Algorithm Types and Modes

- There are two basic types of symmetric ciphers: Block and Stream ciphers.

Block ciphers

- Operate on blocks of plaintext and ciphertext—usually of 64 bits but sometimes longer.
- The same plaintext block will always encrypt to the same ciphertext block, using the same key.

Stream ciphers

- Operate on streams of plaintext and ciphertext one bit or byte (sometimes even one 32-bit word) at a time.
- With a stream cipher, the same plaintext bit or byte will encrypt to a different bit or byte every time it is encrypted.

Block Cipher Cryptographic Modes

- Block ciphers are used in a variety of **modes of operation** in order to encrypt large blocks of plaintext.

- A cryptographic **mode** is a **combination** of the **basic cipher** with some sort of **feedback**, and some **simple operations**.

- The operations are simple because the security **is a function of the underlying cipher and not the mode.** More importantly, the cipher mode should not compromise the security of the underlying algorithm.

- These modes of operation have three main goals:

    - Confidentiality (the obvious one)
    - Authenticity (validate and authenticate the sender)
    - Integrity (Ensure that the ciphertext was not corrupted during transmission)

- Beyond being simply an encryption algorithm, a block cipher can be used as a building block for implementing many other ciphers and protocols, such as implementing various types of block-based encryption schemes

    - stream ciphers
    - hash functions
    - message authentication codes
    - key establishment protocols

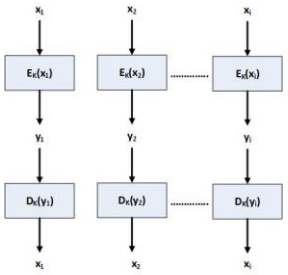
Electronic codebook (ECB)

- This mode is a most straightforward way of processing a series of sequentially listed message blocks.
- The first block of plaintext is encrypted with a key to produce the first block of ciphertext. Then the second block of plaintext follows the same process with **same key** and so on until the whole file is encrypted.
- In ECB mode a block of plaintext is converted to ciphertext using a **fixed mapping** between the blocks of plaintext and the corresponding blocks of cipher text using a **key**. Thus every different key will generate a different codebook.
- The ECB process for fixed-sized, b-bit blocks is defined as follows:

  $E_k(x_i) \triangleq encryption\ of\ a\ plaintext\ block\ x_i, with\ key, k$

  $D_k(y_i) \triangleq decryption\ of\ a\ ciphertext\ block\ y_i, with\ key, k$

- The following diagram depicts the ECB process, where each block is encrypted separately:



ECB Encryption and Decryption using the same Key

- The encryption of a plaintext block is the corresponding ciphertext block entry in the code book. Since the same block of plaintext always encrypts to the same block of ciphertext, it is theoretically possible to create a "**code book"** of a set of plaintext block and corresponding ciphertext blocks.
- However, if the block size is 64 bits, the code book will have 2^{64} entries, which is much too large to precompute and store. Keep in mind that every key has a different code book.
- Each plaintext block is encrypted independently. The encryption process can encrypt the blocks in the middle first, then the blocks at the end, and finally the blocks in the beginning.
- This is very well-suited to encrypted files that are accessed randomly, like a database. If a database is encrypted with ECB mode, then any record can be added, deleted, encrypted, or decrypted independently of any other record.
- In addition, this process lends itself very well to parallel processing. Multiple encryption processors (or cores) can encrypt or decrypt different blocks independently of each other.

- The problem with ECB mode is that if a cryptanalyst has the plaintext and ciphertext for several, repetitive messages, a code book can be compiled without knowing the key.

- This makes it not very secure for long segments of plaintext, especially plaintext containing repetitive information (particularly if the nature of what is repetitive in the plaintext is known to the attacker).

- In most real-world situations, fragments of messages tend to repeat. Different messages may have bit sequences in common. Computer-generated messages, like electronic mail, may have regular structures (headers for example).

- Messages may be highly redundant or have long strings of zeros or spaces. If a cryptanalyst learns that the plaintext block "5e081bc5" encrypts to the ciphertext block "7ea593a4," then it becomes a trivial matter to decrypt that ciphertext block whenever it appears in another message.

- Messages that have a lot of redundancies, and with a tendency to appear in the same places in different messages, provide a cryptanalyst with a lot of valuable information.

- Using this information, statistical attacks can be applied to the obtained plaintext, regardless of the strength of the block cipher.

- This vulnerability is greatest at the beginning and end of messages, where well-defined headers and footers contain information about the sender, receiver, date, and so on.

- This problem is referred to as **stereotyped beginnings** and **stereotyped endings**.

- Practically any application data tends to have partial information which can be guessed with a high degree of success. For example, the range of salary can be guessed, or in the case of many public salary scales, is available in public documents.

- A ciphertext using ECB mode is known to contain salary information will become very vulnerable to a trial-and-error attack.

- From a practical standpoint deterministic ciphers such as ECB mode should never be used to send long, repetitive messages. It is used primarily for secure transmission of short pieces of information, such as an encryption keys or passwords.

- Another shortcoming of ECB is that the length of the plaintext message must be integral multiple of the block size. When that condition is not met, the plaintext message must be **padded** appropriately.

- Most messages don't divide neatly into 64-bit (or whatever size) encryption blocks; there is usually a short block at the end. ECB requires 64-bit blocks.

- To solve this problem, the last block is padded with a repetitive known bit pattern (01010101 for example) to fill it out to a complete block. In order to delete the padding after decryption, a count of the number of padding bytes is added as the last byte of the last block.

- For example, assume the block size is 64 bits and the last block consists of 3 bytes (24 bits). Five bytes of padding are required to make the last block 64 bits; add 4 bytes of zeros and a final byte with the number 5.

- After decryption, delete the last 5 bytes of the last decryption block. For this method to work correctly, every message must be padded. Even if the plaintext ends on a block boundary, you have to pad one complete block.

- ECB has the following advantages:

    - There is no need for block synchronization between the sender and receiver.
    - There is no error propagation; bit errors caused by noisy channels only affect the corresponding block but not succeeding blocks.
    - Block cipher operations can be parallelized for high-speed implementations.

- The main disadvantage of the algorithm is that it is highly deterministic; identical plaintext result in identical ciphertext.

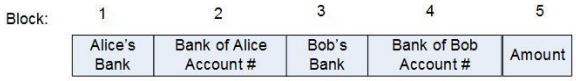
- In addition, plaintext blocks are encrypted independently of previous blocks, which means that an attacker may reorder ciphertext blocks, which results in valid but incorrect plaintext.

Block Replay

- A more serious problem with ECB mode is that an adversary could modify encrypted messages without knowing the key, or even the algorithm, in such a way as to fool the intended recipient.

- Basically, once a particular plaintext to ciphertext block mapping $x_i \rightarrow y_i$ is known, a subsequent sequence of ciphertext blocks can easily be manipulated.

- To illustrate the problem, consider an electronic bank transfer system that moves money between accounts in different banks. The banks agree on a transfer protocol that uses a standard message format for money transfer that is as shown below:



- Let as assume that Mallory has set accounts at both banks and also assume that the encryption key between banks does not change frequently.

- Mallory then starts to listen in on the communications link between the two banks in order to collect information and the use it to steal money.

- Mallory now sends $100 transfers from his account at Alice's bank to his account at Bob's bank several times.

- He examines the recorded messages looking for repeating, identical messages, which are the messages are the ones authorizing the $100 transfers to his account.

- If he finds more than one pair of identical messages, he does another money transfer and records those results. Eventually he can isolate the message that authorized his money transaction.

- He stores blocks 1, 3 and 4 of these transfers. He can now **replace block 4** of other transfers (having the same blocks 1 & 3) with the block 4 that he stored earlier.

- Now he can insert that message into the communications link at will. Every time he sends the message to Bank of Bob, another $100 (or larger amounts) will be credited to his account.

- When the two banks reconcile their transfers (probably at the end of the day), they will notice the phantom transfer authorizations; but by then Mallory will have already withdrawn the money, changed his identity and vanished to a tax-free haven.

Cipher Block Chaining Mode (CBC)

- Chaining adds a **feedback** mechanism to a block cipher: The results of the encryption of previous blocks are fed back into the encryption of the current block. Thus, each block is used to modify the encryption of the next block.

- The plaintext is XOR'd with the previous ciphertext block before it is encrypted. After a plaintext block is encrypted, the resulting ciphertext is also stored in a feedback register.

- Before the next plaintext block is encrypted, it is XOR'd with the feedback register to become the next input to the encrypting routine.

- The resulting ciphertext is again stored in the feedback register, to be XOR'd with the next plaintext block, and so on until the end of the message.

- CBC mode forces identical plaintext blocks to encrypt to different ciphertext blocks only when some previous plaintext block is different.

- Some messages have a common header which has identical fields such as "To", or a "From" field, etc. While block replay would not be possible, the identical beginning might give a cryptanalyst some useful information.

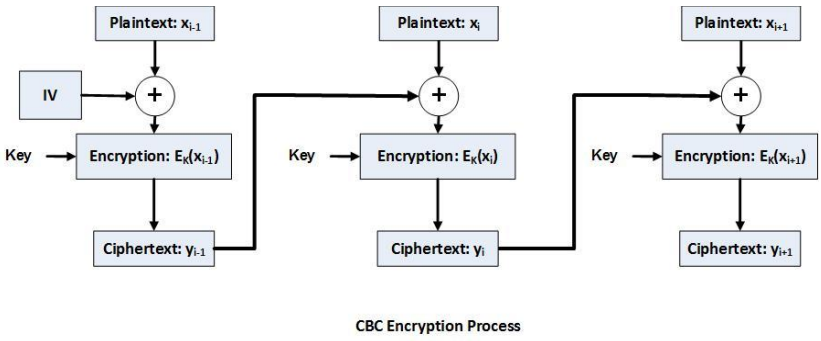
- This is prevented by encrypting random data as the first block. This block of random data is called the **initialization vector** (denoted **IV**), initializing variable, or initial chaining value.

- The initialization vector can be sent separately as a short message using the ECB mode, although it is not necessary to encrypt it all (later discussion).

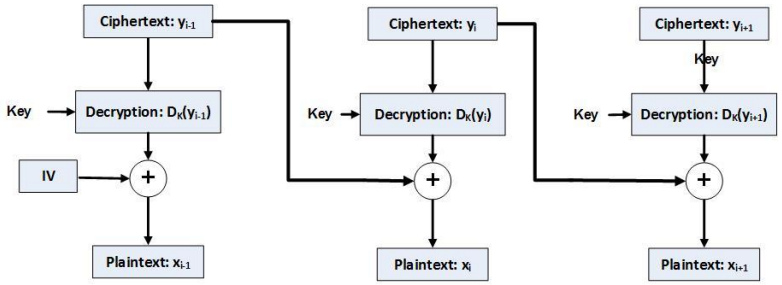
- Mathematically this process is defined as follows:

    - Encryption (first block): $\boldsymbol{y_1 = E_k(x_1 \oplus IV)}$
    - Encryption (general block): $\boldsymbol{y_i = E_k(x_i \oplus y_{i-1}),\ \ i \geq 2}$
    - Decryption (first block): $\boldsymbol{x_1 = D_k(y_1 \oplus IV)}$
    - Decryption (general block): $\boldsymbol{x_i = D_k(y_i \oplus y_{i-1}),\ \ i \geq 2}$

- The following diagrams illustrate the mathematical model:



CBC Encryption Process



CBC Decryption Process

- Using the IVs as above, identical plaintext messages will encrypt to different ciphertext messages. Thus, it is impossible for an eavesdropper to attempt a block replay attack, and more difficult to build a code book.

- While the IV should be unique for each message encrypted with the same key, it is not an absolute requirement.

- The IV need not be secret; it can be transmitted in the clear with the ciphertext. Assume that we have a message of several blocks: $x_1, x_2, \ldots, x_n$:

    - x_1 is encrypted with the **IV** to produce the ciphertext: y_1.
    - x_2 is encrypted using (y_1 + **IV**) as the new "IV" to generate the ciphertext: y_2.
    - x_3 is encrypted using $y_1, y_2,$ and the **IV**, and so on.

- So, if there are ***n*** blocks, there are ***n* - 1** exposed "IVs," even if the original IV is kept secret. It is very difficult to derive any information from the ciphertext or run known-plaintext attacks since the **beginning** of the plaintext (the IV) will be random "noise".

- The important thing is that the initial IV must be **unique** and truly random for every session.

Error Propagation

- CBC mode can be characterized as feedback of the ciphertext at the encryption end and feedforward of the ciphertext at the decryption end.

- Thus, a single bit error in a plaintext block will affect that ciphertext block and all subsequent ciphertext blocks. However, decryption will reverse that effect, and the recovered plaintext will have the same single error.

- Ciphertext errors result from a noisy communications path or a malfunction in the storage medium. In CBC mode, a single-bit error in the ciphertext affects one block and one bit of the recovered plaintext.

- The block containing the error is completely garbled. The subsequent block has a 1-bit error in the same bit position as the error.

- This property of taking a small ciphertext error and converting it into a large plaintext error is called **error extension**.

- Blocks after the second are not affected by the error, so CBC mode is **self-recovering**. Two blocks are affected by an error, but the system recovers and continues to work correctly for all subsequent blocks.

- CBC is an example of a block cipher being used in a self-synchronizing manner, but only at the block level.

- While CBC mode recovers quickly from bit errors; however it will not recover at all from synchronization errors.

- If a bit is added or lost from the ciphertext stream, then all subsequent blocks are shifted one bit out of position and decryption will generate garbage indefinitely.

- Any cryptosystem that uses CBC mode must ensure that the block structure remains intact, either by framing or by storing data in multiple-block-sized chunks.

Security Problems

- In CBC mode, it is important to structure the plaintext so that the point where the message ends is clearly identifiable in order to detect the addition (maliciously) of extra blocks.

- Mallory can alter a ciphertext block to introduce controlled changes in the following decrypted plaintext block. For example, if a single ciphertext bit is toggled, the entire block will decrypt incorrectly, but the following block will have a 1-bit error in the corresponding bit position.

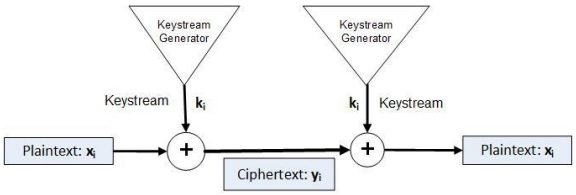
- The entire plaintext message should include some kind of controlled redundancy or authentication.

- Finally, although plaintext patterns are concealed by chaining, very long messages will still have patterns.

- The birthday paradox predicts that there will be identical blocks after $\mathbf{2^{m/2}}$ blocks, where **m** is the block size. For a 64-bit block size, that's about 34 gigabytes (quite long).

Stream Ciphers

- Stream ciphers convert plaintext to ciphertext 1 bit at a time. The simplest implementation of a stream cipher is shown below:



Stream Cipher

- A keystream generator (running-key generator) outputs a stream of bits: $\mathbf{k_1, k_2, .., k_i}$. This keystream (running key) is XOR'd with a stream of plaintext bits, $\mathbf{x_1, x_2,..., x_i}$, to produce the stream of ciphertext bits:

$$\boldsymbol{y_i = (x_i \oplus k_i)}$$

- At the decryption end, the ciphertext bits are XOR'd with an identical keystream to recover the plaintext bits:

$$\boldsymbol{x_i = (y_i \oplus k_i)}$$

- The system's security depends entirely on the implementation of the keystream generator. If the keystream generator outputs a continuous stream of zeros, the ciphertext will equal the plaintext, which is worthless.

- If the keystream generator produces a repeating 16-bit pattern, the algorithm will be a simple XOR with negligible security.

- If the keystream generator spits out an endless stream of true random (not pseudo-random) bits, then it becomes a one-time pad and perfect security.

- The reality of stream cipher security lies somewhere between the simple XOR and the one-time pad.

- The closer the keystream generator's output is to random, the harder time a cryptanalyst will have breaking it.

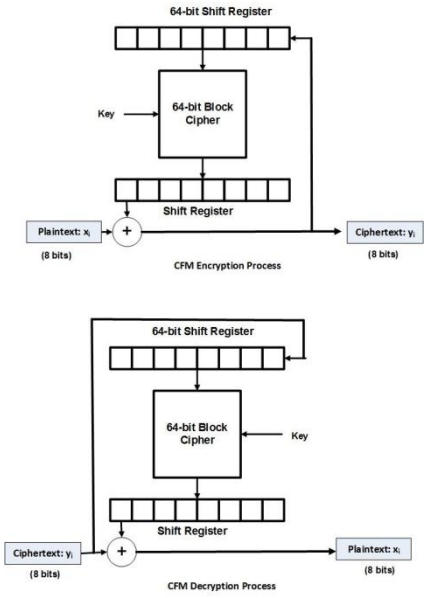
Cipher-Feedback Mode

- Uses a block cipher as a building block for **asynchronous stream cipher** (similar to Output Feedback Mode- OFB – covered later). This is also known as "Ciphertext Feedback Mode" (CFB).

- In CFB mode, data can be encrypted in units smaller than the block size. The following example will encrypt one ASCII character at a time (this is called **8-bit CFB**)

- Key stream k_i is generated in a block-wise fashion and is also a function of the ciphertext. Initializing vectors (IV) are used with CFB encryption to make it nondeterministic

- The following diagram illustrates the CFB encryption/decryption (one byte at a time):



- The above diagram illustrates a 8-bit CFB mode working with a 64-bit block algorithm. A block algorithm in CFB mode operates on a queue the size of the input block.

- Initially, the queue is filled with an IV, as in CBC mode. The queue is encrypted and the left-most eight bits of the result are XOR'd with the first 8-bit character of the plaintext to become the first 8-bit character of the ciphertext.

- This character can now be transmitted. The same eight bits are also moved to the right-most eight bit positions of the queue, and all the other bits move eight to the left. The eight left-most bits are discarded.

- Then the next plaintext character is encrypted in the same manner. Decryption is the reverse of this process. On both the encryption and the decryption side, the block algorithm is used in its encryption mode.

- A general n-bit CFB algorithm process is mathematically defined as follows:

    - Encryption (first block): $\boldsymbol{y_1 = E_k(IV) \oplus x_1}$
    - Encryption (general block): $\boldsymbol{y_i = E_k(y_{i-1}) \oplus x_i,\ \ i \geq 2}$
    - Decryption (first block): $\boldsymbol{x_1 = D_k(IV) \oplus y_1}$
    - Decryption (general block): $\boldsymbol{x_i = D_k(y_{i-1}) \oplus y_i,\ \ i \geq 2}$

Initialization Vector

- To initialize the CFB process, the input to the block algorithm must be initialized with an IV. Like the IV used in CBC mode, it need not be secret.

- However, the IV must be unique, though. (This is different from the IV in CBC mode, which should be unique but does not have to be.) If the IV in CFB is not unique, a cryptanalyst can recover the corresponding plaintext.

- The IV must be changed with every message. It can be a serial number, which increments after each message and does not repeat during the lifetime of the key.

Error Propagation

- With CFB mode, an error in the plaintext affects all subsequent ciphertext and reverses itself at decryption.

- However, the initial effect of a single-bit error in the ciphertext is a single error in the plaintext.

- Subsequently, the error enters the shift register, where it causes ciphertext to be garbled until it falls off the other end of the register.

- In 8-bit CFB mode, 9 bytes of decrypted plaintext are garbled by a single-bit error in the ciphertext.

- After that, the system recovers and all subsequent ciphertext is decrypted correctly. In general, in n-bit CFB a single ciphertext error will affect the decryption of the current and following **m/n-1** blocks, where **m** is the block size.

- This type of error propagation results is a security problem. If Mallory knows the plaintext of a transmission, he can toggle bits in a given block and make it decrypt to whatever he wants.

- The **next** block will decrypt to garbage, but the damage may already be done. And he can change the final bits of a message without detection.

Output-Feedback Mode

- **Output-feedback (OFB)** mode is a method that is used to build a synchronous stream cipher from a block cipher.

- It is similar to CFB mode, except that ***n* bits** of the previous output block are moved into the right-most positions of the queue.

- The key stream is not generated bitwise but in a block-wise fashion. The cipher output generates key stream bits $\mathbf{S_i}$. This is sometimes called **internal feedback**, because the feedback mechanism is independent of both the plaintext and the ciphertext streams

- Key stream independent of ciphertext and can be generated in parallel.

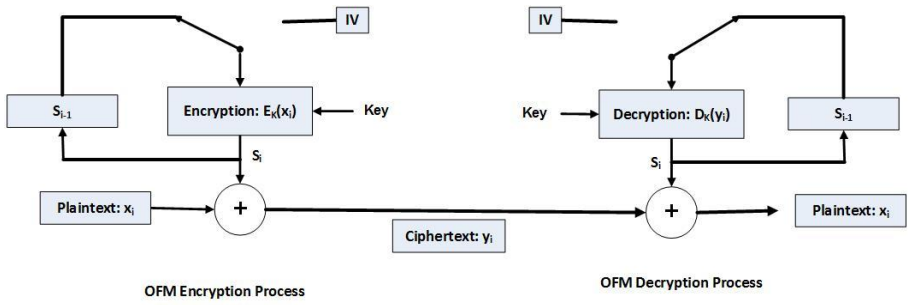
- Decryption is the reverse of this process. This is called ***n*-bit** OFB. On both the encryption and the decryption sides, the block algorithm is used in its encryption mode.

- The OFB shift register must also be initially loaded with an IV. It should be unique but does not have to be secret.

- A general n-bit OFB algorithm process is mathematically defined as follows:

    - Encryption (first block): $\boldsymbol{s_1 = E_k(IV) and\ y_1 = s_1 \oplus x_1}$
    - Encryption (general block): $\boldsymbol{s_i = E_k(s_{i-1})\ and\ y_i = s_i \oplus x_i,\ \ i \geq 2}$
    - Decryption (first block): $\boldsymbol{s_1 = D_k(IV)\ and\ x_1 = s_1 \oplus y_1}$
    - Decryption (general block): $\boldsymbol{x_i = D_k(s_{i-1})\ and\ x_i = s_i \oplus y_i,\ \ i \geq 2}$

- The following diagram illustrates the encryption and decryption process:



Error Propagation

- OFB mode has no error extension. A single-bit error in the ciphertext causes a single-bit error in the recovered plaintext.

- However, there is no tolerance for loss of synchronization. If the shift registers on the encryption end and the decryption end are not identical, then the recovered plaintext will be gibberish.

- Any system that uses OFB mode must have a mechanism for detecting a synchronization loss and a mechanism to fill both shift registers with a new (or the same) IV to regain synchronization.

OFB Security Issues

- Analysis of OFB mode has demonstrated that OFB should be used only when the feedback size is the same as the block size (use a 64-bit algorithm in 64-bit OFB mode).

- OFB mode XOR's a keystream with the text and this keystream will eventually repeat. It is important that it does not repeat with the same key; otherwise, there is no security.

- When the feedback size equals the block size, the block cipher acts as a permutation of ***m*-bit** values (where *m* is the block length) and the average cycle length is $2^m - 1$.

- For a 64-bit block length, this is a very long number. When the feedback size *n* is less than the block length, the average cycle length drops to around $2^{m/2}$. Thus, for a 64-bit block cipher, this becomes 2^{32}, which is not long enough.

Counter Mode (CTR)

- CTR is a block cipher generates a stream cipher (similar to OFB and CFB). The key stream is computed in a block-wise fashion

- It uses sequence numbers as the input to the algorithm instead of using the output of the encryption algorithm to fill the register, the input to the register is a counter.

- The input to the block cipher is a counter. After each block encryption, the counter increments by some constant, typically one.

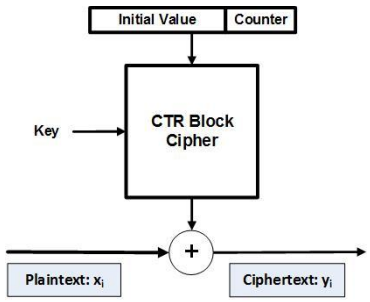
- The synchronization and error propagation characteristics of this mode are identical to those of OFB.

- Counter mode solves the OFB mode problem of *n*-bit output where *n* is less than the block length.

- The general n-bit CTR algorithm process is mathematically defined as follows:

    - Encryption: $\boldsymbol{y_i = E_k(IV\,||CTR_i) \oplus x_i, i \geq 1}$
    - Decryption: $\boldsymbol{x_i = E_k(IV\,||CTR_i) \oplus y_i, i \geq 1}$

- The following diagram illustrates the encryption/decryption process (note that due to the XOR operation there is no need to separate the two processes):



CTR Cipher Process

- AES-CTR is an example of CTR mode implementation for Wi-Fi Protected Access WPA2.

- Unlike OFB and CFB modes, the CTR mode can be parallelized since the 2nd encryption can begin before the 1st one has finished. This is very useful for high-speed network implementations (routers).

Security of Block Ciphers

- There are two approaches to increasing the security of block ciphers such as DES:
    - Multiple encryption (3DES, NDES)
    - Key whitening
- In the case of multiple encryption the plaintext (**C**) is first encrypted with a 1st key $\mathbf{k_1}$, and the resulting ciphertext is encrypted again using a 2nd key $\mathbf{k_2}$.
- Block ciphers such as DES, 3DES, etc., are vulnerable to an attack known as the "meet-in-the-middle" attack, which is described next.

Meet-in-the-middle attack

- The meet-in-the-middle attack belongs to a class of attacks known as "plaintext attacks". The attacker has to know some parts of the plaintext and the corresponding ciphertext.
- Using meet-in-the-middle attacks it is possible to break ciphers, which have two or more secret keys for multiple encryptions using the same algorithm, 3DES for example. It specifically targets block ciphers such as DES, 3DES, etc.
- This was first presented by Diffie and Hellman for cryptanalysis of the DES algorithm. The attack operates by exponentially reducing the number of brute force permutations required to decrypt text that has been encrypted by more than one key.
- The basic technique is to apply brute force techniques to both the plaintext and ciphertext of a block cipher.
- Brute force attempts are carried out to encrypt the plaintext according to various keys to achieve an intermediate ciphertext (a text that has only been encrypted by one key).
- Simultaneously, attempts to decrypt the ciphertext according to various keys are carried out, seeking a block of intermediate ciphertext that is the same as the one achieved by encrypting the plaintext.
- If there is a match of intermediate ciphertext, it is highly probable that the key used to encrypt the plaintext and the key used to decrypt the ciphertext are the two encryption keys used for the block cipher.
- That is how the attack derives its name; because the attacker tries to break the two-part encryption method from both sides simultaneously, a successful effort enables the attacker to meet in the middle of the block cipher.

- Although a meet-in-the-middle exploit can make the attacker's job easier, it cannot be conducted without a piece of plaintext and corresponding ciphertext.

- That means the attacker must have the capacity to store all possible intermediate ciphertext values from both the brute force encryption of the plaintext and decryption of the ciphertext.

- The attack is well beyond the reach of an average attacker due to the fact that it will require and enormous amount of computing resources (processing and storage).

Attack details

- A typical block cipher (using multiple keys), which is the target of a meet-in-the-middle attack, can be defined as follows:

  $$C = E_{K2}((k_2, E_{K1}(k_1, M))$$
  $$M = D_{k1}(k_1, D_{k2}(k_2, M))$$

- **Phase 1:**

  - For all given plaintext, ciphertext pairs, (m_i, c_i) we start on one side and brute force the encryption for all possible keys: $k_{1,i}$ $i = 1,2,\ldots..,\ 2^k$

  - A lookup table containing 2^k rows (each $n + k$ bits wide) is computed (and stored) for all possible values of ciphertext:

    $$c_{k1,i} = E_{K1}(k_1, m_i)$$

  - The lookup table is ordered by the results of the encryption ($c_{k1,i}$)

- **Phase 2:**

  - Simultaneously with phase 1, the encryption values generated in phase 1 are brute-forced (using decryption with k_2) for all possible values of the ciphertext:

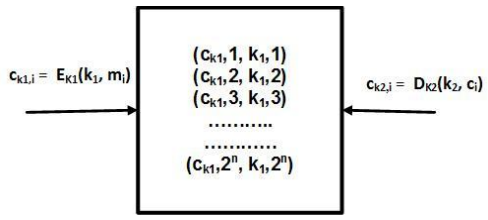
    $$c_{k2,i} = D_{K2}(k_2, c_i)$$

  - For each $c_{k2,i}$ a comparison is done to check whether $c_{k2,i}$ is equal to any $c_{k1,i}$ value in the table of the first phase.
  - The attacker is attempting to verify the following equation:

    $$D_{k2}(k_2, C) = E_{k1}(k_1, M)$$

  - Where C **is the ciphertext, known to the attacker,** which corresponds to the message **M, also known to the intruder**.

- The following diagram illustrates the process:



- This attacked can be extended to 3DES using 3 keys by extending the process above.

Meet-in-the-middle computational complexity

- Assuming a key length of ***b*** bits, an exhaustive key search would require $\mathbf{2^b \cdot 2^b = 2^{2b}}$ encryptions or decryptions.

- However, in the example above, if we take into account the table sort of the encrypted values and the length of the two keys then we have both time and computational complexity expressed as the sum of two products:

$$\mathbf{2^{len(k_1)} log(2^{len(k_1)}) + 2^{len(k_2)} log(2^{len(k_1)})}$$

- Where:

$2^{len(k_1)}$: creating the table with all possible values of $E_1(k_1,M)$,
$log(2^{len(k_1)})$: sorting the table with all possible values of $E_1(k_1,M)$,
$2^{len(k_2)}$: calculating all possible values of $D_2(k_2,C)$,
$log(2^{len(k_1)})$: searching the sorted table with values of $E_1(k_1,M)$

- If lengths of both keys k_1 and k_2 are the same and equal to k, then time complexity of the meet-in-the-middle attack can be presented as $O(2^{k+1})$.

- Memory usage can be approximated as $O(2^k)$.

- Time complexity of the brute force attack is much greater and equals to approximately $O(2^{2k})$. However, the brute force attack uses only O(1) memory.

Key Whitening

- Key whitening is an extremely simple technique to make block ciphers like DES much more resistant against brute-force attacks.
- In addition to the regular cipher key ***k***, two whitening keys k_1 and k_2 are used to XOR-mask the plaintext and ciphertext.
- However, it does not strengthen block ciphers against most analytical attacks such as linear and differential cryptanalysis.
- This technique is mainly applied to ciphers that are relatively strong against analytical attacks but possess too short a key space especially, for example, DES. There is also a variant of DES which uses key whitening called DESX.
- Mathematically it is defined as follows:

Encryption: $y = E_{k,k1,k2}(x) = E_k(x \oplus k_1) \oplus k_2$

Decryption: $x = D_{k,k1,k2}(y) = D_k(y \oplus k2) \oplus k1$