Modern Cryptography, Part III

Block ciphers

In classical cryptography, the Playfair cipher is an example of a block cipher. Instead of encrypting each letter individually, the Playfair cipher encrypts them in groups (or blocks) of two letters. In modern cryptography, the idea of a block cipher is to group the bits of the plaintext into blocks and encrypt entire blocks at once, instead of one bit at a time like a stream cipher would do.

Let's look at an example. The cipher in this example is one I've made up to help demonstrate block ciphers. This will be a 4-bit block cipher, which means we will start by grouping the bits of the plaintext into groups of 4 bits. Then we will use the following rules to encrypt each block.

1. First break it into two mini-blocks of two bits each. Apply the substitution rules 00 → 01, 01 → 11, 10 → 00, and 11 → 10 to each.
2. Then rotate all the bits one step to the right, wrapping the last bit around to the front.
3. Finally XOR the block with the key 1101.

Let's suppose our plaintext is 1001101011010111, We start by breaking it into 4-bit blocks: 1001 1010 1101 0111. Then we apply the rules on each block. For the first block, we break it into the miniblocks 10 01. According to the substitution rule, this should become 00 11. Then we rotate all the bits around to get 1001. Finally, we XOR with 1101 to get 0100. This is our first ciphertext block.

We then do the same process to the other three blocks. Here is a table summarizing the results. The final line gives us the ciphertext 0100110100001010.

operation	Block 1	Block 2	Block 3	Block 4
plaintext	1001	1010	1101	0111
after substitution	0011	0000	1011	1110
after rotation	1001	0000	1101	0111
after XOR	0100	1101	0000	1010

Decryption is the reverse of encryption. We would first XOR with the key, then rotate left, and finally apply the reverse substitution. Many real block ciphers operate on a similar principle of combining substitutions and permutations, along with something to mix in the key. Those ciphers use several rounds of these operations as well as much larger block sizes.

How big should the blocks be?

First, block sizes are usually taken to be powers of 2 because sizes that are powers of 2 are more efficient to implement at the CPU level than other sizes.

Next, block ciphers that use small block sizes can be subject to something called a codebook attack, where attackers build up a table of which plaintext blocks correspond to which ciphertext blocks. The block size needs to be large enough to make it so that the table is unmanageably large.

The birthday problem also comes into play. For similar reasons as to why you don't want to reuse the keystream in a stream cipher, it's important that ciphertext blocks don't repeat. A block cipher that uses 32-bit blocks would have repeating blocks becoming likely after $\sqrt{2^{32}} = 2^{16} =$65,536 blocks. This is unacceptably low. If we go with 64-bit blocks, then repeats are likely after $\sqrt{2^{64}} = 2^{32} \approx 4$ billion blocks. This is better, but it is still only a few gigs of data, which can be reached quickly on a fast internet connection. So we would need to use at least 128-bit blocks.

We could go higher, but 256-bit blocks would be too big to fit inside CPU registers. We would have to split them, would cause a considerable slowdown in the encryption process. So 128-bit blocks is what most good block ciphers use.

AES and DES

DES (data encryption standard) is a block cipher developed at IBM in the mid 1970s with help from the NSA. It was the most widely used block cipher up until the early 2000s. It uses a 56-bit key, which was a compromise between the 64-bit key IBM wanted and the 48-bit key the US government wanted. Interestingly, though the NSA weakened the cipher by shortening the key, they strengthened it against a technique called differential cryptanalysis, which was not a publicly known technique in the 1970s.

That 56-bit key was large enough that probably the only organization at the time that could do a brute-force search of all the keys was the US government. However, computing power exploded exponentially in the intervening decades, and in the late 1990s, the Electronic Frontier Foundation built a machine that could complete a brute-force search in a few days' time. It cost about $250,000. Nowadays, a single GPU costing around $1000 can break DES keys in a few weeks' time.

To address this, people starting running DES three times in a row with different keys. This is known as *triple DES* or 3DES. Three encryptions in a row would seem to give a $3 \cdot 56 = 168$ bit key, but because of something called a meet-in-the-middle attack, 3DES only has 112-bits of security. This, however, is still well beyond the limits of brute-force. But 3DES isn't considered a very good cipher anymore. Part of the problem is that running DES three times in a row is slow. Part of the problem is that DES uses a 64-bit block size. That was fine in the 1970s, but as we saw above, it's not ideal anymore.

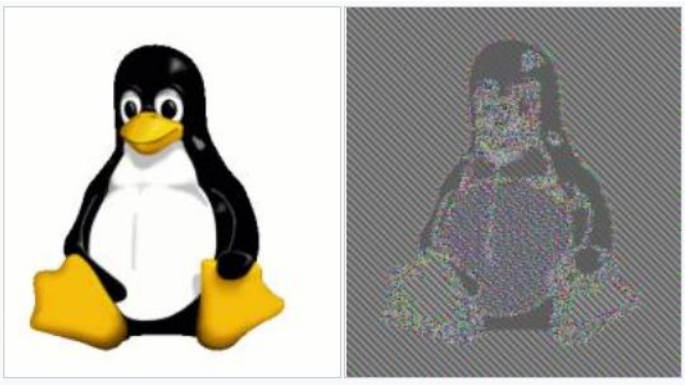
Recognizing the problems with DES, in the late 1990s the US government sponsored a contest to come up with a replacement. Cryptographers from around the world submitted entries and tried to break others' entries. Many of them turned out to be breakable, but there were five good ciphers that were selected as finalists. The winner, called Rijndael, was renamed as AES, the *advanced encryption standard*.

AES is still considered to be the standard today. Despite 20 years of trying, no one has found a viable direct attack on the algorithm. There are indirect attacks called *side-channel attacks* that rely on weaknesses of how the algorithm is programmed or implemented by humans, but almost any cipher can suffer from these. We will talk more about them later.

To give you a rough idea of how AES works internally, it uses what's called a substitution-permutation network. It involves many carefully chosen substitutions and permutations, along with steps to mix in the key. There are many rounds of this. To try to encrypt a single block with AES by hand would probably take you the better portion of a day. AES uses 128-bit blocks. It can be used with keys of size 128, 192, or 256 bits. It was designed with speed in mind, so it is very fast. In fact, modern CPUs have special instructions designed just for AES.

Block cipher modes

You may have noticed with the Playfair cipher or with the simple block cipher above that the same plaintext blocks are encrypted to the same ciphertext blocks. This is a major weakness. The picture below, courtesy of Wikipedia[1], is a famous description of why.



[1] Specifically, the image attribution is to Larry Ewing, lewing@isc.tamu.edu, and The GIMP.

On the left is Tux, the Linux mascot. On the right, he has been encrypted with a block cipher. Notice how the same color regions tend to be encrypted into the same things, leaving the overall structure of the image intact. This is bad.

Using a block cipher directly like this, without any additional processing, is called *electronic code book* or *ECB mode*. It is very insecure. I've heard that the only reason ECB mode was even given a name is so that people can tell you not to use ECB mode.

To be secure, a block cipher needs a secure *mode of operation*. ECB mode, as we've seen, is not secure. Currently, the three most important modes of operation are CBC, CTR, and GCM. We will cover the first two here. The last one requires some high-powered math, so we will just touch on it below.

CBC mode

CBC is short for *cipher block chaining*. The chaining comes from the fact that each block is sort of tied into or chained with the block that follows it. This happens by XOR-ing each plaintext block with the ciphertext from the block before it. CBC mode also requires something called an *initialization vector* (IV) to disguise the first plaintext block. Here is a step-by-step description of the process.

1. Disguise the first plaintext block: $IV \oplus P_1 = P_1'$. Then encrypt P_1' to get C_1.
2. Disguise the second plaintext block: $C_1 \oplus P_2 = P_2'$. Then encrypt P_2' to get C_2.
3. Disguise the third plaintext block: $C_2 \oplus P_3 = P_3'$. Then encrypt P_3' to get C_3.
4. ...

The process continues in the same way for the rest of the blocks. Let's look at an example with a 4-bit block cipher. For simplicity, let's assume the encryption process is simply given by the lookup table below.

$0000 \to 1010$	$0100 \to 0110$	$1000 \to 0001$	$1100 \to 1100$
$0001 \to 1110$	$0101 \to 1011$	$1001 \to 0111$	$1101 \to 0000$
$0010 \to 1000$	$0110 \to 0101$	$1010 \to 0011$	$1110 \to 1001$
$0011 \to 1111$	$0111 \to 0010$	$1011 \to 1101$	$1111 \to 0100$

We'll encrypt the plaintext 1001 1111 1101. And let's go with 1011 for the IV.

CBC mode encryption starts by XOR-ing the IV with the first plaintext block. This gives $1011 \oplus 1001 = 0010$. Then we encrypt this block using the look-up table to get 1000. This the first ciphertext block.

We then XOR this with the second plaintext block to get $1000 \oplus 1111 = 0111$. We encrypt 0111 using the table to get 0010. This is the second ciphertext block.

Then we XOR this with the third plaintext block to get $0010 \oplus 1101 = 1111$. We encrypt 1111 using the table to get 0100.

The end result is then 1000 0010 0100.

CBC mode is reasonably secure, and has been used in TLS for a long time, though there is a class of attacks called *padding oracle attacks* that have made it less widely used.

CTR mode

CTR mode is short for *counter mode*. It essentially turns a block cipher into a stream cipher. CTR mode uses a counter variable along with a *nonce*, which is a *number used once*. The counter is incremented by 1 for each block of the plaintext. The nonce and counter are concatenated together and that value is run through the block cipher. The result is then XORed with the corresponding plaintext block to get the ciphertext block.

Let's look at an example with 8 bit blocks. We will use a very simple block cipher that encrypts blocks by flipping all their bits and then reversing the block. For instance, 10010000 would be flipped into 01101111 and then reversed into 11110110.

With CTR mode, the nonce and the counter combined should come out to the block size. For this example, let's use a 3-bit nonce 101 and a 5-bit counter. The counter will count up in binary starting at 00000. It will go 00000, 00001, 00010, 00011, 00100, 00101, ... We will concatenate these to the end of the nonce.

Let's say our plaintext is 00110101 00001111 01010101 11011010. Here is how the encryption process will go.

plaintext	00110101	00001111	01010101	11011010
nonce+counter	10100000	10100001	10100010	10100011
encrypted nonce+counter	11111010	01111010	10111010	00111010
plaintext ⊕ previous row	11001111	01110101	11101111	11100000

The final ciphertext is the last row, which is 11001111 01110101 11101111 11100000.

Notice how we XOR the first row with the third row to get the ciphertext. This is just like a stream cipher, where the third row is acting like a stream cipher's keystream. This is why it's said that CTR mode turns a block cipher into a stream cipher.

The nonce should be a random number that is not reused. If it is, then since CTR mode is just like a stream cipher, we would end up reusing a keystream. This, as we know, is bad news. Also, the length of the counter should be chosen so that it doesn't wrap back around. Our 5-bit counter in the example above would repeat after 32 blocks, which would also cause keystream reuse.

GCM mode

GCM, *Galois counter mode*, is a variation on ordinary counter mode. It uses some topics from abstract algebra (an upper-level undergraduate math class) along with ideas from CTR mode. The result has the nice benefit that it combines encryption and *authentication*. Not only is the plaintext encrypted, but GCM gives us a way to tell if the ciphertext was modified in transit by a third party. We'll cover authentication in greater detail when we get to hash functions.

Of the three modes, GCM, is currently the most widely used.