Reference: Computer Security by Stallings and Brown, Chapter 20

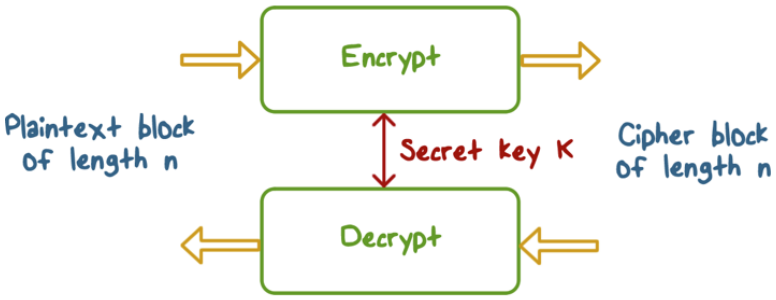
Symmetric Encryption
Lesson Introduction

- Block cipher primitives
- DES
- AES
- Encrypting large message
- Message integrity

Symmetric encryption algorithms are typically block ciphers that take thick size input. In this lesson, we will first discuss the basic primitives of block ciphers, and then we will study the DES and AES algorithms. We will also discuss how to encrypt a large message that has multiple blocks of input data, and how to protect message integrity.

Most symmetric encryption schemes are called block ciphers because they take a fixed length plain text block as input. We will discuss later how these schemes can be used to encrypt input text arbitrary length.

Block Cipher Scheme



Here is a very high level view of block ciphers. A plain text block of fixed length, say n, is encrypted using secret key, k, to produce a cipher attacks block of same length, n. The same secret key, k, is used to decrypt the cipher attack's block into the plain text block.

Block Cipher Primitives

Confusion:

- An encryption operation where the relationship between the key and ciphertext is obscured
- Achieved with **substitution**

The goal of encryption is to transform a plaintext into an intelligible form. That is, given that the attacker can obtain the ciphertext, we don't want the attacker to be able to learn about the plaintext. In order to accomplish this, we apply two principles.

The first is called confusion, it is a way to obscure the relationship between a key and a ciphertext. That is, although

the attacker can obtain the ciphertext, the attacker will not be able to find out the key. This is typically achieved with substitution. For example, in the generalized substitution ciphers that we have discussed, each letter can be mapped to any other letter. Confusion or substitution alone is not sufficient. For example, even when a letter can be mapped to any other letter, the attacker can use statistical analysis of data frequencies to break the scheme.

Block Cipher Primitives

Diffusion:

- An encryption operation where the influence of one plaintext bit is spread over many ciphertext bits with the goal of hiding statistical properties of the plaintext
  - Achieved with **permutation**

Therefore, the second principle is diffusion. This means that the influence of one plaintext bit is spread over many bits in a ciphertext and the goal is to hide the statistical properties of the plaintext. For example, instead of mapping an English letter to another English letter, we can map a letter to parts of many a bit letters. And if you do that, then the frequency distribution of English letters will not be very useful in cryptanalysis. We can achieve diffusion with permutations.

Block Cipher Primitives

- Both confusion and diffusion by themselves **cannot provide (strong enough) security**
- **Round:** combination of substitution and permutation, and do so often enough so that a bit change can affect every output bit

Further, we need this combination to effect every bit of the ciphertext. Therefore, typically, a block cipher has multiple rounds where each round combines substitution and permutation. That is, the first round affects some parts of the ciphertext and the next round further propagates these effects into other parts of the ciphertext. Eventually, all bits of ciphertext are affected.

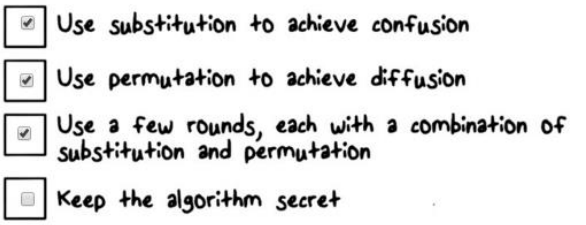
Block Cipher Quiz

Select all correct answers to complete that statement.

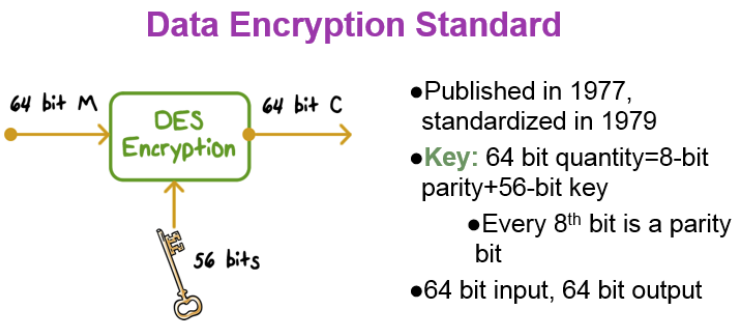
A block cipher should...

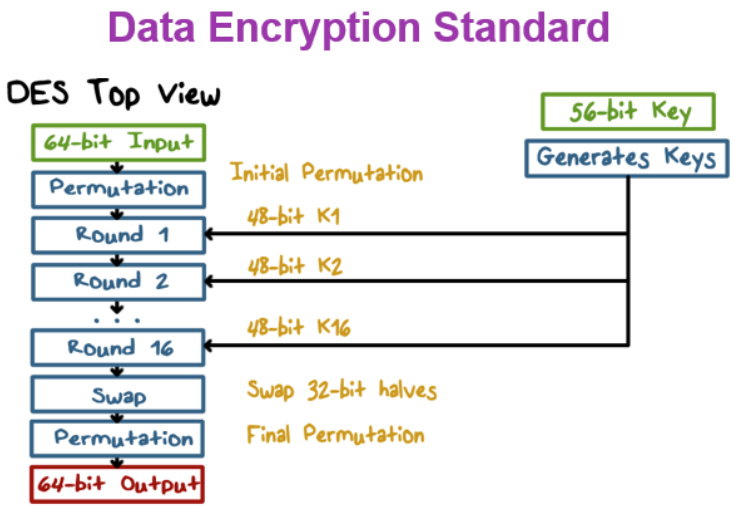
- ☐ Use substitution to achieve confusion
- ☐ Use permutation to achieve diffusion
- ☐ Use a few rounds, each with a combination of substitution and permutation
- ☐ Keep the algorithm secret

Now let's do a quiz on the high level concepts of block ciphers. Select all correct answers to complete this statement. A block cipher should first use substitution to achieve confusion. Second, a block cipher should use permutation to achieve diffusion. Third, a block cipher should use a few rounds, each with a combination of substitution and permutation. Fourth, a block cipher should keep the algorithm secret.



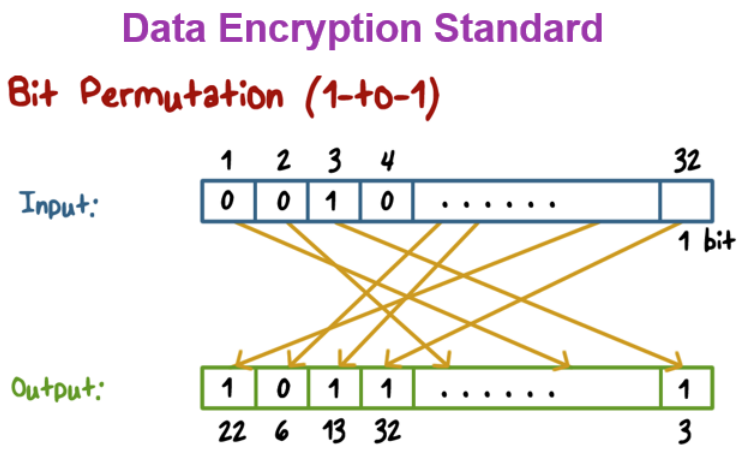
First, a block cipher should use substitution to achieve confusion. This is correct. Second, a block cipher should use permutation to achieve diffusion. This is correct. Third, a block cipher should use a few rounds, each with a combination of substitution and permutation. This is correct. These are the three high level properties of block ciphers that we have just discussed. Fourth, a block cipher should keep the algorithm secret. This is false. Because we should never keep the algorithm secret. We should only keep the key secret. Many researchers have analyzed a public or standard algorithm and improved its security. Therefore, a public or standard algorithm can be more secure than a secret algorithm. Therefore, we should use a public algorithm.



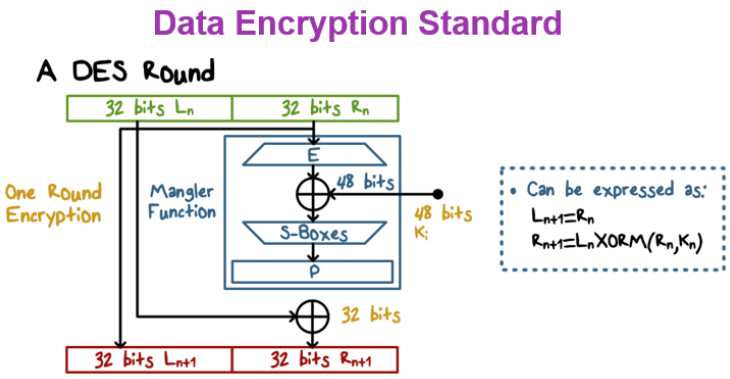


A widely used symmetric encryption scheme is based on the data encryption standard, or DES. It was published in 1977 and standardized in 1979. In DES the key is 64 bit, which is 8 bites but for each bite, there's one parity bit. Therefore, the actual value in the key is only 56-bit. And the oppo cipher text is again a 64-bit block.

Here's a high level view of DES. There are 16 rounds of operations. From the origin of 56-bit key, 16 subkeys are generated. One for each round. The process of decryption with DES, is essentially the same as the encryption process. It works as follows. Use the cipher text as input to this. But then, for sub keys, they use in reverse order. That is use key 16 at the first round of decryption. Use key 15 in the second round of decryption. And so and so forth. That is for decryption, we run the same algorithm. But only the keys in reverse order.



We call that diffusion is one of the principles in encryption and it is typically achieved through permutation. In DES, permutation works by changing the positions of the bits.



Recall that DES has 16 rounds of operations. Each DES round has the same operations and uses a different per round key. Each DES round takes as input the server text produced by the previous round. And outputs text for the next round. The input is divided into the left half and the right half. The output left half is just the right half of the input. The right half of the output is a result of the left half of the input and the output of the function. The mangler function takes this input, the 32 bit input right half, expands it to 48 bit, then XORs with the 48 bit program key. Then use the s-boxes to substitute the 48 bit value into a 32 bit value. We can also use algebra to represent your in a DES round. That is, the left half of output is the right half of the input. And the right half of the output is the result of XOR'ing the left half of input and the result of a mangler function, which takes as input the right half of the input and the pre-round [??] key.

Decryption

- Apply the same operations key sequence in reverse:
  - Round 1 of decryption uses key of the last round in encryption
- Each round:
  - Input: $R_{n+1}|L_{n+1}$
    - Due to the swap operation at the end of encryption
  - Output: $R_n|L_n$
- The swap operation at the end will produce the correct result: L|R

TheDES algorithm has the so-called five style structure. That is, encryption and decryption differ only in key schedule. That is in decryption, we apply the same operations as in encryption, but only with the key sequence in reverse. That is, round one of decryption uses the key of the last round of encryption. And so on and so forth. We can take a closer look to verify that decryption really works by applying the same operations as an encryption, and only the key sequence in reverse. In particular, we only need to look at the decryption operations in the last round.

The input to each round, for example, the input to the first round of decryption is actually R and N in reverse order, meaning the right half is on the left, and the left half is on the right. This is due to the swap operation at the end of the encryption process. For example, the input to the first round of decryption would be R16 and L16, because encryption has 16 rounds. And here, because we are using the key sequence in reverse in decryption, then the key using the first round of decryption is the key using the 16th round of encryption.

Recall that we can use algebra to represent the DES operations. If you use those algebra expressions, you will be able to verify that given this input in the decryption process you should be able to produce this output. I'll leave these as a exercise for you to do at home. Therefore, after 16 rounds of decryption process, the output would be R and L in plaintext. In the final swap operation at the end of the process would produce the correct plain text result L and R in the correct order. And this is how decryption in this works.

XOR Quiz

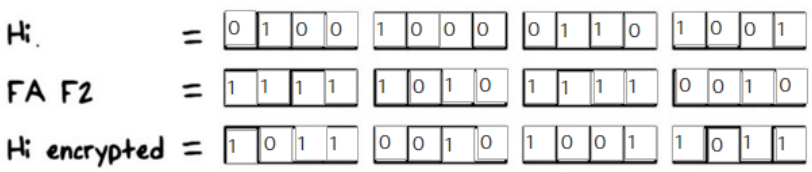
Use the XOR function and the given key to encrypt the word "Hi".

key = FA F2

Hi	=				
FA F2	=				
Hi encrypted	=				

XOR is a basic operation in cartography. For example, in a DES round, the right half of the input is extended to 48-bit and then XOR with the per-rounnd key. So let's do a quiz to review the XOR operations. Here the key is FA F2 in hex, and the plain text is Hi.

We first convert the characters in the plain text into their ASCII codes. So 'H' is 0x48, 'i' is 0x69. So this is in hex, and in binary:



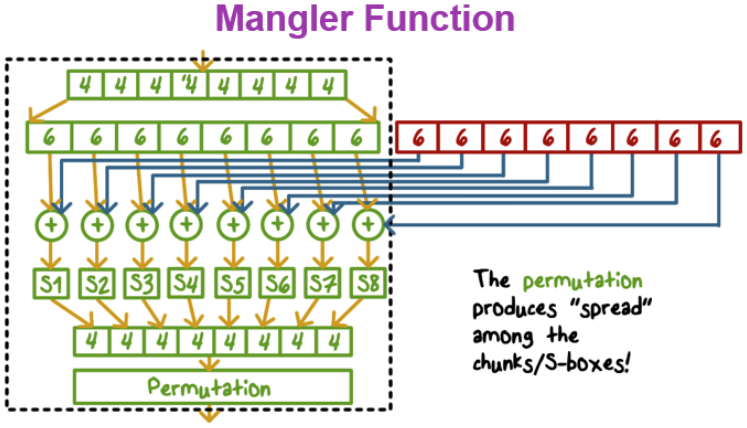
- 'H' = 0x48 = 0,1,0,0 1,0,0,0
- 'I' = 0x69 = 0,1,1,0 1,0,0,1

And 0xFAF2: 0xF is 1,1,1,1 0xA is 1,0,1,0 0xF is again, 1,1,1,1 and 0x2 is 0,0,1,0.

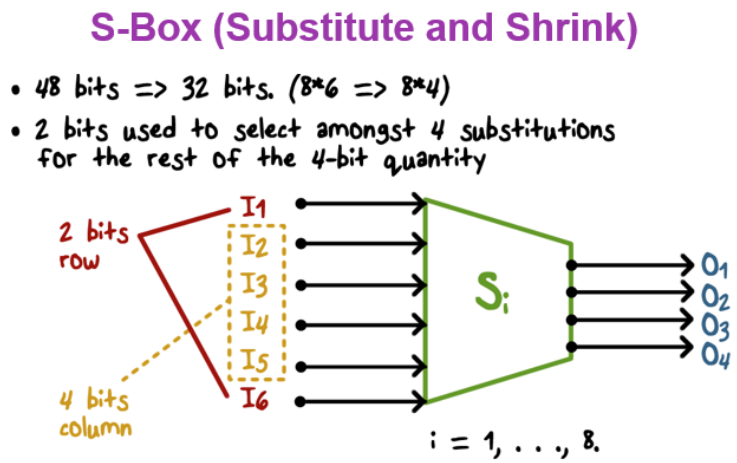
So what's the result of XOR Hi with this key, FA F2? Now, recall that:

- 0 XOR 0 = 0
- 0 XOR 1 = 0 (same as: 1 XOR 0 = 0)
- 1 XOR 1 = 0

Therefore, the 1st result nibble is 1,0,1,1 and the 2nd result nibble is 0,0,1,0 and the 3rd result nibble is 1,0,0,1 and the 4th result nibble is 1,0,1,1. And that is the encrypted version of the plain text.



The Mangler Function, performs the bulk of processing in a DES round. It takes the right half of the input, expands the 32 bit data into 48 bit data, and xor it with a per-round key. The result is that a 48 bit value is being substitute to a 32 bit value, and then the result is permutated. That is, the Mangler Function performs both substitution and permutation.



Now let's take a look at the S-Boxes. A S-Box substitutes a six bit value with a four bit value, using a predefined table. Therefore, we have eight S-Boxes in DES. That is we have eight predefined tables. Given a six bit value, the outer two bits [I1 & I6] are used to index into the rows of the table. And the middle four bits [I2..I5] index into the columns of the table, and the value in a table entry is the output of the four bit value. Again, these tables are pre-defined, and there are eight such tables in DES.

S-Box Quiz

For the given input, determine the output.

Now, let's do a quiz on S-Box. For the given input, what is the output? Again, the input is 6 bit and the output would be 4 bit.

S_5		Middle 4 bits of input															
		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Outer bits	00	0010	1100	0100	0001	0111	1010	1011	0110	1000	0101	0011	1111	1101	0000	1110	1001
	01	1110	1011	0010	1100	0100	0111	1101	0001	0101	0000	1111	1010	0011	1001	1000	0110
	10	0100	0010	0001	1011	1010	1101	0111	1000	1111	1001	1100	0101	0110	0011	0000	1110
	11	1011	1000	1100	0111	0001	1110	0010	1101	0110	1111	0000	1001	1010	0100	0101	0011

Input: 011011 **Output:**

We use the outer 2 bits to index into the rows of the table. The outer 2 bits is 01, so it's this row, 01. And the middle 4 bits are used to index into the columns of the table. And the middle 4 bits here are 1101. So the 1101, and that's this column. The entry in a table is the output. In this case, the entry is 1001, therefore 1001 is the output for bit value.

S-Box Quiz

For the given input, determine the output.

S_5		Middle 4 bits of input															
		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Outer bits	00	0010	1100	0100	0001	0111	1010	1011	0110	1000	0101	0011	1111	1101	0000	1110	1001
	01	1110	1011	0010	1100	0100	0111	1101	0001	0101	0000	1111	1010	0011	1001	1000	0110
	10	0100	0010	0001	1011	1010	1101	0111	1000	1111	1001	1100	0101	0110	0011	0000	1110
	11	1011	1000	1100	0111	0001	1110	0010	1101	0110	1111	0000	1001	1010	0100	0101	0011

Input: 011011 Output: 1001

Security of DES

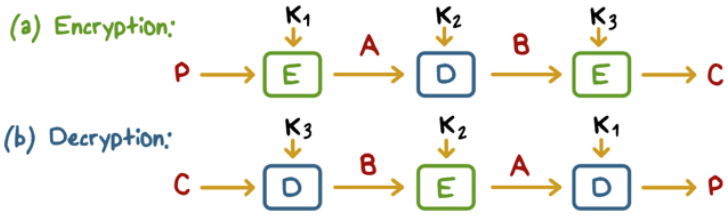
- **Key space is too small** (2^{56} keys)
  - Exhaustive key search relative easy with today's computers
- **S-box design criteria have been kept secret**
- **Highly resistant** to cryptanalysis techniques published years after DES

Let's discuss the security of DES.

First, we observe that the actual key value is only 56 bits. That means there are only a total of 2 to the 56 possible keys. This key space is too small, because an attacker can use brute force to find the correct key relatively, easily using today's computers.

Another issue with DES is that the design criteria for the S-boxes have been kept secret. On one hand, they have been shown to be resistant to attacks that were published years after DES was published. This means that DES is quite secure. On the other hand, because the design criteria have been kept a secret, one have to wonder what are the designer of this actually knew about these attacks years before they were published. One could argue that these design criteria should have been published, so the researchers can review them and improve upon them.

Triple DES



- K_1=K_3 results in an equivalent 112-bit DES which provides a sufficient key space
- Distinct K_1, K_2, K_3 results in an even stronger 168-bit DES
- Can run as a single DES with K_1 = K_2

The main shortcoming of DES, is that it uses a 56 bit key. Which means the keyspace is relatively small.

To overcome this, we can run DES multiple times, each using a different key. The standard is to run DES three times. And this is called a triple DES. The standard is to run the encryption process, then decryption process, then the encryption process again.

And correspondingly for decryption, this will mean to run the decryption process first, then the encryption process, then the decryption process again. And record that with DES, the decryption process is actually the same as the encryption process, only that we apply the keys in the reverse order. The advantage of using this order of operations, is that it supports multiple key lengths. In particular If key 1 is the same as key 3, then the result is a 112-bit DES. If all three keys are different, then the result is a 168-bit DES. If we set key 2 the same as key 1, then the the triple DES has in effect become a single DES with key three. This is useful for compatibility. For example a triple DES device can be configured to communicate with a single DES device, by simply setting key 2, the same as key 1.

DES Quiz

Check all the statements that are true:

- [ ] To decrypt using DES, same algorithm is used, but with per-round keys used in the reversed order
- [ ] With Triple DES the effective key length can be 56, 112, and 168
- [ ] Each round of DES contains both substitution and permutation operations
- [ ] The logics behind the S-boxes are well-known and verified

Now let's do a quiz on DES. Check all the statements that are true. First, to decrypt using DES, the same algorithm is used, but with per-round keys used in the reversed order. Second, with triple DES the effective key length can be 56, 112 and 168. Third, each round of DES contains both substitution and permutation operations. Fourth, the logics behind the S-boxes are well-known and verified.

The first statement, to decrypt using DES, the same algorithm is used but with per-round keys used in the reversed order. This is true. Second, with Triple DES the effective key length can be 56, 112, and 168. This is true. Third, each round of DES contains both substitution and permutation operations. This is true.

- [x] To decrypt using DES, same algorithm is used, but with per-round keys used in the reversed order
- [x] With Triple DES the effective key length can be 56, 112, and 168
- [x] Each round of DES contains both substitution and permutation operations
- [ ] The logics behind the S-boxes are well-known and verified

In particular, the mangler function has the S-boxes that performs substitution. And the mangler function

also performs permutation. Fourth, the logics behind the S-boxes are well-known and verified. This is false, because the design criteria of S-boxes have been kept a secret.

Advanced Encryption Standard

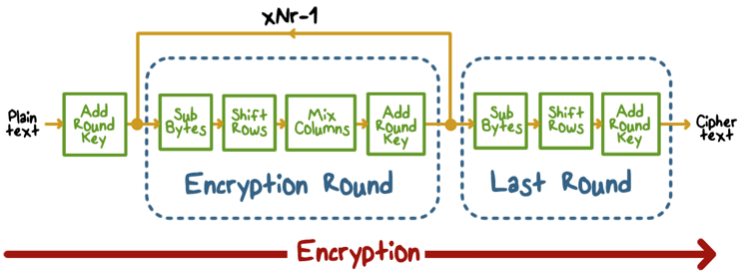
- In 1997, the **U.S. National Institute for Standards and Technology (NIST)** put out a public call for a replacement to DES
- It narrowed down the list of submissions to five finalists, and ultimately chose an algorithm (Rijndael) that is now known as the **Advanced Encryption Standard (AES)**
- New (Nov. 2001) symmetric-key NIST standard, replacing DES
- **Processes data in 128 bit blocks**
- **Key length can be 128, 192, or 256 bits**

Now let's discuss another symmetric encryption algorithm, the advanced encryption standard, or AES.

Recall that a major shortcoming of DES is that the key length is only 56 bit, and that is considered to be short. In other words, the key space is relatively small, and an attacker can use brute force to find the key using the power of today's computers. Of course, we can use Triple DES to increase the key length, but that would mean running DES three times, and that's not the most efficient way of doing encryption. Therefore, there was a need for a new encryption algorithm that can take longer key length, but also be efficient.

So in 1997, NIST put out a public call for a new encryption standard to replace DES. After a few rounds of submissions and reviews, AES was finalized, and it became a new standard, replacing DES. Like DES, AES is also a block cipher, whereas in DES, the input plaintext block is 64 bit, in AES it is 128 bit. In DES the key length is only 56 bit. In AES it can be 128, 192, or 256 bits. These key lengths are considered long enough to defeat brute force attempt to search for a key.

Advanced Encryption Standard



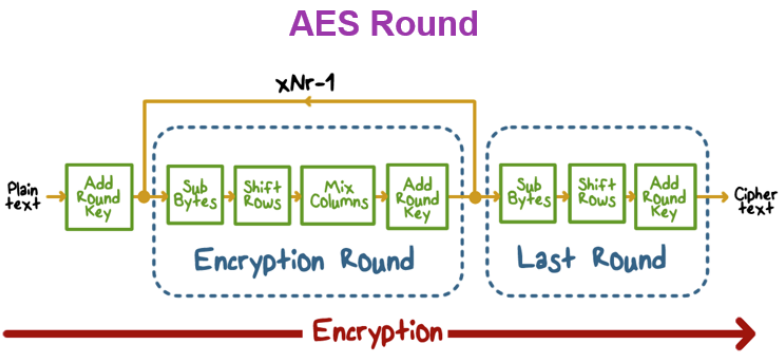
Here is a high level view of AES. The encryption process is as follows.

Again, the plaintext block is represented as a square matrix. We call it a state array, and we first XOR with the key. Again, the key is also represented as a square matrix.

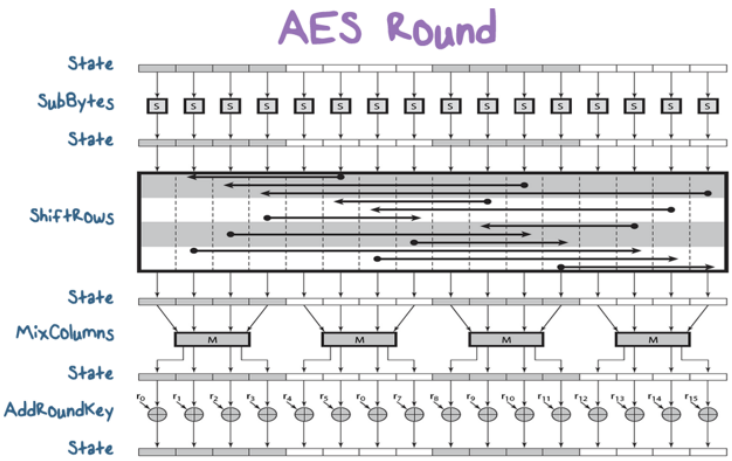
Then the state arrays go through multiple rounds of encryption. At each round, it goes through several operations that represent substitution and permutation, and also the per-round key is XOR'd to this state array. The operations at each round include substitute bytes. This involves using a table referred to as a S-box to perform byte to byte substitution of the block. Shift rows is a simple permutation that is performed row by row. Mixed columns is a substitution that alters each byte in a column as a function of all the bytes in the column. And then the result is XOR in a per-round key.

The operations of the last round includes substitute bytes, shift rows and add round key, and the result is a cipher text.

In AES, the decryption process runs the algorithm in the reverse direction. This means that each of these operations must be reversible. Let's take a look.



Now, adding round key involves the XOR operation. An XOR operation by itself is reversible. The other operations, meaning substitute bytes, shift rows, and mix columns, an inverse function is used in a decryption algorithm. By using this inverse function, we can reverse the action of substitute bytes that was performed in the encryption. Likewise, we can reverse the effects of shift rows and mix columns in the decryption process. Therefore, each of the operations are reversible. As a result, when we run the algorithm in the reverse order, we can decrypt the cipher text back into the plaintext.



Let's take a closer look at each round of AES.

Again, in AES, data is represented as state array. The first operation is substitute bytes. This involves using S-boxes to perform byte by byte substitution. And again, the result is stored in the state array.

The second operation is shift rows. And this is permutation that's performed row by row. Again, the result is in the state array.

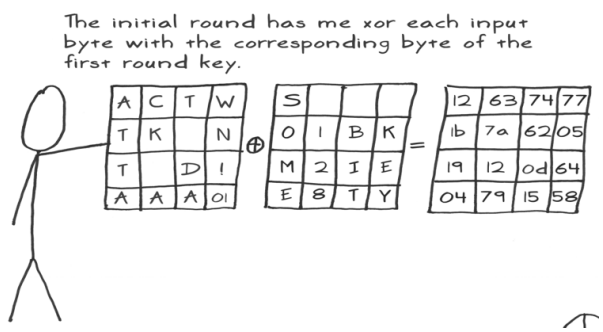
MixColumns is a substitution that alters each byte in a column as a function of all the bytes in the column. Again, the result is in the state array.

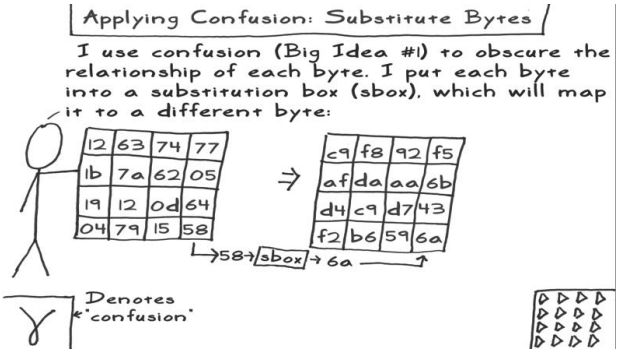
And then we XOR with the round key. The result is in the state array, and this feeds to the next round of AES.

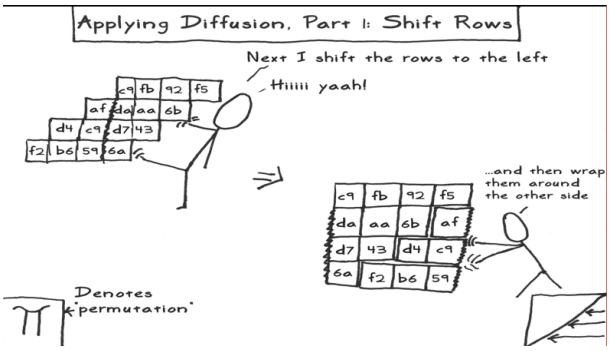
As you can see, each round of AES involves substitution, permutation, which represent confusion and diffusion, and also use the encryption key.

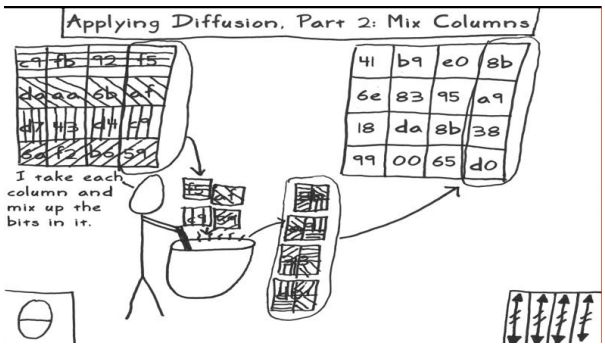
●A Stick Figure Guide to AES

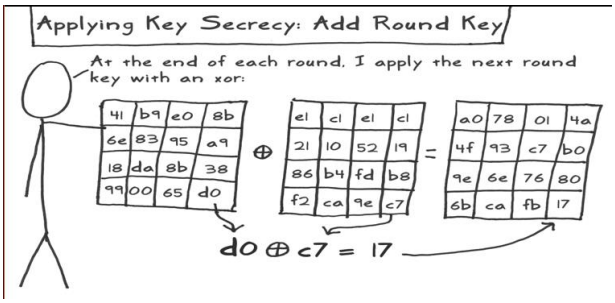
- http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html

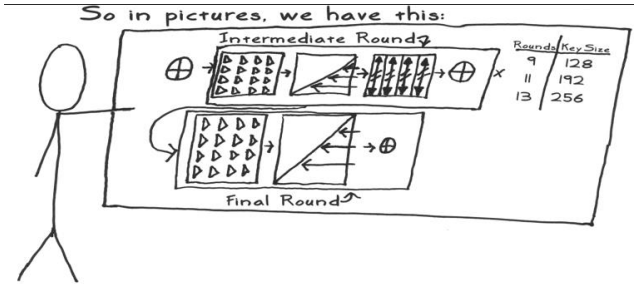


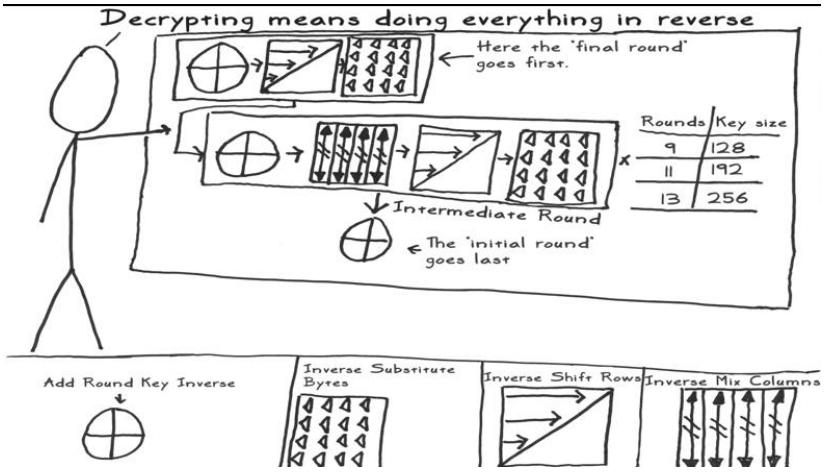












AES Encryption Quiz

Check all the statements that are true:

- ☐ To decrypt using AES, just run the same algorithm in the same order of operations
- ☐ Each operation or stage in AES is reversible
- ☐ AES can support key length of 128, 192, 256
- ☐ AES is much more efficient than Triple DES

Now let's do a quiz on AES. Check all the statements that are true. First, to decrypt using AES, just run the same algorithm in the same order of operations. Second, each operation or stage in AES is reversible. [Third] AES can support key length of 128, 192 and 256. AES is much more efficient than triple DES.

- ☐ To decrypt using AES, just run the same algorithm in the same order of operations
- ☑ Each operation or stage in AES is reversible
- ☑ AES can support key length of 128, 192, 256
- ☑ AES is much more efficient than Triple DES

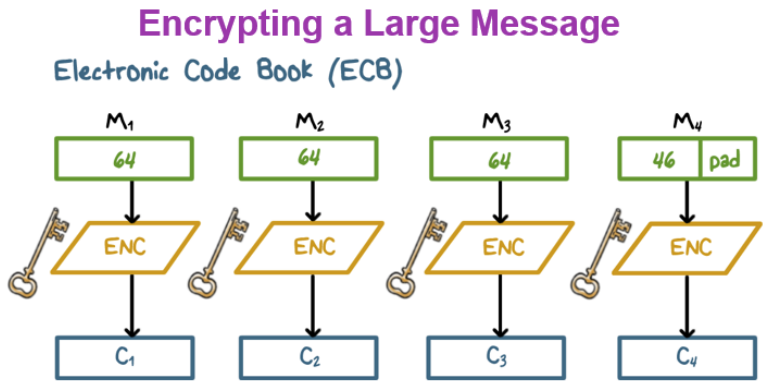
The first statement, to decrypt using AES, just run the same algorithm in the same order of operations. This is false. Because in AES, to decrypt, we run the algorithm in the reverse order of operations. Second, each operation or stage in AES is reversible. This is true. As we have discussed, all the operations in AES are reversible. Third, AES can support key length of 128, 192, and 256. This is true. Fourth, AES is much more efficient than Triple DES. This is also true. In fact, these [3rd and 4th] are the main motivations and requirements for developing AES in the first place.

Encrypting a Large Message

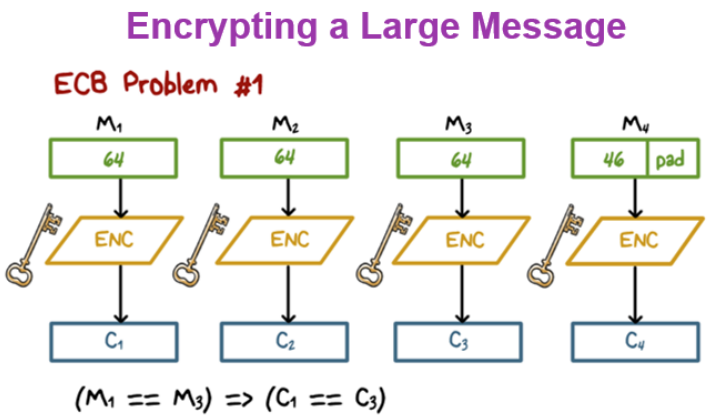
- Break a message into blocks
- Apply block cipher on the blocks
- Is that it?

So far we have discussed block ciphers, which take as an input a fixed length data block, say in 64-bit 40 years, or 128-bit for 8 years.

What if you want to encrypt a large message, that is, a message that's longer than 64-bit, or even 128-bit? The solution seems obvious. Why can't we just break a message into fixed-size blocks, apply a block cipher such as DES or AES on the blocks, then the collection of the ciphertext blocks is the ciphertext of the large message? Is that it? This seems to be very simple.



Let's take a look at a few schemes that encrypt a large message. The simplest way is the so-called electronic code book method or ECB. Here, the original large message is broken down into fixed-sized blocks, say 64-bit, and we pad the last block so that it is also 64-bit. And then each plaintext block is encrypted using the same key independently of each other. And the collection of these ciphertext blocks is the ciphertext of the original large message. The term code book is used, because for given key, there is a unique ciphertext block for every plaintext block. Therefore, we can construct a gigantic code book in which there's an entry for every possible plaintext block, and the table entry shows the corresponding ciphertext block. Of course, we construct one code book for each key. In practice, only the plaintext blocks that are used in an application need to be included in the code book. That is, the code book does not need to include all possible plaintext blocks. Therefore, it does not have to be gigantic. So this seems to be a very convenient of encrypting a large message.



There are major shortcomings with the electronic code-book method.

First, we observe that for the same key, if two plain-text blocks are the same, then their corresponding server text blocks are also the same. This is true if the two plain-text blocks are in the same message, or across two different messages. As long as they use the same key. This is a major weakness that can be exploited by cryptoanalysis. For example, if it is known that the message always starts out with certain predefined fields, then cryptoanalysis can have a number of known plain-text, server-text pairs to work with. As another example, if the message has repetitive elements, then these elements can be identified by cryptoanalysis as well. In other words, whenever we see two identical cipher-text bocks, we know that the corresponding plain-text blocks are exactly the same. And if we have obtained the plain-text block off a cipher-text block, then if you see the same cipher-text block again, we know the corresponding plain-text block already. In other words, ECB does not provide a very strong confidentiality protection.

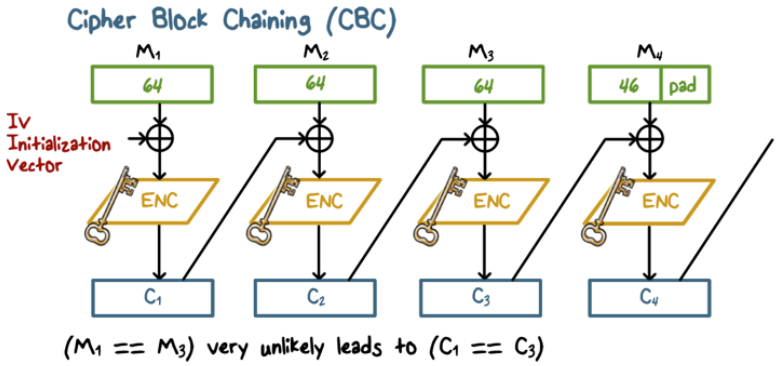
Encrypting a Large Message

ECB Problem #2

- **Lack the basic protection against integrity attacks** on the ciphertext at message level (i.e., multiple cipher blocks)
- Without additional integrity protection
  - **cipher block substitution** and rearrangement attacks
  - **fabrication** of specific information

Another problem with ECB is that the plain text blocks are encrypted into ciphertext blocks independently of each other. Therefore an attacker can use a previously captured ciphertext block to sub out a current ciphertext block. Or we arrange the order of the ciphertext blocks. The result is violation of message integrity. For example the attacker can fabricate specific information.

Encrypting a Large Message



The most widely used approach to encrypt a large message is CBC. It stands for cipher block chaining.

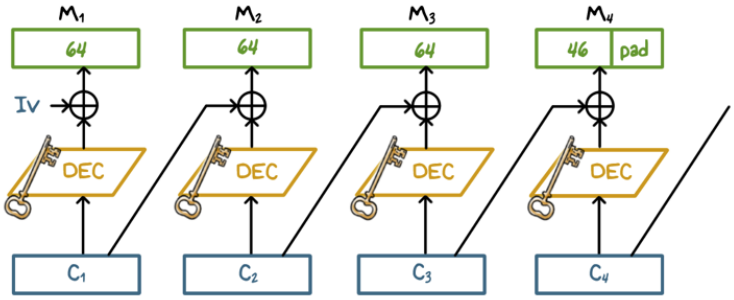
In CBC, the input to the encryption algorithm is the result of xoring the previous ciphertext block and the current plaintext block.

To encrypt the first plaintext block, we use a so-called IV, stands for initialization vector. And we XOR IV with the plaintext block and give the input to the encryption process to produce the first ciphertext block.

And then the first ciphertext block is used to XOR with the second plaintext block as input to the encryption process to produce the second ciphertext block and so on so forth. Again in CBC, the current ciphertext block depends not only on the current plaintext block but also the previous ciphertext block. Which in turn depends on not only the previous plaintext block but also the ciphertext block prior to that.

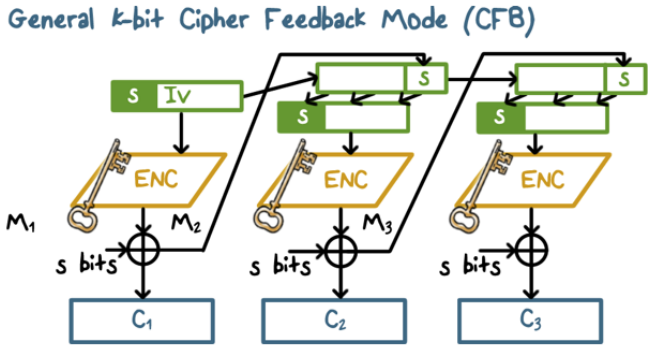
In other words, in analysis it is very hard to figure out the plaintext block just looking at the current ciphertext block. More importantly, if two plaintext blocks are exactly the same, meaning that they repeat in the same message or in two different messages, their ciphertext blocks are not likely to be the same. In addition, if an attacker attempts to swap in a different ciphertext block or rearrange the orders of the ciphertext blocks, the ciphertext is not going to decrypt properly into the plaintext.

CBC Decryption



Here's how decryption in CBC works. A ciphertext block is decrypted, and then the result is XORed with the previous ciphertext block to produce the current plaintext block. For the first ciphertext block after decryption, the result is XORed with Iv to produce the first plaintext block. Therefore, the IV must be known to both the sender and receiver.

General K-Bit Cipher Feedback Mode (CFB)



Protecting Message Integrity

- **Only send last block of CBC** (CBC residue) along with the plaintext
- Any modification in plaintext result in a CBC residue computed by the receiver to be different from the CBC residue from the sender
  - **Ensures integrity**

So far we are focused largely on message confidentiality. What if we want to protect message integrity? That is, we want to prevent or detect any unauthorized modification to the message.

One standard approach is to send the last block of CBC, also called a CBC residue, along with a plain text. Now, if an attacker intercepts the message and modify the plain text, the attacker does not have the correct key, therefore he cannot compute a correct CBC residue with the modified plain text. So he's forced to resend the modified plain text with original CBC residue. The CBC residue that the receiver computes on the modified plaintext will not be the same as the CBC residue that he receives. Therefore, the receiver would know that the plaintext has been modified. In other words, the CBC residue provides a protection of message integrity.

Protecting Message Integrity

- Simply sending all CBC blocks (for confidentiality) replicating last CBC block (for integrity) **does not work**
- **Should use two separate secret keys:** one for encryption and the other for generating residue (two encryption passes)
- Or, **CBC** (message|hash of message)

Now, what if we want both confidentiality and integrity?

How about we encrypt the message and send all the CBC blocks, and that's the ciphertext blocks for protecting confidentiality. And then, we'll replicate the last block, which is a CBC residue, and that's for integrity. Would that work?

Obviously this approach does not work because if we simply replicate the last block, we are not adding anything. That is, we are not doing anything in addition to protecting the confidentiality. Therefore there is no way this approach will work to provide both, confidentiality and integrity.

We can instead use two different keys. One for producing CBC blocks for confidentiality, and the other for generating the residue for integrity, that is we encrypt twice using two different keys. Another approach is that we first compute a hash of the message. Appended it to the message and then encrypt the whole entity.

CBC Quiz

Put a check next to the statements that are true:

- ☐ CBC is more secure than ECB
- ☐ We can have both confidentiality and integrity protection with CBC by using just one key

Now let's do a quiz on CBC. Check the statements that are true. First, CBC is more secure than ECB. Second, we can have both confidentiality and integrity protection with CBC by using just one key.

- ☑ CBC is more secure than ECB
- ☐ We can have both confidentiality and integrity protection with CBC by using just one key

The first statement, CBC is more secure than ECB, this is true. This is because in CBC, the current ciphertext block depends not only on the current plaintext block, but also the previous ciphertext block. Such chaining provides better confidentiality and integrity protection than ECB. The second statement, we can have both confidentiality and integrity protection with CBC by using just one key. This is false. As we have discussed, we need to use two different keys and encrypt a message twice, one for protecting confidentiality and the other for protecting integrity.

Symmetric Encryption
Lesson Summary

- Need both confusion and diffusion
- DES: input 64-bit, key 56-bit; encryption and decryption same algorithms but reversed per-round key sequence
- AES: input 128-bit, key 128/192/256 bits; decryption the reverse/inverse of encryption
- Use cipher-block-chaining to encrypt a large message
- Last CBC block can be use as MIC; use different keys for integrity and confidentiality

Block ciphers need to use both confusion and diffusion operations. AES uses longer keys and is more secure than DES. We should use CBC to encrypt a large message. We can also use the last CBC block as the message integrity code.