Chapter 7 Notes

Block-Cipher Modes

1) Electronic Code Book(ECB): This is the typical manner in which block cipher are executed. Simply grab n bits of the plaintext at a time, where n is the block size, and then encrypt them using the algorithm, and output the corresponding ciphertext. Repeat for each subsequent block.

2) Cipher Block Chaining(CBC) mode:

C_0 = IV (initialization vector)
i = 1
while (!done)
  $C_i = e_K(C_{i-1} \oplus M_i)$

M_i and C_i stand for the ith 64 bit block of plain and cipher text respectively.

3) Output Feedback Mode(OFB):

This is a stream cipher, which means that we obtain the ciphertext by XORing the plaintext with a keystream. The keystream is generated with the key and the block cipher system being used.

K_0 = IV
i = 1
while (!done) {
  $K_i = e_K(K_{i-1})$
  $C_i = M_i \oplus K_i$
}

4) Cipher Feedback Mode(CFB):

This is also a stream cipher, except that the keystream is generated by encrypting the ciphertext instead of the plaintext.

C_0 = IV
i = 1
while (!done) {
  $K_i = e_K(C_{i-1})$
  $C_i = M_i \oplus K_i$
}

5) Counter Mode (CTR)

Here, the counter array stores known, prechosen values, so that each block could be encrypted in parallel. It could just store, 1, 2, 3, etc. Or, the initial counter value could be provided and then future counter values could be based on it, but if this were the case, then the counter values would have to be determined sequentially.

```
i = 1
while (!done) {
   Ki = eK(counter[i])
   Ci = Mi ⊕ Ki
}
```

Data Encryption Standard(DES)

Here is the basic algorithm used for DES:

To encrypt a plaintext x of 64 bits and a secret key K of 56 bits do the following:

1) Compute $x_0 = IP(x)$, a fixed permutation of the bits in x. IP is specified in the text.

2) Let $x_i = L_iR_i$, for $0 \leq i \leq 16$, where L_i is the 32 leftmost bits of x_i and R_i is the 32 rightmost bits of x_i. Make the following sequence of computations:

```
for (i=1 to 16) {
   Li = Ri-1
   Ri = Li-1 ⊕ f(Ri-1, Ki)
}
```

Essentially, each loop iteration is known as a Feistel round. (Feistel is the creator of DES.) DEs comprises 16 of these rounds. Each round encrypts ½ of the bits from the pervious round. The function f and the key for the ith round K_i will be discussed in detail later in these notes.

3) $y = IP^{-1}(R_{16}L_{16})$, this means applying the inverse permutation applied in step 1 to the string $R_{16}L_{16}$. (Notice the "reverse" order of the two blocks L_{16} and R_{16}.)

In essence, you would repeat this process for every block of 64 bits that needs to be encrypted.

Now, we need to mention the details of step 2. First the function f:

The first input to f, R_{i-1} is 32 bits, while the second input K_i is 48 bits from the 56 bits of the key K.

1) Expand the 32 bits of R_{i-1} to 48 bits using the matrix E, which is also shown in the book. This matrix delineates an ordering of the bits of R_{i-1} where 16 of the bits are repeated. Let this computed value be $E(R_{i-1})$.

2) Compute $E(R_{i-1}) \oplus K_i$. Let this computation produce $B = B_1B_2...B_8$, where each B_j, $1 \leq j \leq 8$ is 6 bits of B.

3) This is probably the most strange part of the algorithm. In this step the 48 bits of B need to be reduced to 32 bits. This is done via 8 S-boxes, S_1, S_2, ... S_8. One way to think about these S-boxes is the following. Each is a lookup table with 64 rows, 1 for each possible set of 6 binary bits. The right-hand side of the table has entries from 0 to 15, which correspond to 4 binary bits. In essence an S-box specifies a function from 6 binary bits to 4 binary bits. Compute $C_j = S_j(B_j)$ for , $1 \leq j \leq 8$. Let $C = C_1C_2...C_8$.

4) $f(R_{i-1}, K_i) = P(C)$, where P is a fixed permutation of the bits in C. (P is included in the text.)

How to use the S-boxes in the text

Let the 6 input bits be $b_1b_2b_3b_4b_5b_6$. Let $R = b_1b_6$, a binary value that ranges from 0 to 3, and $C = b_2b_3b_4b_5$, a binary value ranging from 0 to 15. R will tell you the row to look on in the S-box. (Top row is 0, bottom is 3.) S will tell you the column to look on in the S-box. Each value in an S-box is from 0 to 15. This corresponds to 4 binary bits, the output.

How to determine the Key schedule $K_1...K_{16}$ from the key K

The total key including parity bits is 64 bits. The parity bits are bits 8, 16, 24, ... 64. The other 56 bits are the key K. Here is how you compute each K_i :

1) Compute $PC\text{-}1(K) = C_0D_0$, where C_0 is the leftmost 28 bits of PC-1(K), and D_0 is the rightmost 28 bits of PC-1(K). PC-1 is a fixed permutation, also stated in the text.

2) Here is the computation of the key schedule:

for i=1 to 16 {
  $C_i = LS_i(C_{i-1})$
  $D_i = LS_i(D_{i-1})$
  $K_i = PC\text{-}2(C_iD_i)$
}

PC-2 is another fixed permutation. LS_i is a left-shift of either 1 bit or 2 bits. If i=1,2,9, or 16, then LS_i is a left-shift of 1 bit. Otherwise it is a two bit left shift.

Characteristics of the S-boxes, as pointed out by the NSA

1) Each row is a permutation of the values 0, 1, ..., 15.
2) No S-box is a linear or affine function of its inputs.
3) Changing one input bit to an S-box causes at least 2 output bit changes.
4) For all x, S(x) and S(x $\oplus$ 001100) differ in at least 2 digits.
5) S(x) $\neq$ S(x $\oplus$ 11ef00), for all binary bits e and f.
6) If you fix a single input bit and observe a particular output bit, that output bit is relatively random. (The 32 possible inputs (when fixing a bit) lead to at worst a 13-19 split of 0s and 1s in any particular output bit.)

Cryptanalysis of DES: Differential Cryptanalysis

This is a known-plaintext attack that requires many pairs of plaintext-ciphertext blocks. The key observation behind this technique is as follows:

Imagine two pairs matching pairs (P1, C1) and (P2, C2). Imagine if the ciphertexts were created by XORing the plaintexts with a key, K, then we have:

$$C1 = P1 \oplus K, \text{ and } \quad C2 = P2 \oplus K$$

Now compute

$$C1 \oplus C2 = (P1 \oplus K) \oplus (P2 \oplus K) = P1 \oplus P2$$

Thus, the XOR of the two plaintexts is the SAME as the XOR of the two ciphertexts.

This will not be the case with DES, in a regular mode, in fact, there will be a difference in the XOR of the ciphertexts as compared to the XOR of the plaintexts. The key behind differential cryptanalysis is examing whether or not there is a PATTERN between the differences in the XORs of pairs of plaintexts and pairs of ciphertexts.

As mentioned previously, the only part of the DES algorithm that isn't linear is the S-boxes, thus, we will confine your cryptanalysis to these S-boxes. Here's a description of the technique:

Let two inputs to S_j be B_j and B_j^*. Then, we will refer to $B_j \oplus B_j^*$ as the input x-or. Similarly, we will refer to $S_j(B_j) \oplus S_j(B_j^*)$ as the output x-or. Also, let $X' = X \oplus X^*$, for any bitstring X and X^*.

Next, define delta(B_j') to consist of the ordered pairs (B_j, B_j^*) having the input x-or of B_j'. There are 64 elements in this set.

For each of the 64 values in this set, we can calculate the corresponding output x-or. It is possible that these output x-ors are not evenly distributed. This will be the "weakness" that will be exploited in this technique.

Let's take a look at S-box #1, and consider the set delta(110100) = { (000000, 110100), (000001, 110101), ... , (111111, 001011)}.

Now, for each ordered pair, such as the first one, calculate $S_1(000000) = 1110$, and $S_1(110100) = 1001$, then take the xor of these, yielding 0111, now we tally that we have seen the output x-or 0111 (or 7), once. Doing this tally for all 64 ordered pairs yields the following output frequencies:

out x-or	0000	0001	0010	0011	0100	0101	0110	0111
freq	0	8	16	6	2	0	0	12
out x-or	1000	1001	1010	1011	1100	1101	1110	1111
freq	6	0	0	0	0	8	0	6

In this example, only 8 of the 16 possible outputs occur. In general, if we fix an S box and an input x-or, 75%-80% of the outputs typically occur.

Now, define $IN_j(B_j', C_j')$ to be the set of input values to S-box j that create an output-xor of C_j', when our input x-or is B_j'. For example, for $B_j' = 110100$, and $C_j' = 0011$, we have $IN_1(110100, 0011) = \{ 000001, 000010, 010101, 100001, 110101, 110110\}$ because $S_1(000001) \oplus S_1(110101) = 0011$. (To see this, note that $S_1(000001) = 0000$ and $S_1(110101) = 0011$.) Similarly $S_1(000010) \oplus S_1(110110) = 0011$, and $S_1(010101) \oplus S_1(100001) = 0011$.

Advanced Encryption Standard (AES)

In 1999, NIST decided that DES was no longer secure due to the increasing speed of computers. They solicited algorithms to replace DES to be the new government standard, AES. There were five finalists chosen, and of those, the winning entry was submitted by Joan Daemon and Vincent Rijment and is called Rijndael (pronouced "rain-doll".)

Their cipher, along with the other finalists, adhered to the following criteria laid out by NIST:

1) Symmetric block cipher with three possible key lengths: 128 bits, 192 bits and 256 bits.

2) More secure than Triple-DES

3) Must be part of the public domain, royalty free

4) It should remain secure for at least 30 years.

The algorithm utilizes mathematics with polynomials within the field $GF(2^8)$ modulo the irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$. The details of this are beyond the scope of this class, but a method to carry out all the steps necessary to run AES will be given without the mathematical explanation of what is really happening.

For 128-bit AES, the algorithm runs 10 rounds. For 192-bit AES, it runs 12 rounds. For 256-bit AES, it runs 14 rounds. Only the 128-bit AES will be discussed here in detail.

Assume that we have a block of 128 bits, split into 16 bytes, labeled $a_{0,0}$, $a_{1,0}$, $a_{2,0}$, $a_{3,0}$, $a_{0,1}$, $a_{1,1}$, $a_{2,1}$, $a_{3,1}$, $a_{0,2}$, $a_{1,2}$, $a_{2,2}$, $a_{3,2}$, $a_{0,3}$, $a_{1,3}$, $a_{2,3}$, and $a_{3,3}$. You can visualize these 16 bytes filling up four columns of four bytes, with the first four elements in the first column, the second four elements in the second column, etc.

The following is repeated for 10 rounds:

1) Substitute bytes

For each of the sixteen bytes, look up their substitute in the s-box substitution chart, creating the new state matrix $b_{0,0}$, $b_{1,0}$, $b_{2,0}$, $b_{3,0}$, $b_{0,1}$, $b_{1,1}$, $b_{2,1}$, $b_{3,1}$, $b_{0,2}$, $b_{1,2}$, $b_{2,2}$, $b_{3,2}$, $b_{0,3}$, $b_{1,3}$, $b_{2,3}$, and $b_{3,3}$. This s-box is actually created using mathematical inverses mod m(x) in the field GF(2^8). But, we will skip these details here. For our purposes, we can simply look up each substitution value.

2) Shift rows

In row i, perform a cyclic left shift of i bytes. (Note: The rows are numbered 0 through 3.) This will result in the following matrix:

$b_{0,0}$	$b_{0,1}$	$b_{0,2}$	$b_{0,3}$
$b_{1,1}$	$b_{1,2}$	$b_{1,3}$	$b_{1,0}$
$b_{2,2}$	$b_{2,3}$	$b_{2,0}$	$b_{2,1}$
$b_{3,3}$	$b_{3,0}$	$b_{3,1}$	$b_{3,2}$

3) Mix columns

"Multiply" the state matrix with the following matrix:

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}$$

Now this isn't regular multiplication, it's multiplication in the field mentioned before. Also, instead of adding the four terms in each operation, the four terms get XORed together. Multiplying by 01 is just the identity, and multiplying by 03 is the same as multiplying by 01 and multiplying by 02 and XORing the two results. Thus, the only thing that has to be described is multiplying by 02:

To multiply a byte by 02 (for AES, not in general), do the following:

1) Left-shift by 1 bit.
2) If the left-most bit of the original value was a 1, XOR the result from step 1 with 00011011.

Here are three examples:

4E x 02 = 01001110 x 00000010 = 10011100 = 9C (just perform the left shift)

A7 x 02 = 10100111 x 00000010 = 01001110 XOR 00011011 = 01010101 = 55

C9 x 03 = 11001001 x 01 XOR 11001001 x 02 =

= 11001001 XOR 110010010
= 11001001 XOR 10010010
XOR 00011011

= 11001001 XOR 10001001

= 11001001 XOR
10001001
= 01000000 (40)

(Note: The typical convention is to write bytes as two hex characters.)

4) Add Round Key

In this last phase of each round, the state matrix is simply XORed with the key for that particular round.

Now, we must discuss how to generate the round keys from the original 128-bit key. (Each round key is also 128 bits.)

Here is pseudocode which shows how to produce the round keys (taken from Stallings pg. 160):

```
KeyExpansion(byte key[16], word w[44]) {

  word temp;

  for (i=0; i<4; i++)
    w[i] = (key[4i], key[4i+1], key[4i+2], key[4i+3]);

  for (i=4; i<44; i++) {
    temp = w[i-1];
```

```
    if (i%4 == 0)
       temp = SubWord(RotWord(temp)) XOR Rcon[i/4];
    w[i] = w[i-4] XOR temp;
  }
}
```

Normally, we simply XOR two previous words (32 bits – the last four, and the fourth to last word) to get the new word. But, for each multiple of 4, we do a special operation on temp. Namely, we first perform a cyclic left-shift of one word to it (this is the RotWord), then we perform a byte substitution on each byte in it based on the original S-box, also used in the beginning of the algorithm, and finally we XOR it with a value stored in the array Rcon. Here are the values:

j	1	2	3	4	5	6	7	8	9	10
RCon[j]	01	02	04	08	10	20	40	80	1B	36

This array starts with the value 01 in the first index, and all subsequent indexes store a value obtained by doubling the previous value in the field discussed earlier. In all cases except for going from index 8 to 9, this is just regular doubling. Here's how we calculate index 9:

80 x 02 = 10000000 x 00000010 = 00000000 XOR 00011011 = 00011011 = 1B.

(This was covered a bit earlier in these notes – we are just multiplying by 2 in the field.)