that $E_{k_2}(E_{k_1}(m)) = c$. In this case, we have succeeded in finding one key pair that works. In all likelihood there are not many pairs that do work, so with significant probability we have cracked the system. Using additional plaintext-ciphertext pairs, we can verify that we have likely found the correct key pair, or repeat this process again if we have not yet succeeded. I've glossed over many assumptions and details, but that's the basic idea.

The drawback to the birthday attack (from the attacker's perspective) is that it requires a lot of storage in order to find a matching element. Nevertheless, if DES were a group, this attack could be carried out in about a gigabyte of storage, easily within the storage capacity of modern workstations.

2.2 What happens when original system is not a group?

Even if the original system is not a group, double encryption still does not result in a cryptosystem with twice the effective key length. The reason is another "birthday"-style known-plaintext attack.

Let's consider the case of double DES. As before, we start with a known plaintext-ciphertext pair (m, c). We carry out a birthday attack by encrypting m under many different keys, decrypting c under many different keys, and hope we find a matching element in the resulting two sets. Unlike the attack described in section 2.1, we encrypt m and decrypt c using all possible DES keys. Thus, we are guaranteed of finding at least one match, since if (k_1, k_2) is the real key pair used in the

double encryption, then $E_{k_1}(m) = D_{k_2}(c)$. If there is only one match, then we have found the key pair and broken the system. If there are several matches, we know that the key pair is one of the matching pairs. This set is likely to be much much smaller than 2^{56}, so they can each be tried on additional plaintext-ciphertext pairs to find which ones work. (Note that there might be more than one key pair that results in the same encryption function. In that case, we won't be able to know which key pair Alice actually used in generating the ciphertexts, but we will be able to find a pair that is just as good and that lets us decrypt her messages.)