Lecture Notes 6

28 Double DES

Even if the original system is not a group (and DES is not), 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 26, 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 correct 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.)

29 Triple DES

Three key triple DES (3TDES) avoids the birthday attack by using three DES encryptions, i.e., $E_{k_3}(E_{k_2}(E_{k_1}(m)))$, giving it an actual key length of 168 bits. While considerably more secure than single DES, for which a brute force attack requires only 2^{56} decryptions, 3TDES can be broken (in principle) by a known plaintext attack using about 2^{90} single DES encryptions and 2^{113} steps. While this attack is still not practical, the effective security thus lies in the range between 90 and 113, which is much smaller than the apparent 168 bits.[1]

A variant of triple DES in which the middle step is a decryption instead of an encryption is known as TDES-EDE, i.e., $E_{k_3}(D_{k_2}(E_{k_1}(m)))$. The variant does not affect the security, but it means that TDES-EDE encryption and decryption functions can be used to perform single DES encryptions and decryptions by taking $k_1 = k_2 = k_3 = k$, where k is the single DES key.

Another variant, two key triple DES (2TDES) uses only two keys k_1 and k_2, taking $k_3 = k_1$. However, known plaintext attacks or chosen plaintext attacks reduce the effective security to only 80 bits. See Wikipedia for further information on triple DES.

[1] The *effective security* measures the amount of work it takes to break a cryptosystem by comparing it with the amount of work required to carry out a brute force attack on a cryptosystem with keys of that length. Thus, a cryptosystem that can be broken in time 2^{90} is said to have 90-bit effective security, even if the actual key length is much greater, since it is no more secure than a system with a 90-bit key.

30 Block Ciphers

A b-bit *block cipher* takes as inputs a key and a b-bit plaintext block and produces a b-bit ciphertext block as output. Most of the ciphers we have been discussing so far are of this type. Block ciphers typically operate on fairly long blocks, e.g., 64-bits for DES, 128-bits for Rijndael (AES). Block ciphers can be designed to resist known-plaintext attacks and can therefore be pretty secure, even if the same key is used to encrypt a succession of blocks, as is often the case.

Of course, the length messages one wants to send are rarely exactly the block length. To use a block cipher to encrypt long messages, one first divides the message into blocks of the right length, padding the last partial block according to a suitable padding rule. Then the block cipher is used in some *chaining mode* to encrypt the sequence of resulting blocks. A chaining mode tells how to encrypt a sequence of plaintext blocks $m_1, m_2, \ldots, m_t$ to produce a corresponding sequence of ciphertext blocks $c_1, c_2, \ldots, c_t$, and conversely, how to recover the m_i's given the c_i's.

Padding involves more than just sticking 0's on the end of a message until its length is a multiple of the block length. The reason is that one must be able to recover the original message from the padded version. If one tacks on a variable number of 0's during padding, how many are to be removed at the end? To solve this problem, a padding rule must include the information about how much padding was added. There are many ways to do this. One way is to pad each message with a string of 0's followed by a fixed-length binary representation of the number of 0's added. For example, if the block length is 64, then at most 63 0's ever need to be added, so a 6-bit length field is sufficient. A message m is then padded to become $m' = m \cdot 0^k \cdot \overline{k}$, where k is a number in the range $[0, 63]$ and $\overline{k}$ is its representation as a 6-bit binary number. k is then chosen so that $|m'| = |m| + k + 6$ is a multiple of b.

Some standard chaining modes are:

- *Electronic Codebook Mode (ECB)* – apply cipher to each plaintext block. That is, $c_i = E_k(m_i)$ for each i. This becomes in effect a monoalphabetic cipher, where the "alphabet" is the set of all possible blocks and the permutation is defined by E_k. To decrypt, Bob computes $m_i = D_k(c_i)$.

- *Cipher Block Chaining Mode (CBC)* – encrypt the XOR of the current plaintext block with the previous ciphertext block to produce the current ciphertext block. That is, $c_i = E_k(m_i \oplus c_{i-1})$. To get started, we take $c_0 = \text{IV}$, where IV is a fixed *initialization vector* which we assume is publicly known. To decrypt, Bob computes $m_i = D_k(c_i) \oplus c_{i-1}$.

- *Cipher-Feedback Mode (CFB)* – XOR the current plaintext block with the encryption of the previous ciphertext block. That is, $c_i = m_i \oplus E_k(c_{i-1})$, where again, c_0 is a fixed initialization vector. To decrypt, Bob computes $m_i = c_i \oplus E_k(c_{i-1})$. Note that Bob is able to decrypt without using the block decryption function D_k. In fact, it is not even necessary for E_k to be a one-to-one function (but using a non one-to-one function might weaken security).

- *Output Feedback Mode (OFB)* – the encryption function is iterated on an *initial vector (IV)* to produce a stream of block keys, which in turn are XORed with the successive plaintext blocks to produce the successive ciphertext blocks. (This is similar to a simple keystream generator.) That is, $c_i = m_i \oplus k_i$, where $k_i = E_k(k_{i-1})$ is a *block key*. k_0 is a fixed initialization vector IV. To decrypt, Bob can apply exactly the same method to the ciphertext to get the plaintext, that is, $m_i = c_i \oplus k_i$, where $k_i = E_k(k_{i-1})$.

- *Propagating Cipher-Block Chaining Mode (PCBC)* – encrypt the XOR of the current plaintext block, previous plaintext block, and previous ciphertext block. That is, $c_i = E_k(m_i \oplus m_{i-1} \oplus$

c_{i-1}). Here, both m_0 and c_0 are fixed initialization vectors. To decrypt, Bob computes $m_i = D_k(c_i) \oplus m_{i-1} \oplus c_{i-1}$.

Remarks

1. Both CFB and OFB are closely related to stream ciphers since in both cases, c_i is m_i XORed with some function of stuff that came before stage i. Like a one-time pad and other simple XOR stream ciphers, OFB becomes insecure if the same key is ever reused, for the sequence of k_i's generated will be the same. CFB, however, avoids this problem, for even if the same key k is used for two different message sequences m_i and m'_i, it will not generally be the case that $m_i \oplus m'_i = c_i \oplus c'_i$; rather, $m_i \oplus m'_i = c_i \oplus c'_i \oplus E_k(c_{i-1}) \oplus E_k(c'_{i-1})$, and the dependency on k does not drop out.

2. The different modes differ in their sensitivity to data corruption. With ECB and OFB, if Bob receives a bad block c_i, then he cannot recover the corresponding m_i, but all good ciphertext blocks can be decrypted. With CBC and CFB, he needs both good c_i and c_{i-1} blocks in order to decrypt m_i. Therefore, a bad block c_i renders both m_i and m_{i+1} unreadable. With PCBC, a bad block c_i renders m_j unreadable for all $j \geq i$.

3. Other modes can be easily invented. We see that in all cases, c_i is computed by some expression (which may depend on i) built from $E_k()$ and $\oplus$ applied to blocks $c_1, \ldots, c_{i-1}$, $m_1, \ldots, m_i$, and the initialization vectors. Any such equation that can be "solved" for m_i (by possibly using $D_k()$ to invert $E_k()$) is a suitable chaining mode in the sense that Alice is able to produce the ciphertext and Bob is able to decrypt it. Of course, the resulting security properties depend heavily on the particular expression chosen.