**Network Security**

**Professor Gaurav S. Kasbekar**
**Department of Electrical Engineering**
**Indian Institute of Technology, Bombay**

**Week - 02**
**Lecture - 13**
**Principle of Cryptography: Part 3**

Hello, in this lecture, we continue our discussion of the principles of cryptography. In the previous lecture, we discussed block ciphers. In this lecture, we will discuss how to use block ciphers to encrypt long messages. So, suppose a block cipher is used to encrypt a long message. Recall that a block cipher divides the message into blocks of k bits each.

One natural approach to encrypting the long message is to independently encrypt each block of k bits using the block cipher. This is known as "Electronic Codebook". But the electronic codebook has some disadvantages, as we'll see. So, in particular, it is possible that the plaintext in two or more blocks is identical. In this case, the corresponding ciphertext is also identical in the corresponding blocks.

So, hence, in some situations, by observing the fact that the ciphertext is identical in two or more blocks, the intruder may be able to obtain some useful information. So, the intruder, in particular, sees that the plaintext in the corresponding blocks is identical. The intruder gets to know this just by looking at the ciphertext blocks. And in some situations, the intruder can also obtain some other information when the electronic codebook is used. So, we'll discuss an example.

Suppose the intruder knows that the plaintext is an alphabetically sorted list of employees and their salaries being sent from management to the cash section. It looks like this. We have a set of rows, and in each row, we have the name of the employee, then the position of the employee, and the salary. So, this is a set of rows like this. We use a block cipher with k = 64 and electronic codebook for encrypting this list of employees and their corresponding information.

Each line is 64 bytes long, and each line is divided into 8 blocks, each of which is represented using 8 bytes. So, that is shown here. So, each line is divided into 8 blocks of

8 bytes each. And a point to notice is that a new block begins between the 1000's and 10,000's digits of the salary field. That can be seen from here.

So, this 1 is the 1000's digit, and 2 is the 10,000's digit. So, the block begins between the 2 and the 1. That is between the 1000's and the 10,000's place. Likewise, here the block begins between 3 and 8. So, that's a point to note here.

So, just by observing the ciphertext blocks, can the intruder gain some information? In this example, where the electronic codebook is used, the intruder can find out which sets of employees have identical salaries. So, just by observing the last 2 blocks in every line, the intruder can find out when 2 employees have the same salary. So, for 2 different rows, if the last 2 blocks are the same, then it means that the corresponding 2 employees have the same salaries. So, the intruder can find out which sets of employees have identical salaries.

So, it is not difficult to know the alphabetically sorted list of employee names. So, the intruder might know that already. And then, by finding out which rows have the same last 2 blocks, the intruder can find out which employees have identical salaries. The intruder can gain even more information. Notice that, if this second last block of a line is the same for 2 different lines, then it means that the corresponding employees have salaries in the same 10,000's range.

So, for example, consider this line corresponding to Hoover and this line corresponding to Stern. So, they have the same salary in the $10,000^{th}$ place; that is, 3. So, their salary is in the 30,000 range. So, this one is 34,000 roughly, that is in the 30,000 range, and this is also in the 30,000 range. So, hence, this $7^{th}$ block corresponding to these 2 rows are identical in these 2 rows.

So, the intruder can find out which sets of employees have salaries in the same 10,000's range. The intruder can not only get some useful information but can also modify the message to alter the meaning of the message. For example, if the intruder is one of the employees, then the intruder can modify the message to change his or her own salary to match that of any other employee by copying the ciphertext blocks from that employee to the corresponding blocks of his or her own entry. For example, if Woods is an intruder, then Woods might copy these 2 blocks from Bush in place of the corresponding blocks in this row. So, then the salary changes from around 21,000 to around 623,000.

So, this modification is easy to do in the case of an electronic codebook. So, the electronic codebook has these disadvantages. Hence, we typically do not use the electronic codebook

in practice. So, ECB is rarely used in practice because of disadvantages like in the above example. So, the challenge is that we want to apply a block cipher to encrypt a long message, such that identical blocks of plaintext result in different blocks of ciphertext with a high probability.

So, if two or more plaintext blocks are the same, even then we would like the corresponding ciphertext blocks to be different with a high probability. So, to do this, we introduce some randomness in the plaintext. So, we add some randomness to the plaintext. We'll study different techniques for achieving this, and they all use randomness to achieve this property that identical blocks of plaintext map to different blocks of ciphertext with high probability. So, first consider a simple technique.

For $i = 1, 2, 3, ...$, let $m(i)$ be the i[th] plaintext block, and let $c(i)$ denote the i[th] ciphertext block. Let S be the symmetric key used, and let $K_S(m)$ denote the ciphertext obtained by encrypting the plaintext message m using the block cipher. So, the scheme is used as follows. For each $i = 1, 2, 3, ...,$, the sender first generates a random k-bit number, say $r(i)$, and then calculates $c(i) = K_S\big(m(i) \oplus r(i)\big)$. This $\oplus$ denotes the bitwise exclusive OR of $m(i)$ and $r(i)$.

- For $i = 1,2,3 \ldots .$, let:
  - $m(i)$: $i'$th plaintext block
  - $c(i)$: $i'$th ciphertext block
- Let:
  - $S$: the symmetric key used
  - $K_S(m)$: the ciphertext obtained by encrypting message $m$ using the block cipher
- Suppose for each $i = 1,2,3 \ldots .$, sender:
  - generates a random $k$ bit number $r(i)$
  - calculates $c(i) = K_S(m(i) \oplus r(i))$, where $\oplus$ denotes bitwise exclusive OR
- Sender sends $c(1), r(1), c(2), r(2), c(3), r(3)$ ... to receiver
- Receiver recovers plaintext using $m(i) = K_S(c(i)) \oplus r(i)$, for $i = 1,2,3$ ...

So, first the sender finds out $m(i) \oplus r(i)$ (bitwise), and then encrypts that to get $K_S(m(i) \oplus r(i))$. And that is the ciphertext $c(i)$. The sender then sends $c(1)$, $r(1)$, $c(2)$, $r(2), c(3), r(3), ....$ So, the sender sends the generated ciphertext along with the random number that was used to generate each block of ciphertext. So, to recover the plaintext, the receiver reverses this process.

So, the receiver takes the ciphertext block and decrypts it to get this decryption as shown here, so $K_S(c(i))$, and then XORs that with $r(i)$ to get $m(i)$. So, from this equation, we see that first decrypting $c(i)$ and then XORing the result with $r(i)$ will recover the plaintext $m(i)$. So, $r(i)$ is sent in plaintext form, and hence the intruder can find that out by sniffing the communication channel. But the intruder does not know S, the secret key, and hence the intruder cannot perform this step of decrypting the ciphertext to get $K_S(c(i))$. Hence, the intruder cannot calculate $m(i)$ because $K_S(c(i))$ is required to calculate $m(i)$.

Now, let's go back to the problem of identical plaintext blocks mapping to identical ciphertext blocks in ECB. In this scheme, even if the two plaintext blocks $m(i)$ and $m(j)$ are the same, because of these random strings that are XORed into the plaintext, $c(i)$ and $c(j)$ are different, even though $m(i)$ and $m(j)$ are the same. So, with high probability, $c(i)$ and $c(j)$ are different. That is because $r(i)$ is not equal to $r(j)$ with high probability. $r(i)$ and $r(j)$ are chosen randomly.

They are independent k-bit strings, so with a high probability, they are different. Hence, the ciphertext blocks $c(i)$ and $c(j)$, corresponding to identical plaintext blocks $m(i)$ and $m(j)$, are different with a high probability. Let's consider an example to illustrate this scheme. Suppose k = 3, and the block cipher in this table is used. We discussed the same block cipher in an earlier example.

If the plaintext block is 000, then the corresponding ciphertext block is 110, and so on and so forth. Suppose the plaintext is 010010010. There are 3 identical blocks, each one being 010. If it is directly encrypted, that is, without adding any randomness, then what will be the ciphertext? So, this is divided into 3 blocks, each one being 010.

The ciphertext corresponding to 010 is 101. So, hence, the ciphertext is 101101101. If the intruder sniffs this ciphertext, then they can find out that the 3 plaintext blocks are the same because that can be inferred from the fact that the 3 ciphertext blocks are the same. Now, suppose we add some randomness, $r(1) = 001, r(2) = 111$, and $r(3) = 100$.

Then, what will be the corresponding ciphertext blocks? So, first, we have to bitwise XOR $r(i)$ with $m(i)$. So, the first block of the plaintext is 010. And that is to be XORed with $r(1)$, that is, 001. And the result is 011.

Now, we encrypt this by consulting the table there. So, we see that the ciphertext corresponding to 011 is 100. Hence, $c(1) = 100$. So, we get that $c(1) = 100$. Similarly, now to get $c(1)$, we first find out $m(2) \oplus r(2)$.

So, that's $010 \oplus 111 = 101$. And then, we find out the ciphertext corresponding to this plaintext, and hence, we get that $c(2)$ is the ciphertext block corresponding to 101, that's 010, and so on and so forth. So, in this example, we get that $c(1) = 100$ as calculated here, then $c(2) = 010$, and $c(3)$ can be similarly found to be 000. So, we see that the 3 ciphertext blocks are different even though the corresponding plaintext blocks are the same.

So, that's the advantage of using this randomness. Identical plaintext blocks map to different ciphertext blocks with a high probability. Now, this scheme that we just discussed solves the problem of identical plaintext blocks mapping to identical ciphertext blocks, but it has a shortcoming that the sender must transmit twice as many bits as before. For every plaintext message $m(i)$, the transmitter must transmit $c(i)$ as well as $r(i)$. So, the number of bits that need to be transmitted is doubled.

So, we want a scheme that does not have this shortcoming. Can we design a scheme in which the sender sends only one random k-bit number and still identical plaintext blocks map to different ciphertext blocks with a high probability? So, we don't send these numbers $r(1), r(2), r(3)$, and so on. We send only one random k-bit number at the beginning. Still, we want identical plaintext blocks to map to different ciphertext blocks with a high probability.

A scheme that achieves this is Cipher Block Chaining (CBC). So, in this scheme, before encryption of the message starts, the sender generates a random k-bit number. Let's call that $c(0)$. It's called the Initialization Vector (IV). This initialization vector (IV) is also used in other schemes that we'll discuss later.

Again, for $i = 1, 2, 3, ...$, let $m(i)$ denote the $i^{th}$ plaintext block and $c(i)$ denote the $i^{th}$ ciphertext block. Initially, the sender sends the IV $c(0)$ to the receiver in plaintext form. The encryption scheme is as follows. For each $i = 1, 2, 3, ...$, the sender first XORs $m(i)$ with $c(i-1)$ and then encrypts the result to get $c(i)$. So, $c(i) = K_S\big(m(i) \oplus c(i-1)\big)$.

So, for $i = 1, c(i-1)$ is the IV, that is, $c(0)$. The IV is XORed with $m(i)$, and the result is encrypted to get $c(i)$. Then, for the second block, $c(i-1) = c(1)$, so $c(1)$, the first ciphertext block, is XORed with the second plaintext block, and the result is encrypted to get $c(2)$, the second ciphertext block, and so on and so forth. To recover $m(i)$, the receiver reverses the process, so the receiver first decrypts this to get $m(i) \oplus c(i-1)$, and then the receiver XORs the result with $c(i-1)$. So, $m(i) = K_S\big(c(i) \oplus c(i-1)\big)$.

- Then for each $i = 1,2,3 \ldots$, sender:
  - calculates $c(i) = K_S(m(i) \oplus c(i-1))$ and sends it to receiver
- Recovery of $m(i)$ by receiver:
  - $m(i) = K_S(c(i)) \oplus c(i-1)$
- Even if $m(i)$ and $m(j)$ are the same, $c(i)$ and $c(j)$ are different with a high probability
  - since $c(i-1) \neq c(j-1)$ with a high probability

So, even if $m(i)$ and $m(j)$ are the same, that is, two plaintext blocks are identical, $c(i)$ and $c(j)$ are different with the high probability because $c(i-1) \neq c(j-1)$ with the high probability. So, the plaintext block is not encrypted directly, but it is first XORed with the previous ciphertext block, and then the result is encrypted. And the previous ciphertext block is different for i and j with high probability. So, for this reason, $c(i)$ and $c(j)$ are different with high probability. $c(0)$ is sent in plaintext form, and it can be sniffed by the intruder who sniffs the communication channel, but the intruder does not know S and hence cannot calculate $m(i)$.

So, to calculate $m(i)$, the intruder first needs to decrypt $c(i)$; for that, the secret key S is required, which the intruder does not have. So, hence, the intruder is not able to get the plaintext. So, the overhead of this scheme is only 1 extra k-bit number needs to be sent for an entire message. So, that's a very low overhead. Typically, a message would consist of a large number of blocks of k bits each.

But the overhead is a constant. It is just k bits regardless of the length of the message. So, hence, the overhead is not much. Here's an example that illustrates the operation of CBC. Suppose, again, that k = 3 and the block cipher in the table is used.

The plaintext is the same as in the previous example, 010010010. So, there are 3 plaintext blocks, and they are identical. Suppose the IV $c(0) = 001$. Then let's calculate the ciphertext blocks. So, recall that the IV is first XORed with the first block of the plaintext.

So, we calculate 010 XOR the IV, that is, 001. So, the result is 011. So, this is now encrypted to get the first ciphertext block. So, we calculate $K_S(011) = c(1)$. So, if 011 is encrypted, then to get the corresponding ciphertext, we have to look up this table; 011, the corresponding ciphertext is 100.

So, hence, the first ciphertext block is 100. So, $c(1)$ = 100 as calculated here. Similarly, you can calculate $c(2)$ and $c(3)$, and they turn out to be 000 and 101, respectively. So, the 3 ciphertext blocks are different even though the corresponding plaintext blocks are the same. A simple exercise is to check that the receiver can recover the plaintext message using the ciphertext blocks and the IV.

So, you can use the decryption process described on a previous slide to check that the receiver can recover the plaintext. So, the cipher block chaining is an efficient method under which identical blocks of plaintext are mapped to different blocks of ciphertext with a high probability. Now, let's discuss the reason why the initialization vector (IV) is used. Suppose we did not use any initialization vector; it is equivalent to using $c(0) = 0$. So, the transmitter does not send any initialization vector, and the receiver just uses $c(0) = 0$.

So, the security of CBC would be adversely affected in some cases. Here are some examples. One is, suppose in the above example, the encrypted file of employees and salaries is sent every week. Then, the 2 files sent in 2 different weeks would be identical up to the point of the first person whose salary changed from the last week's salary. So, the first several bytes of the files are the same since the list of people as well as the salaries have not changed.

So, the intruder could find out the first person whose salary changed from the last week's salary. So, that's because no initialization vector is used. If the first few bytes of two different plaintext messages are the same, then the corresponding ciphertext blocks will also be the same. This happens because the IV is not used. Another example is, suppose every day an army general sends a message, continue holding your position.

Then, the ciphertext will be the same every day because the plaintext message is exactly the same. This will be the case until the general sends some other message. For example, start bombing. Then, by observing the fact that the ciphertext has changed, the enemy, who is the intruder in this case, gets to know that something has changed, and then that is an alert to the enemy that some change has happened. So, some information about the plaintext has thus leaked out to the intruder.

So, that's the disadvantage of not using the initialization vector. So, if we choose a randomly chosen initialization vector for every message, then even if the same message is sent repeatedly, the ciphertext is completely different each time. That's the advantage of using an initialization vector. And for the same reason, the schemes that we will discuss next, they also use a random initialization vector. So, that concludes our discussion of CBC.

Next, we discuss another scheme, Output Feedback Mode (OFB). Suppose, again, that a block cipher of block length 64 bits is used. Then, under OFB, first a random 64-bit number is generated. Let's call that $b_0$. This is the initialization vector.

Then, OFB operates as follows. $b_0$ is encrypted using the block cipher and the secret key to get $b_1$, which is a 64-bit number. Then, $b_1$ is encrypted to get $b_2$, and $b_2$ is encrypted to get $b_3$, and so on and so forth. So, this result obtained, $b_1 | b_2 | b_3$, and so on. This is known as the one-time pad (OTP).

So, this one-time pad is XORed with as many bits of the plaintext as necessary. So, to encrypt the plaintext message, we XOR the plaintext message with as many bits of the one-time pad, $b_1, b_2, b_3$, and so on as necessary. The result is transmitted along with the initialization vector to the receiver. The receiver, once the receiver receives the initialization vector, the receiver is able to calculate the same one-time pad by using the same process as the transmitter used. And then, the receiver XORs the one-time pad with the ciphertext to recover the plaintext.

So, at the center, the ciphertext is the XOR of the one-time pad with the plaintext. Now, if the ciphertext is XORed with the one-time pad, then clearly we'll get back the original plaintext. This shows a block diagram of the transmitter side of OFB, that is, the encryption side. We take an initialization vector and then encrypt it, so this is the initialization vector $b_0$; it is encrypted to get $b_1$, so this is $b_1$, and then $b_1$ is encrypted to get $b_2$, $b_2$ is encrypted to get $b_3$, and so on. And the plaintext is XORed bitwise with the one-time pad, $b_1, b_2, b_3$, and so on, to get the corresponding ciphertext.

This shows the decryption process. Again, the same one-time pad is generated by starting from the initialization vector, encrypting it to get $b_1$, then $b_1$ is encrypted to get $b_2$, $b_2$ is encrypted to get $b_3$, and so on. So, here, exactly the same one-time pad is generated as the one generated over here. So, this one-time pad is $b_1, b_2, b_3$, and so on, and the one-time pad is XORed bitwise with the ciphertext to recover the plaintext. So, OFB has some advantages over CBC.

One advantage is that the one-time pad can be generated in advance before the message to be encrypted is known. So, when the message to be encrypted arrives, it just needs to be XORed with the one-time pad. There are no computationally expensive operations, such as encryption, required at that point. So, hence, for this reason, OFB is suitable for real-time communication, such as in cellular networks or Wi-Fi. Or rather, OFB is suitable for real-time communication, such as when voice calls or video calls need to be communicated.

So, that's because the latency has to be very low. So, lowering the encryption time helps. So, for real-time communication, such as voice calls or video calls, OFB is suitable. So, the one-time pad can be generated in advance, and then, to encrypt the plaintext, it just needs to be XORed with the one-time pad, which has already been generated. So, that reduces the delay.

So, another advantage is that if some of the bits of the ciphertext get corrupted during transit, then only those bits of the decrypted plaintext at the receiver are corrupted. Recall that, in contrast, in CBC, if block $c(n)$ is corrupted, then the decrypted $m(n)$ is completely corrupted, and the same portion of the decrypted $m(n+1)$ as corrupted in $c(n)$ is also corrupted. So, that's because if some of the bits of the ciphertext get corrupted, suppose some of the bits of ciphertext block $c(n)$ get corrupted; then they are XORed with the corresponding bits of the one-time pad, and as a result, only the corresponding bits of the decrypted plaintext will be corrupted, whereas in CBC, that is not the case. That's because, in the case of CBC, $c(n)$ is first decrypted, and then the result is XORed with $c(n-1)$. Because of this, this property holds that the decrypted $m(n)$ is completely corrupted, and the same portion of the decrypted $m(n+1)$ is corrupted, and $c(n)$ is corrupted if the block $c(n)$ is corrupted.

Next, we discuss another scheme for encrypting long messages, that is, Counter Mode (CTR). It is similar to OFB in the sense that a one-time pad is generated, and it is XORed with the plaintext to get the ciphertext. But CTR operates as follows. It first encrypts the initialization vector, then increments the initialization vector to get IV + 1, and then encrypts the result, and then again increments it to get IV + 2, and then encrypts it, and so on and so forth, to get successive blocks of the one-time pad. So, that is shown here.

This is the initialization vector. This is the IV + 1 obtained by incrementing IV. Then, this is IV + 2, and then we get IV + 3, and so on. So, each one is encrypted, and the result is the corresponding one-time pad. So, the one-time pad is over here.

And then, that is XORed with the plaintext message to get the ciphertext message. So, the advantages are, as in OFB, the cryptography can be pre-computed, and encryption is just a XOR operation. So, the one-time pad can be generated in advance, and then encryption is just a XOR operation. It is well suited to a multi-processor machine because different blocks of the one-time pad can be generated in parallel on a multi-processor or multicore machine. Another advantage is that the message can be decrypted starting at any point rather than being forced to start at the beginning.

This is because if we want to decrypt, suppose we want to decrypt the nth block of the ciphertext at the beginning; in that case, we just find out IV + $n - 1$. And then, encrypt that to get the corresponding one-time pad block, and then just XOR that with the $n^{th}$ block of the ciphertext to get the corresponding plaintext. So, the message can be decrypted starting at any point rather than being forced to start at the beginning. So, hence, CTR is suitable for encrypting randomly accessed files. This is similar to CBC, but it is unlike OFB.

Now, let's consider the case where OFB and CTR are used, but we use the same IV for two different messages. Suppose OFB or CTR is used to encrypt two different messages sent independently, possibly on different days, using the same IV and the same key. Suppose the intruder sniffs the channel during the entire transfer of both the messages. Can the intruder gain some information about the plaintext? So, the intruder can get some information.

That's because, to illustrate this, suppose the message sent on the first day is $M_1$. The plaintext message sent on the first day is $M_1$. And the IV is the same on the two days. So, suppose the one-time pad is, say, P. So, P is the one-time pad used on the first day, and this is the ciphertext.

That is, let's call it $C_1$. $C_1$ is the ciphertext message corresponding to the message sent on the first day. Suppose the message sent on the second day is $M_2$. It is XORed with the same one-time pad. So, notice that the one-time pad is the same on both days because the IV is the same.

The corresponding ciphertext is $C_2$. Now, if the intruder calculates $C_1 \oplus C_2$, then the result is $M_1 \oplus M_2$. So, the intruder is able to get the XOR of the two plaintext messages. So, the intruder has obtained some information about the plaintext messages. So, hence, this is a weakness in these methods, OFB and CTR.

This weakness exists only if the same IV is used for two different messages. So, hence, different IVs should be used for different messages. This concludes our discussion of using block ciphers to encrypt long messages. Thank you.