Lecture 9: Using Block and Stream Ciphers in Real-World Systems for Secure Communications

Lecture Notes on "Computer and Network Security"

by Avi Kak (kak@purdue.edu)

February 17, 2009

©2009 Avinash Kak, Purdue University

Goals:

- To present 2DES and its vulnerability to the meet-in-the-middle attack
- To present two-key 3DES and its potential vulnerabilities
- To present three-key 3DES
- To present the five different modes in which a block cipher can be used in practical systems for secure communications
- To discuss stream ciphers and to review RC4 stream cipher algorithm
- To review problems with the WEP protocol for home wireless networks

9.1: Multiple Encryption with DES for a More Secure Cipher

- As you already know, the DES cryptographic system was shown to not be very secure about 10 years ago.

- We can obviously use AES cryptography that is designed to be extremely secure, but the world of commerce and finance does not want to give up on DES that quickly (because of all the investment that has already been in DES-related software and hardware).

- So that raises questions like: How about a cryptographic system that carries out repeated encryption with DES? Would that be more secure?

- We will now show that whereas double DES may not be that much more secure than regular DES, we can expect triple DES to be very secure.

9.2: Double DES

- The simplest form of **multiple encryption** with DES is **double DES** that has two DES-based encryption stages using two different keys.

- Let's say that P represents a 64-byte block of plaintext. Let E represent the process of encryption that transforms a plaintext block into a ciphertext block. Let's use two 56-byte encryption keys K_1 and K_2 for a double application of DES to the plaintext. Let C represent the resulting block of ciphertext. We have

$$C = E(K_2,\ E(K_1,\ P))$$

$$P = D(K_1,\ D(K_2,\ C))$$

  where D represents the process of decryption.

- With two keys, each of length 56 bits, double DES in effect uses a 112 bit key. One would think that this would result in a dramatic increase in the cryptographic strength of the cipher — *at least against the brute-force attacks to which the regular DES is so vulnerable.* Recall that in a brute force attack, you try every possible key to break the code.

9.3: Can a Double-DES Plaintext-Ciphertext Mapping be Equivalent to a Single-DES Mapping?

- Suppose it should happen that the plaintext to ciphertext mapping achieved with two encryption keys is equivalent to a mapping that one would achieve with some third key:

$$E(K_2,\ E(K_1,\ P)) \quad = \quad E(K_3,\ P)$$

  then the whole point of a double DES would be lost. (One can extend this argument to state that in that case any number of stages of multiple encryption would amount to a single stage of regular DES. Therefore, a cipher consisting of an arbitrary number of applications of DES encryption would be no stronger than regular DES.)

- Fortunately, the probability that plaintext to ciphertext mapping achieved with double DES is equivalent to the mapping with regular DES is extremely remote. The arguments that justify this are presented below.

- Note that every key creates a mapping between the words corresponding to an input block and the words corresponding to an output block.

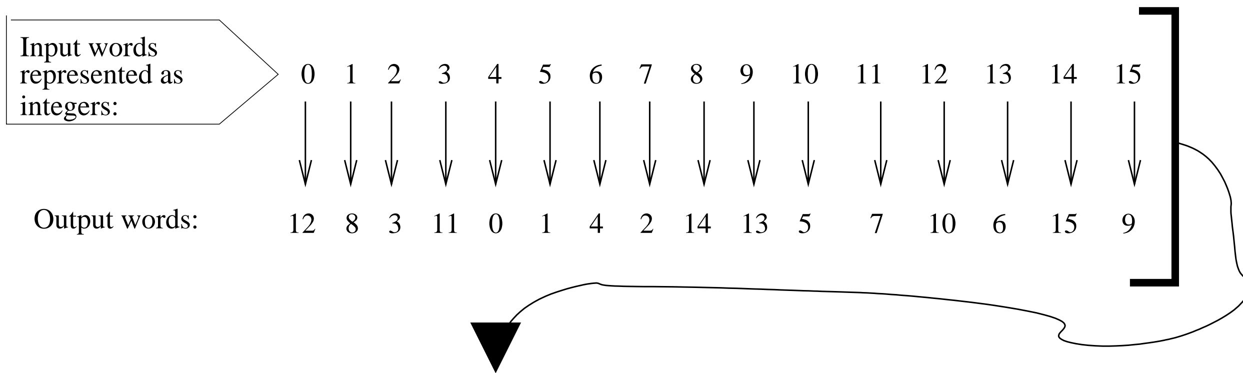
- Consider 4-bit blocks. Every key gives us a unique mapping between the 16 possible words at the input and the 16 possible words at the output.

- Every mapping between the input words and the output words must amount to a **permutation** of the input words. This is necessitated by the fact that any mapping between the plaintext words and the ciphertext words must be 1-1, since otherwise decryption would not be possible.

- To understand what I mean by a **mapping** between the input words and the output words being a **permutation**, let's continue with our block size of 4 bits. Figure 1 shows one possible mapping between the 16 different input words that you can have with 4 bits and the output words. The 16 output words constitute one permutation of the 16 input words. The total number of permutations of 16 input words is 16!. [Note that a permutation means that you are looking at N *different* objects in a sequence. A different permutation would correspond to the N objects appearing in a different order in the sequence. There are $N!$ ways of arranging N different objects in a sequence. Consider the case when $N = 3$ and the objects are a, b, and c. The six different ways of arranging these objects in a sequence is abc, acb, bac, bca, cab, and cba.]

- So with a block size of 4 bits, we have a maximum of 16! mappings between the input words and the output words. In other words,

Blocksize = 4 bits



This is one possible mapping between the 16 input words and the 16 output words

The 16 output words constitute one permutation of the 16 input words.

Since there are 16! permutations of the 16 input words, there exist 16! different possible mappings between the input and the output.

The input–output mapping obtained with one encryption key is only one of 16! different possible mappings.

Figure 1: *This figure is from Lecture 9 of "Computer and Network Security" by Avi Kak*

we have $2^4!$ mappings when block size is 4 bits. When we select a key for encryption, we use one of these $2^4!$ mappings.

- Let's now extend the above argument to the case when the block size is 64 bits.

- How many different possible mappings do we have for 64-bit blocks?

- As before, each encryption key gives us one mapping between the input 64-bit words and the output 64-bit words. Since there are 2^{64} possible words, each mapping is a relationship between the 2^{64} different possible words at the input and equal number of such words at the output.

- Since each mapping can be thought of as a permutation of the 2^{64} possible words at the input, we have a maximum of $2^{64}!$ possible mappings between the input words and the output words.

$$\begin{aligned} \left(2^{64}\right)! &= 10^{347380000000000000000} \\ &> \left(10^{10^{20}}\right) \end{aligned}$$

- Now with a key size of 56 bits, we have a total of 2^{56} different keys. Each key corresponds to one of the $2^{64}!$ different possible mappings. The number 2^{56} is upperbounded by 10^{17}.

- We conclude that the total number of mappings that can be created with single DES encryption is an extremely tiny number compared to the total number of all possible possible mappings from plaintext to ciphertext.

- You may think of the all possible $2^{64}!$ mappings as the stars in the sky. Each star is one such mapping. An extremely tiny number of these stars randomly scattered in the sky correspond to the mappings for a single DES encryption.

- When we carry out a double DES encryption, we obtain a mapping that is again a star in the sky. Since a permutation of a permutation is still a permutation, obviously the mapping corresponding to the double DES encryption must be one of the different possible $2^{64}!$ mappings. But since all possible mappings corresponding to single DES encryption is such a small random sampling of all possible mappings, the chances of double DES encryption corresponding to some single DES encryption are extremely remote.

- Another way of saying the same thing would be that the input-output mappings for single DES encryption constitute an extremely sparse (and random) sampling of the set of all possible mappings (which is of size $2^{64}!$), the probability of the mapping corresponding to double DES encryption belong to the sparse set are negligible.

9.4: Vulnerability of Double DES to the Meet-in-the-Middle Attack

- Any double block cipher, that is a cipher that carries out double encryption of the plaintext using two different keys in order to increase the cryptographic strength of the cipher, is open to what is known as the **meet-in-the-middle attack**.

- To explain the meet-in-the-middle attack, let's revisit the relationship between the plaintext P and the ciphertext C for double DES:

$$C = E(K_2,\ E(K_1,\ P))$$

$$P = D(K_1,\ D(K_2,\ C))$$

  where K_1 and K_2 are the two 56-bit keys used in the two stages of encryption.

- Let's say that an attacker has available to him/her a plaintext-ciphertext pair (P, C). From the perspective of the attacker, there exists an X such that

$$X \quad = \quad E(K_1,\ P) \quad = \quad D(K_2, C)$$

- Now let's say that the attacker creates a sorted table of all possible value for X for a given P by trying all possible 2^{56} keys. This table will have 2^{56} entries. We will refer to this table as T_E.

- The attacker can also create another table of all possible X' by decrypting C using every one of the 2^{56} keys. This table also has 2^{56} entries. Let's call this table T_D.

- Now the question is: How many of the X entries in T_E are likely to be the same as the X' entries in T_D? Ideally, only one X entry in T_E will be the same as a single X' entry in T_D. But, as we will see, in general the number of matches will be very large. So we will refer to this count as the number of **false alarms**.

- Any given input 64-bit word P can be mapped to any one of the output 2^{64} words. With an effective key size of 112 bits, we have a total of 2^{112} keys and each of those keys will result in an output ciphertext word for a given input plaintext word. Obviously, the total of 2^{112} keys will only be able to produce a maximum of 2^{64}

distinct output words. What that means is that we can divide the 2^{112}-sized keyspace into subspaces, each of size,

$$\frac{2^{112}}{2^{64}} = 2^{48}$$

and, on the average, each such subspace will result in a unique ciphertext block C for a given 64-bit plaintext block P.

- Therefore, on the average, 2^{48} of the different possible 2^{112} key pairs (K_1, K_2) will transform a given 64-bit plaintext block into the same block of ciphertext.

- Therefore, when we compare the 2^{56} entries of X in T_E with the 2^{56} entries of X' in T_D, on the average we are likely to run into 2^{48} false alarms. That is, we are likely to run into 2^{48} cases of (K_1, K_2) pairs for which the X that corresponds to K_1 will be the same as X' that corresponds to K_2.

- Now suppose we have another $(P',\ C')$ pair of 64-bit words available to us. This time, we will only try the 2^{48} key pairs (K_1, K_2) on which we obtained equalities when comparing the X entries in T_E with the X' entries in T_D. Let the new tables be called T'_E and T'_D.

- Now we have no redundancy at all with regard to the C values produced by the different keys for a given P. On the other hand, now we have "negative redundancy". Now, on the average, $2^{48}/2^{64} = 2^{-16}$ key pairs (K_1, K_2) will produce the same output C for a given input P.

- Therefore, the matching entry in comparing T'_E with T'_D is practically guaranteed to yield the encryption keys K_1 and K_2.

- The effort required to make such a comparison is proportional to the size of the tables T_E and T_D, which is 2^{56}, which is comparable to the effort required to break the regular DES.

9.5: Triple DES with Two Keys

- An obvious defense against the **meet-in-the-middle** attack is to use triple DES.

- The most straightforward way to use triple DES is to employ three stages of encryption, each with its own key:

$$C \quad = \quad E(K_3,\ E(K_2,\ E(K_1,\ P)))$$

  But this calls for 168-bit keys, which is considered to be unwieldy for many applications.

- One way to use triple DES is with just two keys as follows

$$C \quad = \quad E(K_1,\ D(K_2,\ E(K_1,\ P)))$$

  Note that one stage of **encryption** is followed by one stage of **decryption**, followed by another stage of **encryption**. *This is also referred to as EDE encryption, where EDE stands for encryption-decryption-encryption.*

- There is an important reason for juxtaposing a stage of decryption between two stages of encryption: it makes the triple DES system

easily usable by those who are only equipped to use regular DES. This backward compatibility with regular DES can be achieved by setting $K_1 = K_2$ in triple DES.

- It is important to realize that juxtaposing a decryption stage between two encryption stages does not weaken the resulting cryptographic system in any way. Recall, decryption in DES works in exactly the same manner as encryption. So if you encrypt data with one key and try to decrypt with a different key, the final output will be still be an encrypted version of the original input. The nature of this encrypted output will not be different, from the standpoint of cryptographic strength, from the case if you use two stages of encryption.

- **Triple DES with two keys is a popular alternative to regular DES.**

9.6: Possible Ways to Attack 3DES Based on Two Keys

- It is theoretically possible to extend the **meet-in-the-middle attack** to the case of 3DES based on two keys.

- Let's go back to the encryption equation for two-key 3DES:

$$C \quad = \quad E(K_1,\ D(K_2,\ E(K_1,\ P)))$$

  We can rewrite this equation in the following form

$$\begin{aligned} A &= E(K_1,\ P) \\ B &= D(K_2,\ A) \\ C &= E(K_1,\ B) \end{aligned}$$

- If the attacker had some way of knowing the intermediate value A for a given plaintext P, breaking the 3DES cipher becomes the same as breaking 2DES with the meet-in-the-middle attack.

- In the absence of knowledge of A, the attacker can assume some arbitrary value for A and can then try to find a known $(P,\ C)$ that results in that A by using the following procedure:

Step 1: The attacker procures n pairs of $(P,\ C)$. These are arranged in a two-column table, with all the P's in one column and their corresponding C's in the other column. This table has n rows. We refer to this table as **Table I**.

Step 2: The attacker now chooses an arbitrary A. Using this A, the attacker figures out the plaintext that will result in that A for every possible key K_1:

$$P \quad = \quad D(K_1,\ A)$$

(Recall that, in encryption, A is related to P by $A\ =\ E(K_1, P)$.) If a P calculated in this manner is found to match one of the rows in Table I, for the key K_1 that yielded this match we now find B from

$$B \quad = \quad D(K_1,\ C)$$

for the C value that corresponds to the P value in Table I. This B value and its corresponding key K_1 is entered as a row in **Table II**. (Recall that, in encryption, C is related to B by $C\ =\ E(K_1,\ B)$.)

Step 3: Given all the available $(P,\ C)$ pairs, we now fill Table II with $(B,\ K_1)$ pairs where the set of K_1's **constitutes our candidate pool for the K_1 key**.

Step 4: We now sort Table II on the B values.

Step 5: In Table II constructed as above, the left column entries, meaning B's, were obtained from the available samples of ciphertext C. Recall, the other way to obtain B is by

$$B \quad = \quad D(K_2,\ A)$$

We now try, one at a time, all possible values for the K_2 key in this equation for the assumed value for A. (Obviously, there are 2^{56} possible values for K_2.) When we get a B that is in one of the rows of Table II, we have found a candidate pair $(K_1,\ K_2)$.

Step 6: The candidate pair of keys $(K_1,\ K_2)$ is tested on the remaining $(P,\ C)$ pairs. If the test fails, we try a different value for A in Step 2 and the process is repeated.

- Let's now talk about the effort involved in arriving at a correct guess for the $(K_1,\ K_2)$ pair of keys.

- For a given pair (P, C), the probability of guessing the correct intermediate A is $1/2^{64}$.

- Therefore, given the n pairs of $(P,\ C)$ values in Table I, the probability that **a particular chosen value for** A will be correct is $n/2^{64}$.

- Now we will use the following result in probability theory: the **expected number of draws required** to draw one red ball from a bin containing n red balls and $N - n$ green balls is $(N + 1)/(n + 1)$ **if the balls are NOT replaced**.

- Therefore, given the n pairs for $(P,\ C)$, the number of different possible values for A that we may have to try is given by

$$\frac{2^{64}\ +\ 1}{n\ +\ 1} \quad \approx \quad \frac{2^{64}}{n}$$

  which is roughly in agreement with the probability $n/2^{64}$ of choosing the correct value for A if we are given n pairs for $(P,\ C)$.

- Because the size of the effort involved in Step 5 is of the order of 2^{56}, the above expression implies that the running time of the attack would be of the order of

$$2^{56} \cdot \frac{2^{64}}{n} = 2^{120-\log n}$$

9.7: Triple DES with Three Keys

- Here is a 3-key version of a more secure cipher that is based on multiple encryption with DES:

$$C \quad = \quad E(K_3,\ D(K_2,\ E(K_1,\ P)))$$

  where the decryption step in the middle is purely for the sake of backward compatibility with the regular DES, with 2DES, and with 3DES using two keys.

- When all three keys are the same, that is when $K_1 = K_2 = K_3$, 3DES with three keys become identical to regular DES.

- When $K_1 = K_3$, we have 3DES with two keys.

- Note that as with 3DES using two keys, the decryption stage in the middle does NOT reduce the cryptographic strength of 3DES with three keys. Especially since the encryption and decryption algorithms are the same in DES, decrypting with a key that is different from the key used in encryption does not bring the output any closer to the input.

- A number of internet-based applications have adopted 3DES with three keys. **These include PGP and S/MIME**.

9.8: Five Modes of Operation for Block Ciphers

There are five different modes in which a block cipher, such as DES or AES, can be used:

Electronic Code Book (ECB): Each block of plaintext is coded independently. Not very secure for long segments of plaintext, especially plaintext containing repetitive information (**even more especially if the nature of what is repetitive in the plaintext is known to the attacker**). Used primarily for secure transmission of short pieces of information, such as an encryption key.

This method is referred to as the Electronic Code Book method because the encryption process can be represented by a *fixed mapping* between the input blocks of plaintext and the output blocks of cipher text. So it is very similar to the code book approach of the distant past. The code book would list the ciphertext mapping for each plaintext word.

Another shortcoming of ECB is that the length of the plaintext message must be integral multiple of the block size. When that condition is not met, the plaintext message must be padded appropriately.

We will not talk any further about ECB here. The rest of the

modes discussed below provide enhanced security by making the ciphertext for any block a function of all the blocks seen previously. Some of these methods also do **not** require that the size of the plaintext be an integral multiple of the block size.

Cipher Block Chaining Mode (CBC): The input to the encryption algorithm is the XOR of the next block of plaintext and the previous block of ciphertext. This is obviously more secure for long segments of plaintext. This mode also requires that length of the plaintext message be an integral multiple of the block size. When that condition is not satisfied, the message must be suitably padded.

Cipher Feedback Mode (CFB): Whereas the CBC mode uses all of the previous ciphertext block to compute the next ciphertext block, the CFB mode uses only a fraction thereof. Also, whereas in the CBC mode the encryption system digests b bits of plaintext at a time (where b is the blocksize used by the block cipher), now the encryption system digests only $s < b$ number of plaintext bits at a time even though the encryption algorithm itself carries out a b-bits to b-bits transformation. Since s can be any number, including one byte, that makes CFB suitable as a stream cipher.

Output Feedback Mode (OFB): The basic logic here is the

same as in CFB, only the nature of what gets fed from stage to stage is different. In CFB, you feed $s < b$ number of ciphertext bits from the current stage into the b-bits to b-bits transformation carried out by the next-stage encryption. But in OFB, you feed s bits from the output of the transformation itself. This mode of operation is also suitable if you want to use a block cipher as a stream cipher.

Counter Mode (CTR): Whereas the previous four modes for using a block cipher are intuitively plausible, this new mode at first seems strange and seemingly not secure. But it has been theoretically established that this mode is at least as secure as the other modes. As for CFB and OFB, an interesting property of this mode is that only the encryption algorithm is used at both the encryption end and at the decryption end. The basic idea consists of applying the encryption algorithm **not** to the plaintext directly, but to a b-bit number (and its increments modulo 2^b for successive blocks) that is chosen beforehand. The ciphertext consists of what is obtained by XORing the encryption of the number with a b-bit block of plaintext.

We will next examine the CBC, CFB, OFB, and the CTR modes in greater detail.

9.9: Block-Cipher Modes of Operation: The Cipher Block Chaining Mode (CBC)

- To overcome the security deficiency of the ECB mode, the input to the encryption algorithm consists of the XOR of the plaintext block and the ciphertext produced from the previous plaintext block. See Figure 2.

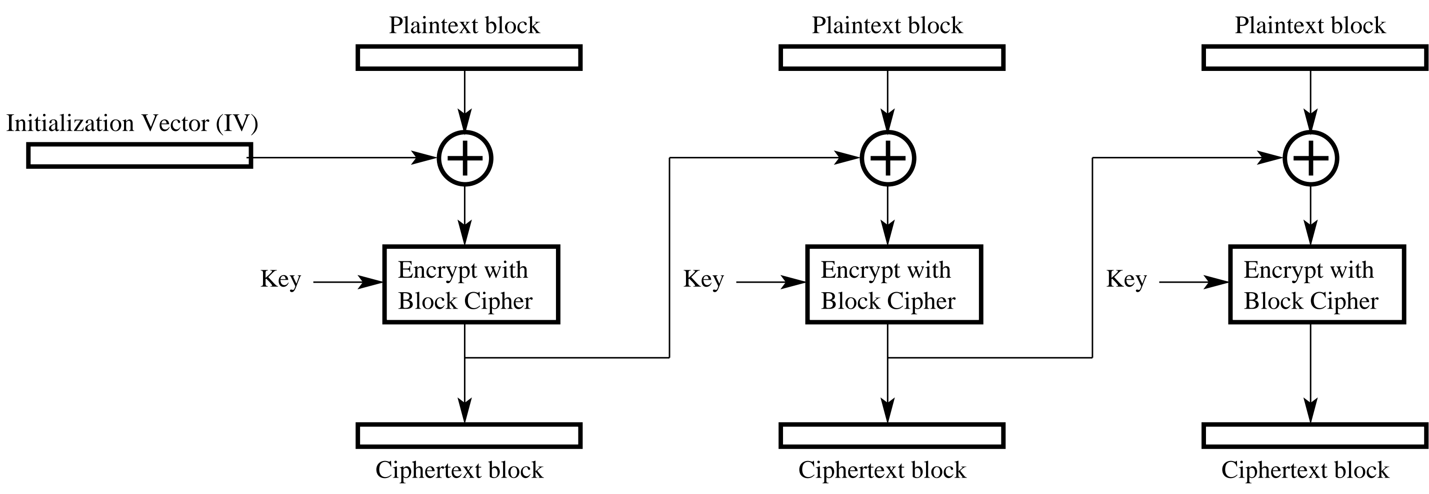
- This makes it more difficult for a cryptanalyst to break the code using strategies that look for patterns in the ciphertext, patterns that may correspond to the known structure of the plaintext.

- To get started, the chaining scheme shown in Figure 2 obviously needs what is known as the **initialization vector** for the first invocation of the encryption algorithm.

- The initialization vector, denoted IV, is sent separately as a short message using the ECB mode.

- With this chaining scheme, the ciphertext block for any given plaintext block becomes a function of all the previous ciphertext

blocks.



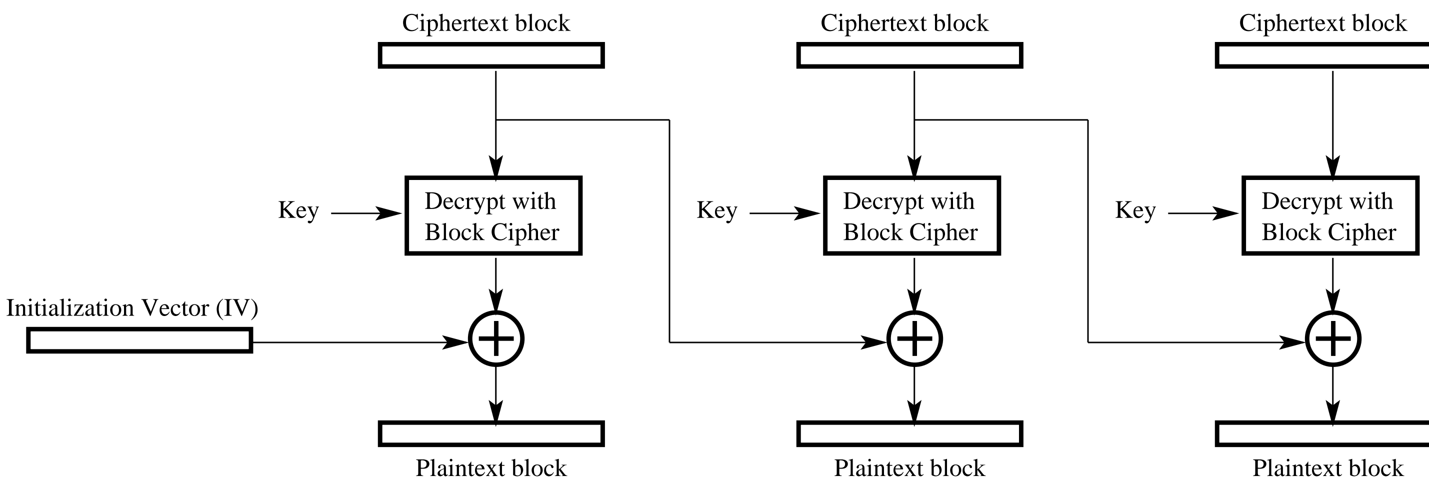


Figure 2: *This figure is from Lecture 9 of "Computer and Network Security" by Avi Kak*

9.10: Block-Cipher Modes of Operation: The Cipher Feedback Mode (CFB)

- This approach. illustrated in Figure 3, allows a block cipher to be used as a **stream cipher**.

- With a block cipher, if the length of the message is not an integral number of blocks, you must pad the message. It is not necessary to do so with a stream cipher.

- Character oriented stream ciphers can be used for real-time encryption of continuous data on a byte-by-byte basis.

- Let's say you want to use a block cipher as a stream cipher that encrypts one byte at a time. Its basic steps are
  - Start with an **initialization vector**, IV, of the same size as the blocksize expected by the block cipher.
  - Encrypt the IV with the block cipher encryption algorithm.

- Retain only one byte from the output of the encryption algorithm. Let this be the most significant byte. Discard the rest of the output.

- XOR the byte retained with the byte of the plaintext that needs to be transmitted. Transmit the output byte produced.

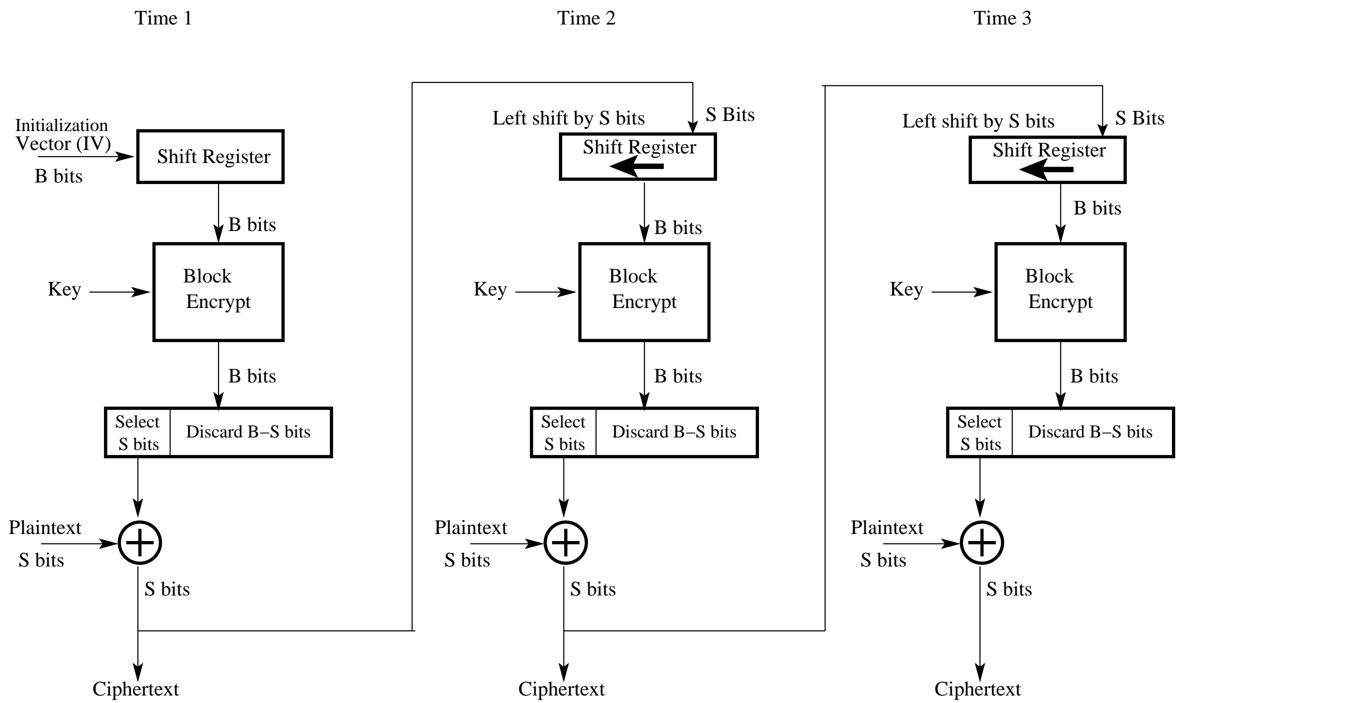
- Shift the IV one byte to the left (discarding the leftmost byte) and insert the ciphertext byte produced by the previous step as the rightmost byte. So the new IV is still of the same length as the block size expected by the encryption algorithm.

- Go back to the step "Encrypt the IV with the block cipher encryption algorithm".

- Figure 3 shows these steps on a recurring basis for both encryption and decryption. The figure is slightly more general than the description above because it assumes that you want the unit of transmission to be s bits, as opposed to 1 byte. But it is typically the case that $s = 8$.

- A most important thing to note about the scheme in Figure 3 is that only the encryption algorithm is used in both encryption and decryption. (This can be an important implementation-level detail for those block ciphers for which the encryption and the decryption algorithms are significantly different. AES is a case in point.)

- Note that the ciphertext byte produced for any plaintext byte depends on all the previous plaintext bytes.



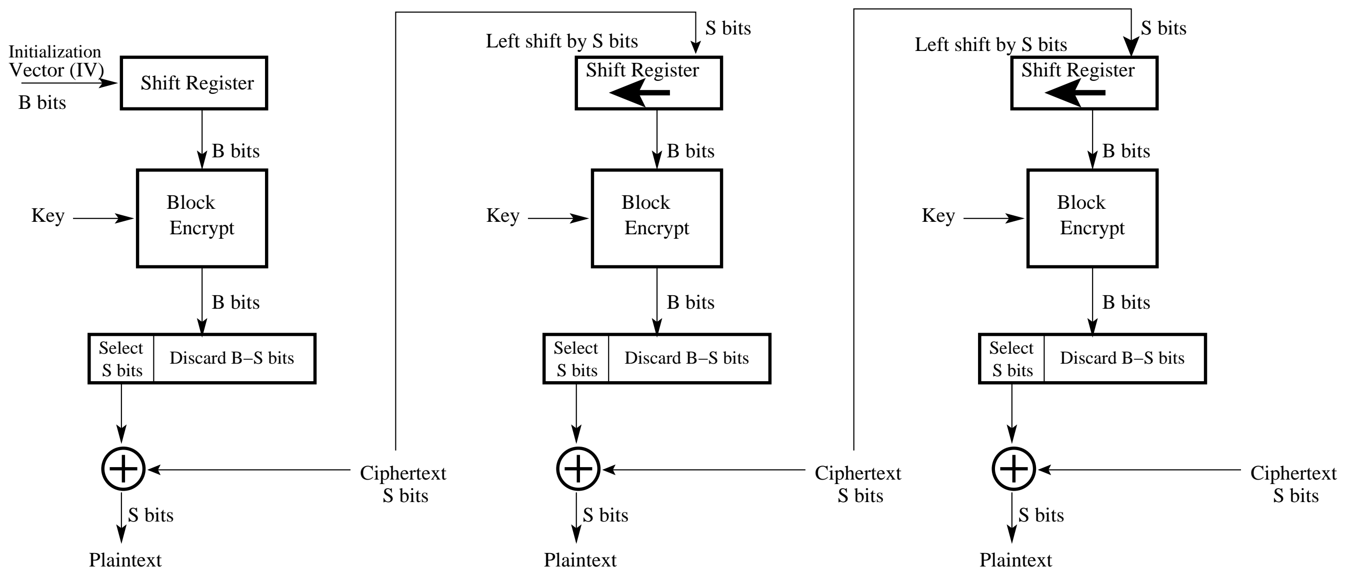


Figure 3: *This figure is from Lecture 9 of "Computer and Network Security" by Avi Kak*

9.11: Block-Cipher Modes of Operation: The Output Feedback Mode (OFB)

- Very similar to the CFB mode. Therefore, this scheme can also be used as a stream cipher.

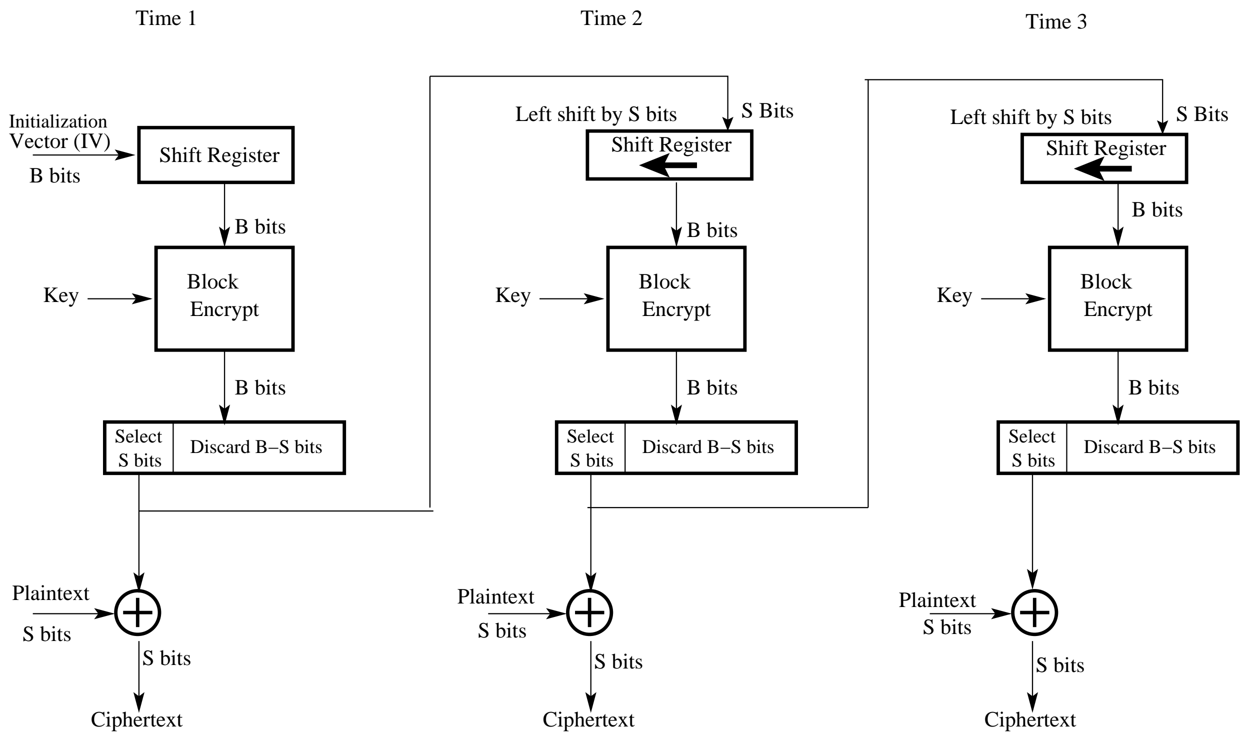
- The only difference between CFB and OFB is that, as shown in Figure 4, now we feed back one byte (the most significant byte) from the output of the block cipher encryption algorithm, as opposed to feeding back the actual ciphertext byte.

- OFB is more resistant to transmission bit errors.

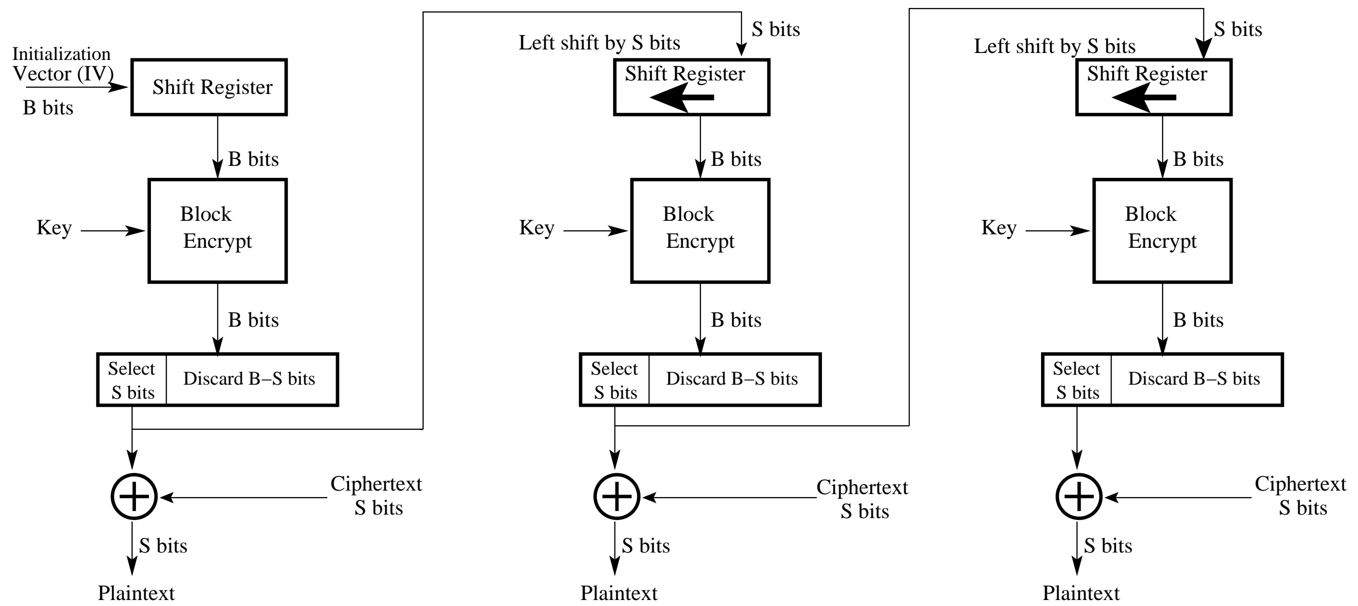
- Considering CFB, let's say that you have encrypted and transmitted the first byte of plaintext. Now suppose this byte is received with a one or more bit errors. In addition to producing an erroneous decryption for the first byte, that error will also propagate to downstream decryptions because the received ciphertext byte is also fed back into the decryption of the next byte.

- On the other hand, what is fed back in OFB is completely locally

generated at the receiver. That is, the information that is fed back is not exposed to the possibility of transmission errors in OFB.



OFB Encryption



OFB Decryption

Figure 4: *This figure is from Lecture 9 of "Computer and Network Security" by Avi Kak*

9.12: Block-Cipher Modes of Operation: The Counter Mode (CTR)

- Whereas the previous two modes, CFB and OFB, are intended to use a block cipher as a stream cipher, the counter mode (CTR) retains the pure block structure relationship between the plaintext and ciphertext.

- In other words, for each b-bit input plaintext block, the scheme produces an b-bit ciphertext block. Furthermore, the block cipher encryption algorithm that is used carries out a b-bits to b-bits transformation.

- In CFB and OFB, on the other hand, whereas the block-cipher encryption algorithm did carry out a b-bits to b-bits transformation, only s bits of plaintext, with $s < b$, were converted into s bits of ciphertext at one time. Moreover, s is typically 8 for the 8 bits of a byte in CFB and OFB.

- As shown in Figure 5 (and as is also true for the OFB mode, but not for the CFB mode), no part of the plaintext is directly exposed to the block encryption algorithm in the CTR mode. The

encryption algorithm encrypts only a b-bit integer **produced by the counter**. What is transmitted is the XOR of the encryption of the integer and the b bits of the plaintext.

- For the counter value, we start with some number for the first plaintext block and then increment this value modulo 2^b from block to block, as shown in Figure 5.

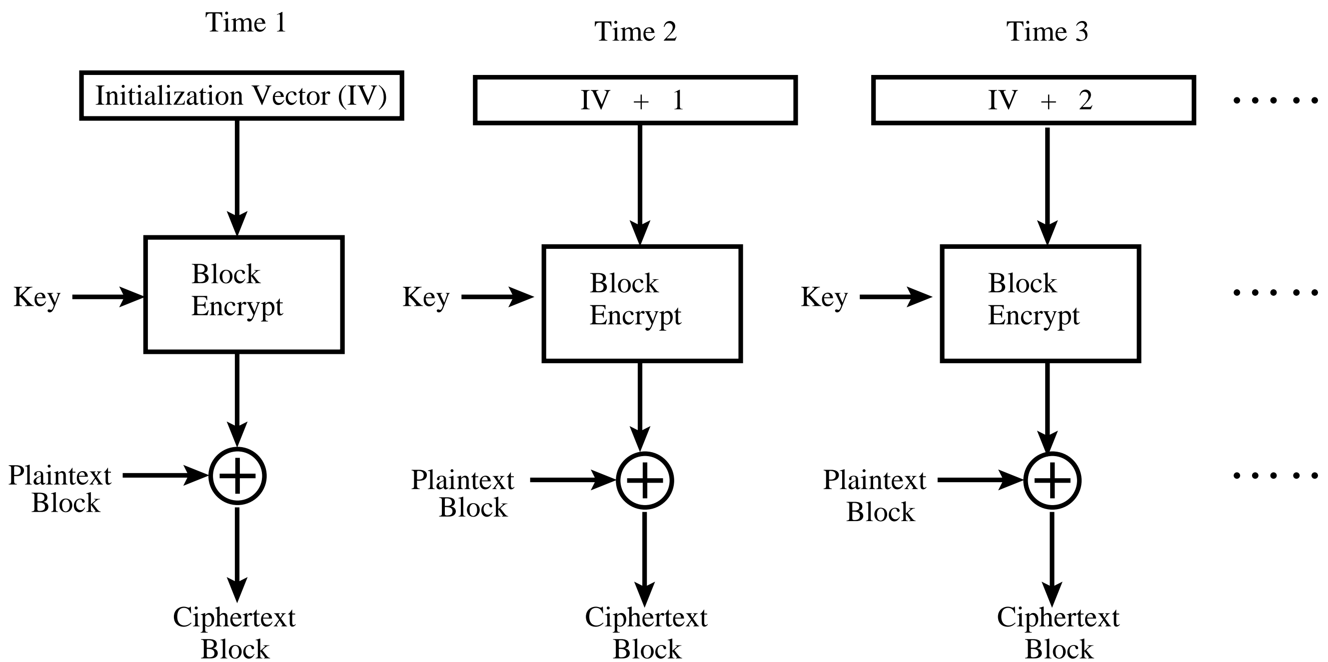
- Note that, as shown in Figure 5, **only the forward encryption algorithm** is used for both encryption and decryption. (This is of significance for block ciphers for which the encryption algorithm differs substantially from the decryption algorithm. AES is a case in point.) (This property of CTR is also true for CFB and OFB modes.)

- Here are some advantages of the CTR mode for using a block cipher:

  - Fast encryption and decryption. If memory is not a constraint, we can precompute the encryptions for as many counter values as needed. Then, at the transmit time, we only have to XOR the plaintext blocks with the pre-computed b-bit blocks. The same applies to fast decryption.

– It has been shown that the CTR is at least as secure as the other four modes for using block ciphers.

– Because there is no block-to-block feedback, the algorithm is highly amenable to implementation on parallel machines. For the same reason, any block can be decrypted with random access.



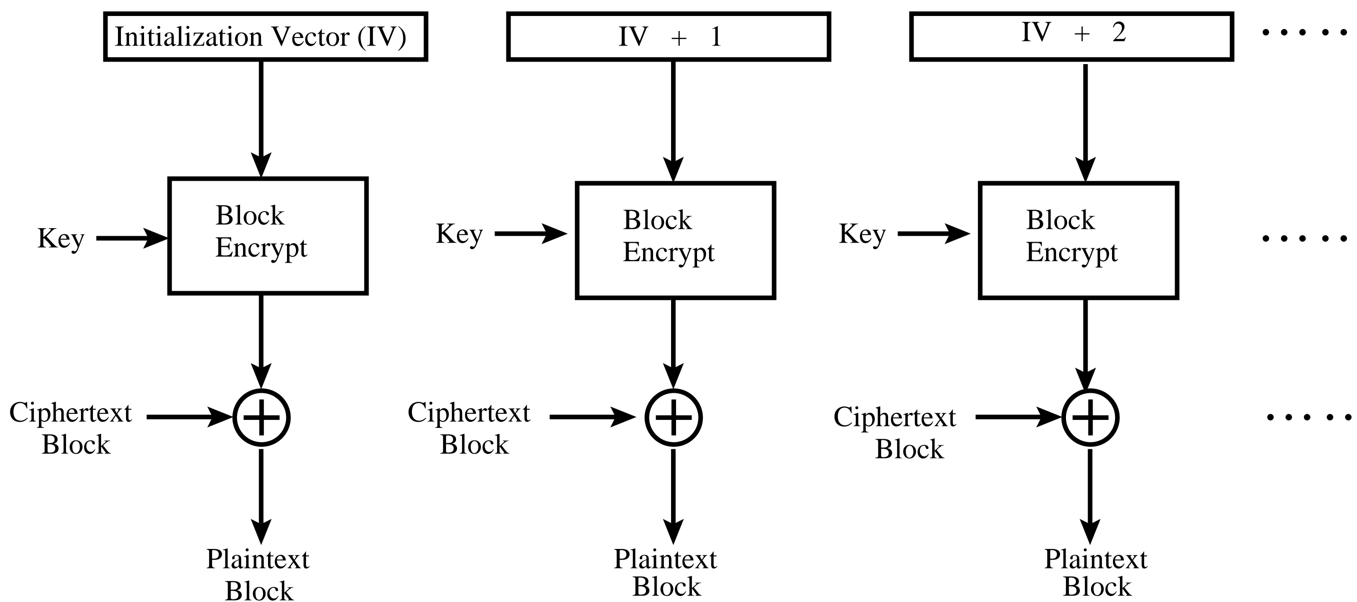


Figure 5: *This figure is from Lecture 9 of "Computer and Network Security" by Avi Kak*

9.13: Stream Ciphers

- Previously we showed how a block cipher, when used in the CFB and OFB modes, can be deployed as a stream cipher.

- So this is a good time to focus on what is generally meant by a **stream cipher** and to talk about real stream cipher algorithms.

- A typical stream cipher encrypts plaintext one byte at a time.

- The main processing step in a general stream cipher is the generation of a **stream of pseudorandom bytes** starting with the encryption key.

- As a new byte of plaintext shows up for encryption, a new byte of the pseudorandom stream also becomes available at the same time and this happens on a continuous basis.

- Obviously, each different encryption key will result in a different stream of pseudorandom bytes. But for a given encryption key,

the stream of pseudorandom bytes will be the same at the both the encryption end and the decryption end of a data link.

- Encryption itself is as simple as it can be. You just XOR the byte from the pseudorandom stream with the plaintext byte to get the encrypted byte.

- You generate the same pseudorandom byte stream for decryption. The decryption itself consists of XORing the received byte with the pseudorandom byte.

- The encryption is shown in the left half and the decryption in the right half of Figure 6.

- For a stream cipher to be secure, the pseudorandom sequence of bytes should have as long a period as possible. (Note that every pseudorandom number generator produces a seemingly random sequence that **eventually** repeats.) The longer the period, the more difficult it is to break the cipher.

- Within the periodicity limitations of a pseudorandom byte sequence generator, the sequence should be as random as possible.

From a statistical point, that means that all of the 256 8-bit patterns should appear in the sequence equally often. Additionally, the byte sequence should be as uncorrelated as possible. This means, for example, that for any two given bytes, the probability of their appearing together should be no greater than what is dictated by their appearance as individual bytes.

- The pseudorandom byte sequence is a function of the encryption key. To foil brute-force attacks, the encryption key should be as long as possible, subject to, of course, all the other practical constraints. A desirable key length these days is 128 bits.

- With a properly designed pseudorandom byte generator, a stream cipher for a given key length can be as secure as a block cipher using keys of the same length.

- The next section presents pseudorandom byte generation for the RC4 stream cipher. (Lecture 10 goes into the subject of pseudorandom number generation for general cryptographic applications.)

- As you would expect, a stream cipher is particularly appropriate for audio and video streaming. A stream cipher could also be

used for browser – web-server links. A block cipher, on the other hand, may be more appropriate for file transfer, etc.

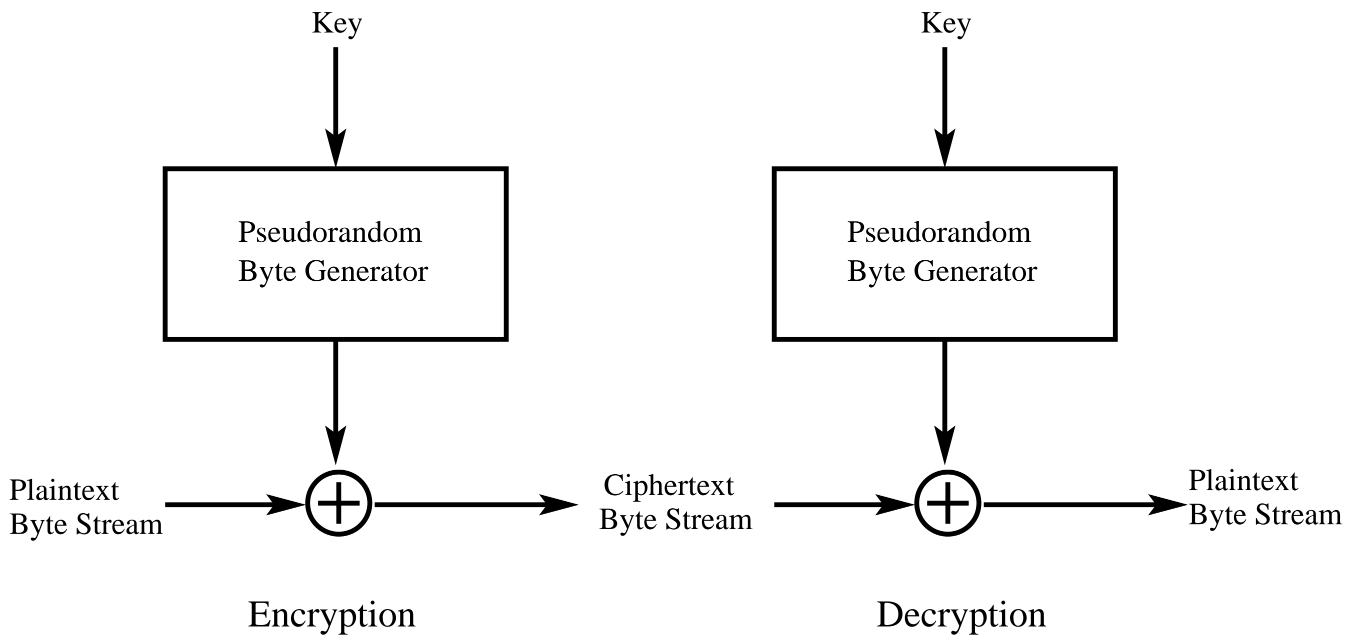


Figure 6: *This figure is from Lecture 9 of "Computer and Network Security" by Avi Kak*

9.14: The RC4 Stream Cipher Algorithm

- As mentioned earlier, a key component of a stream cipher is the pseudorandom byte sequence generator.

- We will now go through the pseudorandom byte sequence generator in the RC4 algorithm.

- RC4 is a variable key length stream cipher with byte-oriented operations.

- Fundamental to the RC4 algorithm is a 256 element array of 8-bit integers. It is called the **state vector** and denoted S.

- The state vector is initialized with the encryption key. The exact initialization steps are as follows:

  – The state vector S is initialized with entries from 0 to 255 in the ascending order. That is

$$S[0] \quad = \quad 0x00 \; = \; 0$$

$$
\begin{aligned}
S[1] &= 0x01 = 1 \\
S[2] &= 0x02 = 2 \\
S[3] &= 0x03 = 3 \\
&\ldots\ldots \\
&\ldots\ldots \\
S[255] &= 0xFF = 255
\end{aligned}
$$

– The state vector S is further initialized with the help of another temporary 256-element vector denoted T. This vector also holds 256 integers. The vector T is initialized as follows

  * Let's denote the encryption key by the vector K of 8-bit integers. Suppose we have a 128-bit key. Then K will consist of 16 non-negative integers whose values will be between 0 and 255.

  * We now initialize the 256-element vector T by placing in it as many repetitions of the key as necessary until T is full. Formally,

$$T[i] \quad = \quad K[i \mod keylen] \qquad for\ 0 \leq i < 255$$

where $keylen$ is the number of bytes in the encryption key. In other words, $keylen$ is the size of the key vector K when viewed as a sequence of non-negative 8-bit integers.

– Now we use the 256-element vector T to produce the **initial permutation** of S. This permutation is according to the following formula that first calculates an index denoted j and then swaps the values $S[i]$ and $S[j]$:

```
j  =  0
for i = 0 to 255
    j  =  ( j + S[i] + T[i] )  mod  256
    SWAP  S[i], S[j]
```

This algorithm is generally known as the **Key Scheduling Algorithm** (KSA).

– There is no further use for the temporary vector T after the state vector S is initialized as described above.

– Note that the encryption key is used only for the initialization of the state vector S. It has no further use in the operation of the stream cipher.

– Note also that initialization procedure for the state S is just a permutation of the integers from 0 through 255. Each integer in this range will be in one of the elements of S after initialization. This happens because all that the initialization does is to swap the elements of S according to the secret key.

- Now that the state vector S is initialized, we are ready to describe how the **pseudorandom byte stream** is generated from the state vector. Recall that when you are using a stream cipher, as each byte of the plaintext becomes available, you XOR it with a byte of the pseudorandom byte stream. The output byte is what is transmitted to the destination.

- The following procedure generates the pseudorandom byte stream from the state vector

```
i, j  =  0
while ( true )
    i  =  ( i + 1 )  mod  256
    j  =  ( j + S[i] )  mod  256
    SWAP S[i], S[j]
    t  =  ( S[i]  +  S[j] )  mod  256
    k  =  S[t]
    output k
```

  Note how the state vector S changes continuously by the swapping action at each pass through the `while` loop. In other words, the state of the pseudorandom number generator changes dynamically as the the numbers are being generated.

- The above procedure spits out values of the variable k for the pseudorandom byte stream. The plaintext byte is XORed with this byte to produce an encrypted byte.

- The pseudorandom sequence of bytes generated by the above algorithm is also known as the **keystream**.

- Theoretical analysis shows that for a 128 bit key length, the period of the pseudorandom sequence of bytes is likely to be greater than 10^{100}.

- RC4 is used in the **SSL/TLS** (Secure Socket Layer / Transport Layer Security) standard for secure communications between web browsers and web servers.

- RC4 is also used in the **WEP** (Wired Equivalent Privacy) protocol and the newer WiFi Protected Access (**WPA**) protocol that are part of the IEEE 802.11 wireless LAN standard.

9.15: Strengths and Weaknesses of the RC4 Stream Cipher

- Its simplicity is its greatest asset.

- Because all operations are at the byte level, the cipher possesses fast software implementation. For that reason, RC4 remains the mostly widely used software stream cipher.

- Although it is still used extensively, RC4 is not considered to be sufficiently secure any longer. In their paper entitled "Weaknesses in the Key Scheduling Algorithm of RC4," the authors Fluhrer, Mantin, and Shamir show that RC4 possesses a large number of **weak keys**. With these keys, a knowledge of just a few key bits can be used to make strong inferences about the state vector used for encryption.

- Efforts are underway (as, for example, in the **eSTREAM** project launched by the European Network of Excellence for Cryptology) to develop more secure stream ciphers.

- We will next focus briefly on some specific weaknesses of RC4 as it is used in WEP for wireless security in home networks. To understand these weaknesses, you must understand how RC4 is used wireless network communications.

9.16: Problems with WEP

- WEP is a protocol for encrypting packets that are transmitted over an IEEE 802.11 wireless communication network. It is the most commonly used protocol for home and small business wireless networks.

- The WEP protocol requires each packet to be encrypted separately with its own RC4 key. So if a TCP packet contains, say, a payload of 1024 bytes, those bytes would be encrypted by RC4 using a key specific to that packet.

- The RC4 key for each packet is a simple concatenation of a 24-bit Initialization Vector (IV) and the root key. (The root key corresponds to the pass phrase you enter in your home wireless router.)

- WEP then computes the CRC32 checksum of the data to be encrypted in the packet. (CRC stands for Cyclic Redundancy Check. It is a generalization of the commonly used parity check that is used to guard against data corruption during transmission. CRC32 gives us a 32-bit checksum. Think of it as a 32-bit digital

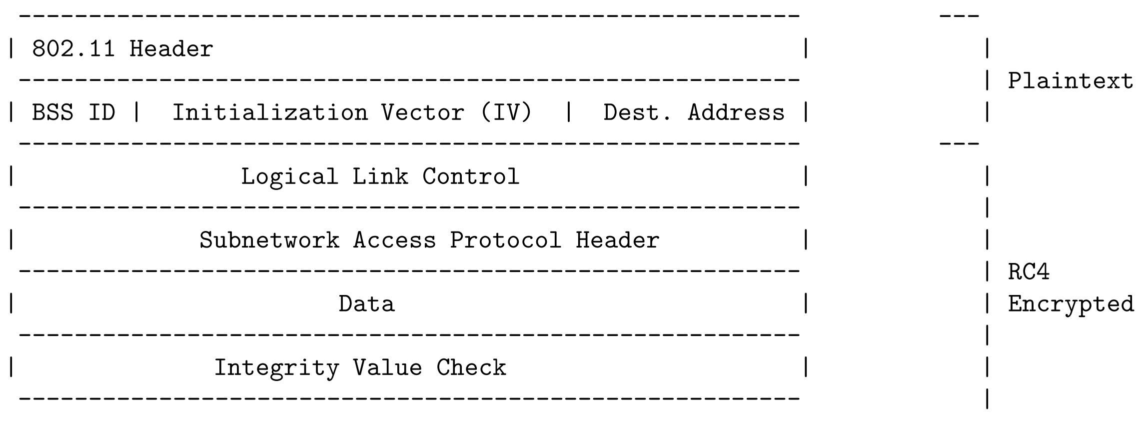
signature.) In WEP, this CRC32 signature is called **Integrity Check Value** (ICV). [Finding CRC32 of a binary data stream amounts to dividing the data bit pattern (which could be the bits in an entire file) by an irreducible (or sometimes reducible) polynomial of degree 32. This would obviously leave a residue polynomial whose highest degree would only be 31. The bit pattern corresponding to the residue would therefore only be 32 bits long. Note also CRC1, which is the same thing as using a parity bit for error detection, amounts to using $x + 1$ as the divisor polynomial. While it is possible (and fairly common) to use reducible polynomials, their error detection capabilities are less effective.]

- The RC4 key for a packet is then used to encrypt the data followed by its ICV.

- The biggest problem with WEP in a typical usage scenario is that the root key remains fixed for long periods of time (in home use, people almost never change their root keys) and the IV has only 24 bits in it. **This implies that distinct keystreams can be generated for only 2^{24} different packets.** This implies that the same keystream will be used for different packets in a long session. How frequently that can happen depends on how the IVs are generated. Note that if the same keystream S is used for two different packets P_1 and P_2. Then, an XOR of the corresponding ciphertexts becomes independent of the keystream because

$$C_1 \oplus C_2 \quad = \quad (P_1 \oplus S) \oplus (P_2 \oplus S) \quad = \quad P_1 \oplus P_2$$

- Since the 3-byte IV is prepended to the root key and since the IV is sent in plaintext, anyone with a packet sniffer can directly see the first three bytes of the RC4 key used for a packet. An 802.11 frame that is encrypted with WEP looks like



- In their paper "Breaking 104 bit WEP in less than 60 seconds," Tews, Weinmann, and Pyshkin have shown how an attacker can discover the additional bytes in the RC4 key (therefore the bytes of the WEP root key) by analyzing the easily identified ARP (Address Resolution Protocol) packets.

- Note that WPA also uses RC4. WPA provides enhanced security because it uses a 48-bit Initialization Vector. Additionally, WPA hashes the Initialization Vector before combining it with the root key.