CS381-Cryptography

Lecture 5: Block Ciphers

February 8, 2017

1 Overview

1.1 What is a Block Cipher?

Most (not all) modern symmetric encryption is built on *block ciphers*, which are algorithms for encrypting fixed-length blocks of data.

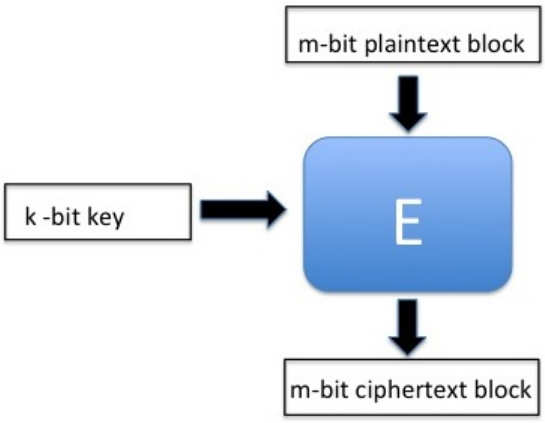

Figure 1: *Block cipher encryption of a single block of data*

Formally, a block cipher is hardly different from the cryptosystems we have already been studying: We have a key space $\mathcal{K} = \{0,1\}^\ell$, a message space $\mathcal{M} = \{0,1\}^n$, and encryption and decryption functions $E, D : \mathcal{K} \times \mathcal{M} \to \mathcal{M}$. What is different is that the block size n is close to the key size ℓ (contrast stream ciphers,

where the message size is typically much larger than the key size). The security requirement for block ciphers is different as well:

If you choose a key $k \in \mathcal{K}$, then the function

$$E_k : m \mapsto E_k(m)$$

is a permutation of $\{0,1\}^n$. The idea is that if you select k at random, this should be for all practical purposes indistinguishable from a randomly-selected permutation of $\{0,1\}^n$. (A randomly-selected permutation is like using the substitution cipher with a random permutation of the alphabet, although here the alphabet has 2^n letters rather than 26.) We do not have enough time or space in the universe to generate or specify a random permutation of $\{0,1\}^n$ for, say, $n = 128$, which is a typical value. The trick is to use a short key to generate a permutation that 'looks' random. Typically, we will use a single key to encrypt multiple blocks of plaintext.

Example. Here is an example of a bad block cipher, our old friend the one-time pad: We set $\mathcal{K} = \mathcal{M} = \{0,1\}^{128}$ and $E(k,m) = k \oplus m$. If we use a single key k to encrypt two blocks m_1 and m_2, giving ciphertext blocks c_1, c_2, we will always have

$$m_1 \oplus m_2 = c_1 \oplus c_2.$$

The probability of such behavior for a random permutation of $\{0,1\}^{128}$ is astronomically small, so we can easily distinguish the E_k of this cryptosystem from randomly selected permutations. (Note that to observe this special behavior, we had to encrypt two plaintexts, something we never do with stream ciphers of the one-time pad.)

1.2 Block Ciphers in Wide Use

We will concentrate on two (or three, depending on how you count it) block ciphers in wide use. DES, the Data Encryption Standard, adopted as a government standard for encryption of non-classified information, has key size 56 bits and block size 64 bits. 3DES, which is built on DES, has key size 168 bits and block size 64 bits. AES, the Advanced Enryption Standard, originally called the Rijndael Cipher, was adopted in 2000 as the replacement for DES. It has 128-bit blocks. It can be used with several different key sizes, but the smallest is $\mathcal{K} = \{0,1\}^{128}$.

2 Block Ciphers as Black Boxes

Here we discuss properties that are independent of the internal structure of the block cipher. We will assume that we have an 'ideal' block cipher: in other words, we will pretend that our key is a permutation of $\{0,1\}^n$ selected uniformly at random from all $(2^n)!$ such permutations.

2.1 Security Against Brute-force Attack

2.1.1 Attack with two known plaintexts

If Eve intercepts a block obtained by encrypting with a block cipher (for example, a 16-byte AES block) she really can't decipher it, even with exhaustive search, if she knows nothing about the distribution of plaintext messages—we are no longer thinking of these things as being normal English.

What we'll show here is that if Eve knows just a few plaintext-ciphertext pairs, then exhaustive search is effective.

First imagine the following experiment. Suppose you have r randomly-selected permutations $\pi_1, \ldots, \pi_r$ of $\{1, \ldots, N\}$. Suppose you know a value of π_1, say $\pi_1(i) = j$. What is the probability that $\pi_k(i) = j$ for one of the other permutations π_k? We can obtain an upper bound on this probability by

$$\begin{aligned} \texttt{Prob}[\bigvee_{k=2}^{r} \pi_k(i) = j] &\leq \sum_{k=2}^{r} \texttt{Prob}[\pi_k(i) = j] \\ &= \frac{r-1}{n}. \end{aligned}$$

Suppose instead that we know *two* values of π_1, say $\pi_1(i_1) = j_1$, $\pi_1(i_2) = j_2$. What is the probability that one of the other permutations π_k agrees at these two values?

$$\begin{aligned} \texttt{Prob}[\bigvee_{k=2}^{r} (\pi_k(i_1) = j_1) \wedge (\pi_k(i_2) = j_2)] &\leq \sum_{k=2}^{r} \texttt{Prob}[(\pi_k(i_1) = j_1) \wedge (\pi_k(i_2) = j_2)] \\ &= \frac{r-1}{n(n-1)}. \end{aligned}$$

What does this say about exhaustive-search attacks? For DES, if we know a single plaintext-ciphertext pair of blocks we can encrypt the plaintext under all

possible keys until we find a block that matches the ciphertext. The probability that this can happen with more than one key is no more than

$$\frac{(r-1)}{n} \approx \frac{2^{56}}{2^{64}} = 2^{-8} = \frac{1}{256},$$

so this attack is likely to reveal the key. If we have two plaintext-ciphertext pairs, and search for a key that encrypts both plaintexts correctly, then the probability that more than one key does this is no more than

$$\frac{r}{n(n-1)} \approx \frac{2^{56}}{2^{128}} = 2^{-72},$$

and for AES it is no more than

$$\frac{2^{128}}{2^{256}} = 2^{-128}.$$

So it is almost certain that there is a unique key that generates the pair of blocks.

This attack requires 2^{k+1} encryptions in the worst case, and 2^k encryptions on average. For 128-bit keys it is really impossible: we have only a negligible probability of finding the key as a result of any search that we could carry out in practice.

The 56-bit key for DES, on the other hand, is problematic. This weakness was pointed out when the standard was first proposed in the 1970's. The attack was successfully carried out in 1997, first by distributed computation by many computers working over the Internet, and then by specialized hardware that could test many keys in parallel. (The hardware was able to perform something like 80 billion encryptions per second!)

2.1.2 Multiple Encryption for Increased Key Size: Meet-in-the-middle attack

One might try to remedy the weak key length in DES by encrypting twice, using two different keys:

$$E'_{(k_1,k_2)}(m) = E_{k_2}(E_{k_1}(m)).$$

The hope is that in doing so we will have effectively doubled the key length, and thus squared the size of the key space. For DES this means a 112-bit key, which should be secure against brute-force attack.

You might well wonder at this point whether composing two short keys in this way is not equivalent to using a single short key, *i.e.,* is there a key k_3 such that

$$E(k_3, m) = E(k_1, E(k_2, m))$$

for all blocks m? It is known that this does not occur in DES.

However, this double-encryption approach is vulnerable to a 'meet-in-the-middle attack', which renders it not much more secure than using a single key, provided one has a great deal of memory available. The idea is simple—for concreteness, we will assume a key size of $k = 56$ bits and a block size of $m = 64$ bits, which are the DES parameters.

Suppose we know two plaintext-ciphertext pairs $(m_1, c_1), (m_2, c_2)$, where

$$c_1 = E(k_2, E(k_1, m_1)), c_2 = E(k_2, E(k_1, m_2)).$$

The attack begins by encrypting the pair of blocks (m_1, m_2) under all 2^{56} keys, storing the resulting ciphertext pairs in a table, and sorting the table. Sorting a list of size N requires time proportional to $N \cdot \log_2 N$, so in this case we require 56×2^{56} steps for the sorting in addition to the 2×2^{56} encryptions of blocks. For the second step, we decrypt (c_1, c_2) under all 2^{56} keys and search for the resulting pair in the table. This requires $2 \cdot 2^{56}$ decryptions of blocks, plus 2^{56} searches of a sorted table, each of which uses about 56 probes. We know that there is at least one pair of keys (k_1, k_2) that results in a match. How likely is it that we will find a second match?

Our table contains no more than 2^{56} pairs of blocks. The probability that a randomly generated pair of blocks occurs in the table is $2^{56}/2^{128} = 2^{-72}$. Thus the probability that we will find more than one match in the decryption phase is no more than

$$2^{56}/2^{72} = 2^{-16} = 1/65536.$$

It is thus highly likely that the two known-plaintext pairs reveal the pair of keys that was used. In the unlikely event that we find more than one pair of keys that works, a third known-plaintext pair is virtually certain to resolve the issue.

The point of this attack is that it is only a few orders of magnitude more costly in terms of time that using a single 56-bit key, and thus the encryption method is far weaker than what one expects with a 112-bit key. (Although it does require a massive table.)

Triple encryption with DES (3DES) is believed to be secure, and was adopted as a standard in the late 1990's.

2.2 Modes of Operation

You have a block cipher that encrypts a single block of data. How do you use it to encrypt multiple blocks?

2.2.1 The obvious (bad) answer: ECB mode

ECB stands for 'electronic code book'. It is the obvious thing, just encrypt each block independently. You've already seen this with the monoalphabetic substitution cipher. The key is a permutation of the 26 letters, a block is a single letter, and encryption is applied block by block. Much of the weakness follows from the fact that repeated blocks of plaintext show up as repeated blocks of ciphertext.

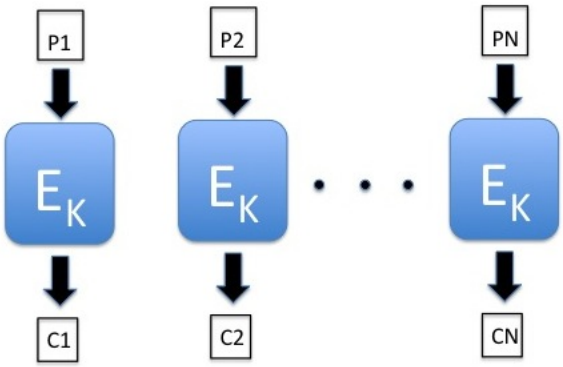

Figure 2: *ECB encryption of N blocks of data*

The same weakness is present in ECB with larger block sizes. If two blocks of ciphertext are identical, then the corresponding blocks of plaintext are also identical. Thus the ciphertext leaks information about the plaintext at the cost of very little computational effort.

You might think that in spite of this, a large block size would make repeated blocks in plaintext very rare, and that in any case, an occasional repeated block in ciphertext would not give away very much useful information. The following demonstration shows just how wrong that can be: The image on the left of Figure 3 was saved in a bit-mapped file format, so that apart from the short file header, every byte of the file is associated with one image pixel, with three bytes for each pixel. The file contents were encrypted by AES in ECB mode, and the original header restored so that the encrypted file could be displayed as a bit-mapped

image.

Figure 3: *ECB encryption of a bit-mapped image. Large regions of the image with a single color lead to a large number of repeated blocks.*

Moral: Don't use ECB mode.

2.2.2 CBC Mode

CBC stands for *cipherblock chaining*. The ciphertext block generated in one step is mixed with the plaintext block of the next step prior to encryption. This leaves the problem of what to mix the first plaintext block with, and for this reason a special block called an *initialization vector* (IV) must be supplied. So we have (see Figure 4)

$$\begin{aligned} c_1 &= E_k(IV \oplus m_1) \\ c_{i+1} &= E_k(c_i \oplus m_{i+1}), \end{aligned}$$

for $i \geq 1$.

Figure 5 shows the image experiment, this time conducted in CBC mode.

To decrypt, Bob computes

$$\begin{aligned} m_1 &= IV \oplus D_k(c_1) \\ m_{i+1} &= c_i \oplus D_K(c_{i+1}), \end{aligned}$$

Bob must have IV available to obtain the first plaintext block, so the *IV must be sent unencrypted.* One consequence of this is that the ciphertext is one block longer than the plaintext message.

You might wonder why we bother with the IV at all, and not simply send $c_1 = E_K(m_1)$ as the first ciphertext block. The reason is that if we have several

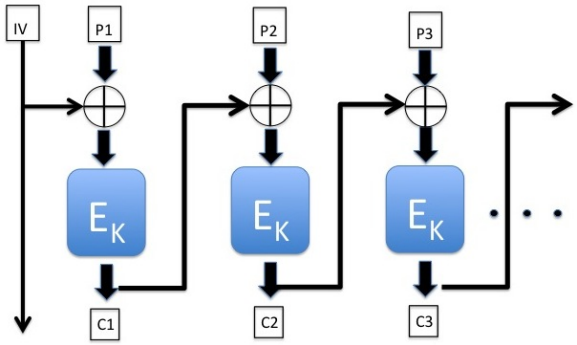

Figure 4: *CBC encryption of multiple blocks of data*

Figure 5: *CBC encryption of a bit-mapped image. The chaining process alters the input to each encryption block, and the resulting ciphertext looks completely random.*

long messages encrypted with the same key, then a repetition of the first blocks in two of the messages will show up as a repetition of the first blocks in the received ciphertexts.

For the same reason, one should not use the same IV for two different messages. (For a related, but somewhat subtler reason, it should not be possible to predict the IV of the next message.) A sound practice is to choose a random IV for each message.

CBC has a nice error-recovery feature: Although changing one bit of a plaintext block completely changes all subsequent ciphertext blocks, a change of one bit of a ciphertext block because of a transmission error will only alter one bit in the next decrypted plaintext block, and have no effect on the subsequent ones.

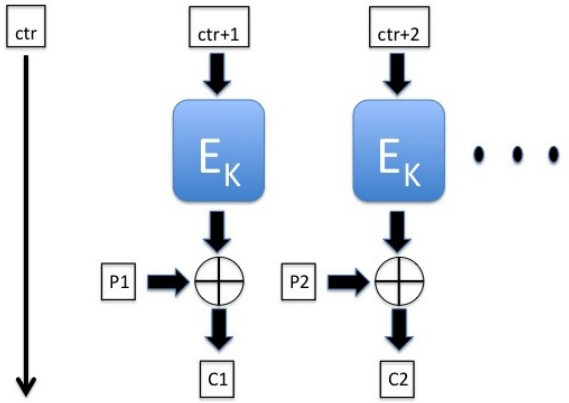

Figure 6: *Encryption of multiple blocks in CTR mode. Just as with the one-time pad, the plaintext is XORed with a keystream. In this case the keystream is generated using the block cipher applied to successive values of a counter block.*

2.2.3 CTR Mode

CTR stands for *counter.* An initial value ctr is chosen for a counter block. If we have N plaintext blocks to send, we encrypt the blocks $\mathrm{ctr} + i$ for $i = 1, \ldots, N$, and XOR the results with the subsequent blocks of plaintext. (Figure 6). Note that we do not use the block cipher to directly encrypt the plaintext blocks; instead this is a stream cipher in which we use the block cipher to generate the keystream. As is the case with stream ciphers, decryption is the same as encryption, and you can not use the same keystream twice. For this reason, counter values should never repeat.

If the encryption function were truly random, then this method would be as secure as the one-time pad, provided that we never repeat a value of the counter fed to E_k. The probability that we repeat a value of the counter with two messages (or any small number of messages) is quite small, unless the messages are huge. For example, suppose we have two messages that are both one million blocks long, and that we use a block cipher with a 128-bit block size. We randomly initialize `ctr` for each of these messages. We will get a repeated value of the counter only if the initial values of `ctr` for the two messages are within two million of one another. The probability of this happening is no more than

$$\frac{2 \cdot 10^6}{2^{128}} < 3 \cdot 10^{-31}.$$

Observe that the large block size is crucial here.

An advantage of CTR mode is that it is very fast and can be executed in parallel.

2.3 Padding oracle attack on CBC

ECB and CBC cannot be used directly if the number of bits in the plaintext message is not an integer multiple of the block size. If you need to encrypt such a message, you must first add padding bits to adjust the block size before encrypting. After decryption, the padding must be removed to obtain the original plaintext. This means that the padding must be identified unambiguously: there should be no question about which bytes belong to the pad and which to the original plaintext.

There is a very simple and widely used padding standard called PKCS7. Let ℓ be the length, in bytes, of the message, and let L be the smallest multiple of the block size (in bytes) strictly larger than ℓ. For instance, if $\ell = 55$ and the block size is 8 bytes, then $L = 56$, but if the block size is 16 bytes, then $L = 64$. The pad consists of $L - \ell$ bytes, each of which has the value $L - \ell$. For example, if the block size is 8 bytes, then the padded version of Boston College is

```
Boston College\x02\x02
```

because the original text is 14 bytes long, so we pad with two bytes, each having value 2. Note that even messages whose length is a multiple of the block size get padded: The padded version of `Chestnut Hill MA` with 8-byte blocks is

```
Chestnut Hill MA\x08\x08\x08\x08\x08\x08\x08\x08
```

Here the padding takes up an entire block.

A server receiving encrypted messages removes the padding after decryption. What does it do if the padding is incorrect? It might indicate this in an error message. The server thus acts as a *padding oracle*. If you possess a ciphertext, then by repeatedly modifying it and submitting it to the oracle, it is possible to decrypt it completely! The number of queries to the oracle is relatively modest (about 256 times the number of bytes in the message).

Here is how the attack works. Let us say we have a three-block message that is the DES encryption of a two-block plaintext (the first block of the message is the IV). So we have

$$IV||c_1||c_2,$$

and each of these blocks is 8-bits long. Presumably this was padded correctly, but we can verify this by submitting it to the oracle. Let $m_1||m_2$ denote the plaintext, which we do not know. The idea is that by tweaking the *first* ciphertext block c_1, we induce changes to the *second* plaintext block m_2. More precisely, changing c_1 can always be accomplished by XORing the byte with some other block b, and the effect on the plaintext block m_2 will be to change it to $m_2 \oplus b$.

Now let's change the highest-order byte of c_1, giving c_1'. When decrypted, this will change the highest-order byte of m_2. What happens when we submit $IV||c_1'||c_2$ to the padding oracle?

If the padding oracle returns an error, then the high-order byte was part of the padding, which means that the pad value is 8, repeated in all 8 bytes of the plaintext. If the padding oracle does not return an error, then the length of the pad is less than 8. We restore the changed byte of c_1 and change the next-highest byte, and repeat the experiment. Eventually we will locate the highest-order byte of m_2 that is part of the padding. This tells us both the length of the padding and the value of the byte.

Let's suppose, by way of example, that the padding has length 3, so that the successive bytes of m_2 are

```
? ? ? ? ? 03 03 03
```

We first XOR the ciphertext block c_1 with 00 00 00 00 00 07 07 07. Since $03 \oplus 07 = 04$, when we submit the result for decryption, the last block will have the form

```
? ? ? ? ? 04 04 04
```

The high-order five bytes are ones from the original plaintext. If the padding oracle does not report an error, then we know that the rightmost unknown byte is 04, and that m_2 is `? ? ? ? 04 03 03 03`. If, as is more likely, the padding oracle does report an error, then we XOR c_1 with, in turn, `00 00 00 00 xx 07 07 07`, where `xx` takes on all 255 values from `01` to `ff`. Exactly one of these values will cause the last block of decrypted plaintext to have the form `00 00 00 00 04 04 04 04` and not produce an error message from the padding oracle. Let us suppose that `xx` $=$ `3f` is this value. Then the rightmost unknown byte of m_2 is `3f` $\oplus$ `04` $=$ `3b`. Thus we have recovered one byte of the original plaintext. We now know m_2 has the form

```
? ? ? ? 3b 04 04 04
```

We can continue in this manner, decrypting the remaining three bytes of m_2.

We now proceed to decrypt m_1. To do this, we need to tweak the IV to induce corresponding changes in m_1. This time, m_1 is probably not correctly padded (unless its least significant byte happens to be 1). We thus XOR the least significant byte of the IV with all the values from `01` to `ff` and see which gives us correct padding. A small problem results from the fact that there might be more than one answer to this question. If our XORing forces the lowest-order byte of the plaintext to be `01`, the padding will be correct. But let's imagine that the next higher byte of m_1 is `02`, in which case there will now be two different values that give correct padding. However, once we find a modification that causes m_1 to be correctly padded, we can use the method above to find the length of the pad, and thus figure out which of the possible solutions changes the lowest byte of m_1 to `01`. This will tell us the low-order byte of m_1. We now decrypt the rest of m_1 exactly as we did above with m_2.

This attack was discovered by Serge Vaudenay in 2002. It was subsequently found that there some real systems could be used in this way as padding oracles, and were vulnerable to the attack.

3 Internal Structure of Block Ciphers (especially AES)

Good block cipher design is an extremely difficult task. Both AES and triple-DES have been subject to intense scrutiny for many years, and no major security weakness has been discovered. If you need a block cipher, use one of these—don't try to design your own.

Still, it's worth talking about some of the design principles behind these block ciphers. Block ciphers typically have a *round structure:* In each round, the key, or a portion of the key, is used to derive a *round key.* Each round further alters the plaintext. Each individual round has a simple, easy-to-understand structure, and permits a rapid implementation. But the accumulated effect of all the rounds is supposed to make it effectively impossible to recover information about the plaintext block from the ciphertext block.

3.1 Substitution-Permutation Networks

What's inside a round? One general model (essentially followed in AES, but not in DES), is called a *substitution-permutation network.* This is illustrated in Figure 8. The input block of the round is first XORed with the round key. The resulting

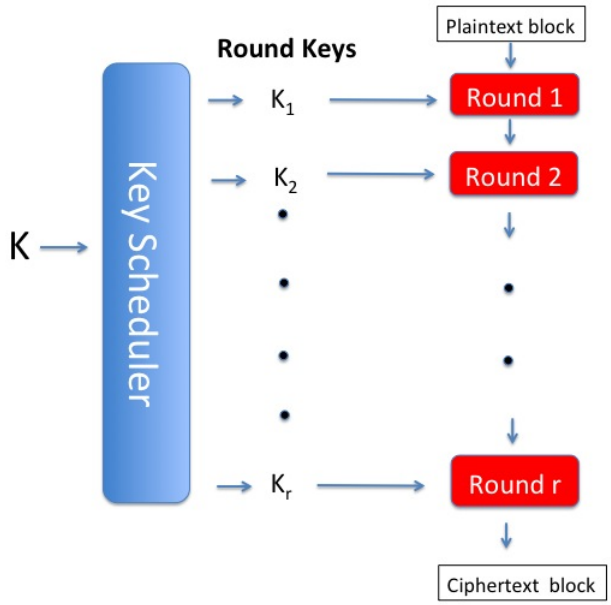

Figure 7: *Round structure of a block cipher.*

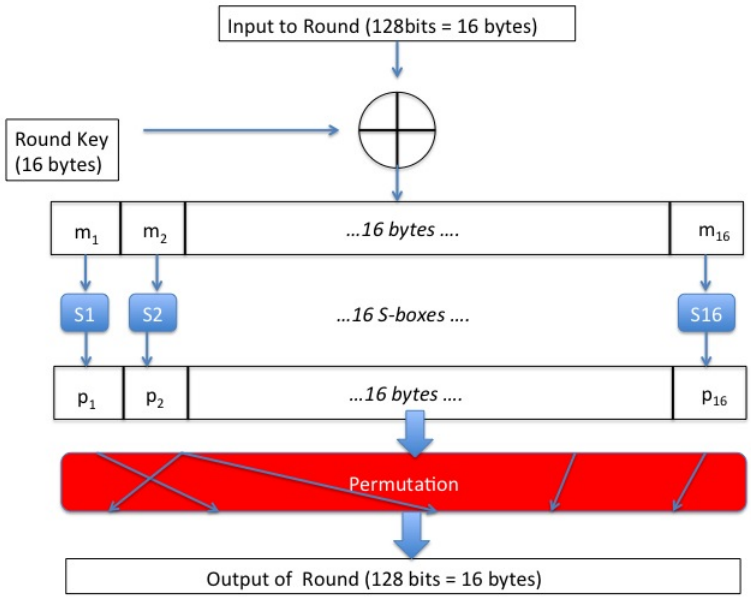

Figure 8: *One round of a substitution-permutation network. The S-boxes are lookup tables that provide a permutation of $\{0,1\}^8$. The mixing permutation permutes the bits of a 128-bit block.*

block (16 bytes in the example) is partitioned into 1-byte sub-blocks. The byte in each sub-block is replaced by a different byte, by means of a lookup table called an *S-box*. The bits of the outputs of the S-box are then permuted and the resulting output becomes the input to the next round. We call these three phases of a round AddKey, Sub, and Mix.

If we did not have the S-boxes, then regardless of the number of rounds, the ciphertext C would be $\pi(P) \oplus K'$, where K' is a block derived from the key, and $\pi(P)$ is some known permutation of the bits of P. This means that if we encrypted two different plaintext blocks P_1, P_2 to produce ciphertexts C_1, C_2, we would have $C_1 \oplus C_2 = \pi(P_1) \oplus \pi(P_2)$, revealing information about the plaintext with no knowledge of the key.

For the same reason, the S-boxes must be a *nonlinear* functions of the input bytes.

The S-boxes are designed so that changing any bit of the input byte changes at least two bits of the output byte. The mixing permutation is designed so that the

output bits of a single S-box are sent to *different* sub-blocks. The effect of these two operations is the following: If we change one bit of plaintext, it will result in at least two bits difference in the output of one S-box in the first round. The mixing permutation will send these two bits to different S-boxes in the subsequent round, resulting in 4 bits of difference at the end of round 2, then 8 bits of difference at the end of round 3, etc. Of course, it cannot keep doubling like that, and there will be some collisions along the way, where output bits of two different subblocks are sent to the same subblock. But the net effect of all of this should be that changing one bit of a plaintext block will on average change about *half* the bits of the resulting ciphertext block. This is called the *avalanche effect.* As a result, local alterations to a block are diffused throughout the block. This also shows that the cipher requires a relatively large number of rounds to achieve this kind of thorough mixing.

You can see from even this brief description that the design of a round requires some intricate engineering, and is not a question of choosing something really confusing-looking at random.

AES is a variant of the substitution-permutation network idea. In AES, all the S-boxes are the same. The mixing permutation step in each round is divided into two phases: In the first phase (ShiftRows), the 16-byte state is represented as a 4×4 array of bytes, and each row of this matrix is cyclically shifted. In the second phase (MixColumns), the state is represented as a 32×4 matrix of bits, which is multiplied on the left by a particular 32×32 matrix of bits, with addition being done modulo 2. This is not precisely a permutation of the bits, so in this respect, the description differs from that of the generic substitution-permutation network. There are ten such rounds, with the last round being slightly different from the earlier ones.

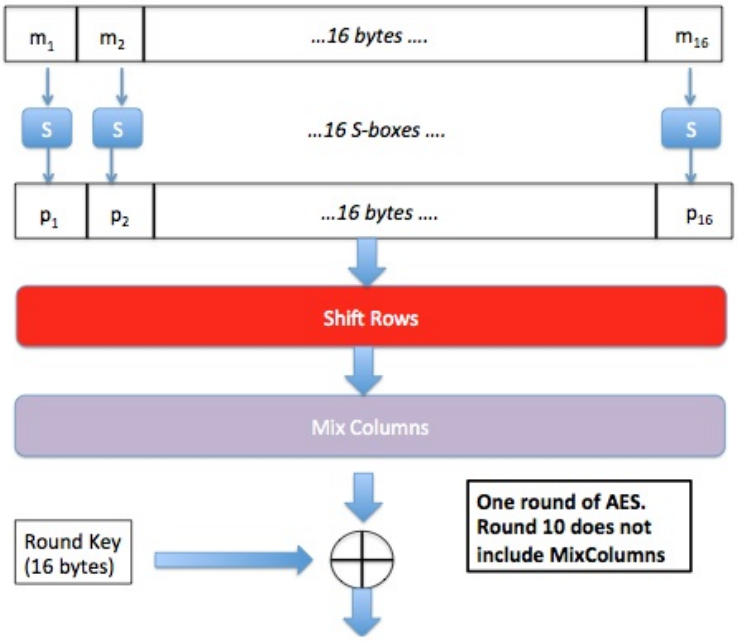

Figure 9: *One round of AES-128. All the S-boxes are the same, and the mixing permutation is replaced by a permutation of the bits followed by a linear transformation.*