Network Security

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

Week - 02

Lecture - 12

Principle of Cryptography: Part 2

Hello, in this lecture, we continue our discussion of the principles of cryptography. We discussed monoalphabetic and polyalphabetic ciphers in the previous lecture. We now discuss block ciphers, which are another kind of symmetric key based ciphers. Block ciphers are symmetric key based. They are also used in many internet protocols currently.

For example, PGP is a protocol for securing email, and SSL is a protocol for securing TCP connections. PGP as well as SSL use block ciphers. Examples of popular block ciphers used in the internet are as follows. One is, Data Encryption Standard (DES), then 3DES, Advanced Encryption Standard, (AES). These are popular block ciphers used in the internet.

The way a block cipher operates is as follows. The message to be encrypted is first represented using its binary representation as a string of bits. It is broken into blocks of k bits each. For example, k might be 64. Each block is encrypted independently.

And for transforming a particular block of k bits, the encryption process is as follows. A one-to-one function is used to map the k bit block of plaintext to a k bit block of ciphertext. We'll discuss what the one-to-one function is. But notice that the function has to be one-to-one because we want to recover the plaintext from the ciphertext. So, if it were a many-to-one function, then we could not recover the plaintext from the ciphertext uniquely.

So, hence, we use a one-to-one function to map the k bit block of plaintext to a k bit block of ciphertext at the sender side, and at the receiver, we apply the reverse function to get back the k bit block of plaintext from the k bit block of ciphertext. This is a simple example of a block cipher. We use block size k = 3, so the message to be encrypted is broken into blocks of 3 bits each, and we use the mapping in the table for encryption. So, for example, if the plaintext block is 000, then the corresponding ciphertext block is 110. If the plaintext block is 001, then the ciphertext block is 111, and so on and so forth.

If the plaintext block is 100, then the ciphertext block is 011, etc. So, suppose the plaintext is this. Let's now find out the corresponding ciphertext. So, recall that the first step is to break down the plaintext into blocks of k bits each. Since k is 3, we divide the plaintext into one block of 3 bits, then another block of 3 bits, and so on and so forth.

So, we have divided the plaintext in this example into four blocks of 3 bits each. Now, we transform each block independently using this block cipher. The first block is 010. It is to be transformed to 101. So, the first block of the ciphertext is 101.

The next block of the plaintext is 110. It is to be transformed to 000. The next block of the plaintext is 001. The corresponding ciphertext block from this table is 111, and so on and so forth. So, the ciphertext is this.

So, this is a simple example, which shows the operation of a block cipher. What is the key in a block cipher? The key in a block cipher is the one-to-one mapping used to generate the ciphertext from the plaintext. So, for example, in the previous slide, this table is the key used to transform the plaintext to the ciphertext and to get back the plaintext from the ciphertext. Now, consider a k bit block cipher.

What is the number of possible mappings from the plaintext to the ciphertext? My claim is that the number of possible mappings is $(2^k)!$. That's because there are 2^k possible strings of k bits each. And these can be permitted in the number of possible permutations of these, 2^k possible strings is $(2^k)!$. So, hence, the number of possible mappings is $(2^k)!$.

Now, a brute force attack tries out all possible keys. That is, it tries out all possible mappings from the plaintext to the ciphertext. So, to defend against such brute force attacks, k must be large. Typical values used in practice are k = 64 or larger. For example, k = 128, 256, and so on.

If k is 64, then $2^{64} \approx 1.8 \times 10^{19}$. So, we see that this factorial is an extremely large number. So, there are a large number of mappings. To implement a k bit block cipher using a look-up table, like the one on the previous slide, we would require 2^k rows in the table. So, if k is 64, then we would require of the order of 10^{19} rows in the table.

So, that is infeasible for values of k used in practice. So, hence, we use an alternative approach to map the plaintext to the ciphertext. We use a one-to-one function that simulates a randomly permuted look-up table. So, we don't actually use a look-up table, but we use a one-to-one function that transforms the plaintext in a manner that it looks like a random permutation. This shows an example of such a function.

This is a 64-bit input, which is the plaintext. To transform it into the ciphertext, it is first divided into chunks of 8 bits each. So, this is the first chunk, then this is the second chunk, and so on and so forth. This is the third chunk, etc. Then, each chunk is passed through a look-up table.

The first look-up table is T_1, the second look-up table is T_2, and so on and so forth. The last look-up table is T_8. So, the first chunk is transformed through this look-up table into k bits of the output of the look-up table. Similarly, each chunk is transformed. Then, the outputs of the 8 chunks are combined, and then they are scrambled. That is formulated to produce a 64-bit intermediate output, and that 64-bit output is here.

It is obtained by scrambling the outputs of the 8 look-up tables, and this is the scrambler that is shown. It permits the outputs of the 8 look-up tables. Then n such rounds are performed; that is, this intermediate output is fed back to this register, and then the same transformation is again repeated on it, and similarly, n such rounds are performed to make the output a very complicated function of the input. So, what is the key for this cipher? The key for this cipher is the 8 look-up tables, assuming that the scrambling function is publicly known.

So, a reason for using n rounds instead of using only one round is that we want to make the output a very complicated function of the input. Another reason is that we want a particular bit of this output to depend on all the bits of the input or as many bits of the input as possible. So, if we were to pass the input only once through these functions, then a particular bit of the input would affect only a few bits of the output. But we want a bit of the output to be influenced by a large number of bits of the input. So, for this reason, we pass the input n times through this process to get the ciphertext from the plaintext.

What is the objective in the design of a block cipher? The objective is to take a reasonable-length key, for example, 64 bits long, and generate a one-to-one mapping that looks completely random to someone who doesn't know the key. For a block cipher of k bits, "completely random" means the following. For the first input of k bits, the output is selected from among the 2^k possible outputs uniformly at random. Then, we have $2^k - 1$ remaining possible outputs.

For the next input of k bits, the output is selected uniformly at random from the remaining $2^k - 1$ possible outputs, and so on and so forth. So, this is what is meant by completely random. So, the plaintext is transformed to the ciphertext using a function, one-to-one function. And the one-to-one function looks like a random transformation; that is,

it appears that each possible output is a randomly generated output. It looks like for each input, the corresponding output is randomly generated independently of the other inputs.

What is the advantage of a completely random mapping? If the input were mapped to the output in predictable ways, then it may be possible for the intruder to guess parts of the plaintext by looking at the ciphertext. So, then it would be easy to break the cipher. So, for this reason, we don't map the input to the output in predictable ways. We make the transformation look like a completely random transformation.

An example of a predictable mapping is as follows. Suppose the 10^{th} bit of the output always changes when the 5^{th} bit of the input changes, then the intruder who observes two ciphertext blocks that differ in the 10^{th} bit would be able to find out that the 5^{th} bit of the input may have changed. So, in this way, the intruder gets some information about the plaintext, but we don't want the intruder to get any information about the plaintext whatsoever. Another example of a predictable mapping is the following. If bits 17 to 24 of the output are functions of only bits 57 to 64 of the input, then an intruder who observes two ciphertext blocks with identical bits 17 to 24 could infer that the corresponding plaintext blocks have the same bits in places 57 to 64.

So, the intruder is able to guess some aspects of the plaintext, which we don't want. So, to avoid examples like these, we choose the one-to-one function so that the output looks like a completely random mapping from the input. In particular, even if a single bit in the input changes, then the new output should be a random string that is chosen independently of the old output. So, these are the objectives that we require while choosing the one-to-one function used to map the plaintext to the ciphertext. Recall that, one way to generate a completely random one-to-one mapping is to use a look-up table of size $k \times 2^k$ bits.

But this is infeasible when k = 64 or larger, as we have seen, because there are too many rows in the look-up table. Another possible approach is as follows. We define the mapping in terms of a set of linear equations. Let x_j denote the j^{th} bit of the plaintext block and y_i denote the i^{th} bit of the ciphertext block, where $i, j \in \{1, \ldots, k\}$. Consider the block cipher defined by the following k linear equations. The i^{th} bit of the ciphertext is obtained using this equation, $y_i = \sum_{j=1}^{k} a_{ij} x_j$, where $a_{ij} \in \{0,1\}$.

- Let x_j denote j'th bit of plaintext block and y_i denote i'th bit of ciphertext block, where $i, j \in \{1, \ldots, k\}$
- Consider the block cipher defined by the k linear equations:
  - $y_i = \sum_{j=1}^{k} a_{ij} x_j$, where $a_{ij} \in \{0,1\}$
- Key:
  - $a_{ij}, i, j \in \{1, \ldots, k\}$
- So key is k^2 bits in length

The key for this cipher is the values $a_{ij}, i, j \in \{1, \ldots, k\}$. So, the key is k^2 bits in length. But this cipher has a disadvantage. The disadvantage is that if an attacker is, it is vulnerable to cryptanalysis by an attacker who is aware of the structure of the encryption algorithm. That is, the attacker knows that the encryption algorithm is of this form. It's a set of linear equations.

And the attacker has somehow found out some plaintext-ciphertext pairs. If the attacker gets some values of x_1 to x_k and the corresponding values of y_1 to y_k, then by writing these equations and substituting these values, the attacker may be able to calculate some of the values of a_{ij} for certain values of i, j. That is, the attacker may get some parts of the key. So, sometimes, it is possible for the attacker to guess the plaintext corresponding to a particular ciphertext. In that case, the attacker, as we just discussed, will be able to guess some of the parts of the keys, that is, some values of a_{ij}.

So, hence, we don't use a system of linear equations to transform the plaintext to the ciphertext. Instead, modern block ciphers are product ciphers. We'll discuss what is meant by product ciphers. A product cipher approximates a completely random mapping using a small key, for example, 64 bits long, and it is robust to cryptanalysis. So, this, we now discuss what a product cipher is.

The building blocks of a product cipher are the following. It uses two kinds of transformation. One is a P-box; it implements a permutation of the input, and S-box implements a substitution, that is, a look-up table. But as we'll see, the size of this S box is not k, that is, the block cipher size, but we use a smaller value of the number of bits in the look-up table. This figure shows hardware implementations of a P-box and a S-box.

It's very easy to implement in hardware. This shows a permutation. So, first bit of the input is mapped to the third bit of the output. The second bit is mapped to the fifth bit of the output, and so on and so forth. So, this is a permutation.

And this implements an S-box that is a substitution. So, for example, 000 might be mapped to 110. And this is done by using the decoder in combination with the permutation and then, encoder. So, you can verify that this transformation indeed implements a look-up table. What is the number of bits required to store a k bit P-box in software?

So, the number of bits required is approximately $k \log_2 k$. The reason is that we have to specify the output for k possible inputs. So, for each of the possible k inputs, we have to

specify the output. So, that requires $k \log_2 k$ bits. For example, for k = 64, 384 bits are required.

No. of bits required to store k bit P-box:
- $\approx k \log_2 k$
- e.g., for $k = 64$, 384 bits required

No. of bits required to store m bit S-box:
- $m2^m$

The number of bits required to store an m bit S-box is $m \times 2^m$. It is left as an exercise for you to verify that we require $k \log_2 k$ bits to store a k bit P-box and we require $m \times 2^m$ bits to store an m bit S-box. So, if m is, for example, 64, then the number of bits required to store an m bit S-box is too large. For example, it is 1.18×10^{21}. But if m is small, such as 8, then we require only 2048 bits to store an m bit S-box. So, we can use S-boxes of a small number of bits.

In a product cipher, a series of S-boxes and P-boxes are cascaded to get a product cipher. This shows an example of a product cipher. First we have a P-box, then we have a set of small S-boxes, then we have a P-box, then a set of S-boxes, and then another P-box, and then S-boxes, and then another P-box. So, we have such a cascade of many stages of P-boxes and S-boxes. So, here k = 12, that is the input; the plaintext length is 12 bits, and the ciphertext length is also 12 bits.

These 12 bits of the plaintext are first formatted by the P-box P_1, and then the result is broken into four groups of 3 bits each, and then each of these groups is independently passed through a look-up table. These look-up tables are S_1, S_2, S_3, S_4. These are S-boxes. Then the outputs of the four look-up tables are combined, and they are then passed through this P-box P_2, and so on and so forth. So, this process transforms the plaintext to the ciphertext.

By using sufficient number of stages in a product cipher, the output can be made a highly complicated function of the input. So, this is a product cipher; modern block ciphers use a product cipher. They use such a cascade of P-boxes and S-boxes to transform the plaintext to a ciphertext, to the corresponding ciphertext. So, a product cipher approximates a completely random one-to-one function. Recall that we discussed earlier that the objective in a block cipher is to transform the plaintext to the ciphertext in a way that simulates a completely random one-to-one function.

And a product cipher achieves this objective. In particular, each input bit affects all output bits, so we don't have predictable mappings of the sort that we discussed in the above examples. Also, a product cipher uses a small key, for example, 64 bits in length. The example product cipher that we studied earlier, which is shown in this figure here, is a product cipher. Several block ciphers used in the internet are product ciphers.

Examples are DES, 3DES, AES. These are all product ciphers. So, in summary, we discussed several kinds of symmetric key ciphers. For example, we discussed Caesar cipher, then monoalphabetic cipher, polyalphabetic cipher, and then block cipher. Modern ciphers used in the internet are product ciphers.

And popular examples are DES, 3DES, and AES. Thank you.