

Forum:[Hack Forums](#)

Thread: [Cryptograpy, conf-modes and symmetric block cipher algos](#)

SovietBastard - October 5, 2024, 1:33 pm

time ago, i wrote this kind of "documentation" on what the title says, posting this here, so anyone could read and learn a bit more about this, there might be MINOR grammar errors, but maybe not, when i wrote this, i tried to cover all possible operation modes(CBC, GCM, CTR, etc...), and symmetric block cipher algorithms.i think cryptography is such a cool topic that i could write an entire wiki page for it one day, but for now, let's keep it short and compact, i have this on one of my many GH accounts where i hold little projects and code, actually im not even sure if it will fit the max size of a post, but anyway, is shorter and more compact than what you may find online ,that's why im sharing it here, people like short things right? except women of course...

this will cover some confidentiality modes to use with symmetric key block cipher algorithms

ECB(Electronic Codeblock)
CBC(Cipher Block Chaining)
CFB(Cipher Feedback)
OFB(Output Feedback)
CTR(Counter)
GCM(Galois-Counter Mode)
EAX

Definitions, abbreviations, and symbols

Definitions and Abbreviations

Bit

a binary digit: 0 or 1

Bit Error

Substitution of a '0' bit for a '1' bit, or vice versa

Bit String

ordered sequence of '0's and '1's

Block Cipher

a set of functions and their inverse functions with a key parameter. The functions map bit strings of a fixed length to bit strings of the same length.

Block Size

number of bits in an input or output block of the block cipher.

CBC

Cipher Block Chaining Operation Mode.

CFB

Cipher Feedback operation mode.

Ciphertext

encrypted data

Confidentiality Mode

mode used to encipher and decipher blocks(CBC, ECB, CFB, OFB, CTR)

CTR

Counter operation mode.

Cryptographic Key

parameter used in the block cipher algorithm that determined the forward cipher operation and the inverse cipher operation.

Data Block

a sequence of bits with the block size length.

Data Segment

a sequence of bits whose length is a parameter that does not exceed the block size.

Decryption(deciphering)

the confidentiality mode that transforms encrypted data into original usable data.

Encryption(enciphering)

the confidentiality mode that transforms plaintext into cipher version.

ECB

electronic Codebook operation mode.

Exclusive-OR

bitwise addition of bit strings of the same length.

FIPS

Federal Information Processing Standard

Forward Cipher Function(Forward Cipher Operation)

one of the two functions of the block cipher algorithm selected by the cryptographic key.

Inverse Cipher Function(Inverse Cipher Operation)

the function that reverses the transformation of the forward cipher function if the key is equal.

Initialization Vector(IV)

data block that some modes of operation require as initial input.

Input Block

a data block that is an input to either a forward cipher function or the inverse cipher function of the block cipher algorithm.

Least Significant Bit(s)

the right most bit(s) of a bit string.

Most Significant Bit(s)

the left most bit(s) of a bit string.

Mode of Operation(Mode)

an algorithm for a cryptographic transformation of data that uses a symmetric key block cipher algorithm.

Nonce

a value that is used only once.

Octet

a group of eight binary digits.

OFB

Output Feedback operation mode.

Output Block

a data block that is an output of either a forward cipher function or the inverse cipher function of the block cipher algorithm.

Plaintext

usable data that is formatted as input to a mode.

Underlying Block Cipher Algorithm

this documentation assumes that FIPS-approved symmetric key block cipher algorithms has been chosen as the underlying algorithm. The cryptographic key(K) regulates the functioning of the block cipher algorithm, meaning that it regulates the functioning of the mode. A confidentiality mode of operation of the block cipher algorithm consists of two processes that are inverses of each other, encryption and decryption. For any given Key(K), the block cipher algorithm of the mode also consists of two functions that are inverses of each other(encryption and decryption), but as part of the choice of the block cipher algorithm one of the two functions is designed as Forward Cipher

Function(CIPH), the other is the inverse cipher function, the input and output of both functions are called input blocks and output blocks, the input and the output block cipher algorithm have the same bit length, called the block size(b).

Representation of the Plaintext and Ciphertext

For ECB and CBC modes, the total number of bits in the plaintext must be a multiple of the block size(b), the plaintext is a sequence of n (bits), each with bit length b . The bit strings in the sequence are called data blocks, and the plaintext is denoted as P , P^n , $P(n)$, etc. For the CFB Mode, the total number of bits in the plaintext must be a multiple of a parameter(s), that does not exceed the block size. The bit strings in the sequence are called data segments. For OFB and CTR Modes, the plaintext could not be a multiple of the block size. The plaintext consists of a sequence of n bit strings, in which the bit length of the last bit string is the number of bits in the last plaintext or ciphertext block, the bit strings are called data blocks.

For each mode, the encryption process transforms every plaintext data block or segment into a corresponding ciphertext data block or segment with the same bit length, so that the ciphertext is a sequence of data blocks or segments.

Initialization Vectors(IV)

the input to the encryption process of the CBC, CFB, OFB modes includes, in addition to the plaintext, a data block called the initialization vector(IV). The IV is used in an initial step in the encryption of a message and in the corresponding decryption of the message.

The IV can also not be secret, but for CBC and CFB modes, the IV must be unpredictable for each execution, and for OFB mode, IV must be unique.

Block Cipher Modes of Operation

ECB(Electronic-Codebook) Mode

... posting the rest separated due to space limit.

ECB mode is a way to encrypt data, it takes a piece of information called plaintext and turns it into a scrambled version called ciphertext using a specific key. The process is done in blocks, this means that the plaintext is divided into smaller pieces (blocks) and each block is processed separately. The formula is: $C_j = \text{CIPH}(P_j)$ for $j = 1 \dots n$. C_j is the ciphertext block for the j -th plaintext block. CIPH is the encryption function that uses a key(K) to turn the plaintext block(P_j) into a ciphertext. This means that for each block of plaintext ($P_1, P_2, \dots, P_n$), you apply the encryption function to get the corresponding ciphertext($C_1, C_2, \dots, C_n$).

The Formula for decryption is: $P_j = \text{CIPH}^{-1}(C_j)$ for $j = 1 \dots n$. C_j is the ciphertext block, and P_j is the resulting plaintext block after decryption. CIPH^{-1} is the decryption function that reverses the encryption process. This means that for each block of ciphertext, you apply the decryption function to get back the original plaintext. In ECB each block is processed independently, meaning that if two identical plaintext blocks are encrypted, they will produce the same cipher text block. This can be a security risk because patterns in the plaintext can be visible in the ciphertext. ECB is simple and fast, but not the most secure method for encryption because of the way it handles identical blocks. In ECB encryption and ECB decryption, multiple forward cipher functions and inverse cipher functions can be computed in parallel. In ECB encryption and decryption data can be processed in parallel. This means that if you have a plaintext block and encrypt it, it will always produce the same ciphertext everytime encrypted with that key(K), this can be bad for security.

Cipher Block Chaining Mode(CBC)

The CBC mode combines each block of plaintext with the previous block of ciphertext before encrypting it. In CBC mode the plaintext is divided into blocks, but before the first block gets encrypted, an IV is used to start the process. For each subsequent block of plaintext, it takes the previous ciphertext block and combine it with the current block before encrypting. This IV can also not be secret, but it must be unpredictable for each encryption.

CBC Encryption

when execution starts, the first block of plaintext is combined with a random value(IV) using XOR, this creates the first input block. Then encrypt this block to get the first ciphertext block. For the second block of plaintext, it combines it with the first ciphertext block from the previous operation using XOR to create the second ciphertext block. And so all the subsequent operations.

CBC Decryption

To decrypt the first ciphertext block, it applies the inverse of the encryption function and then XOR the result with the IV to get back the first plaintext block. For the second ciphertext block, it applies the inverse function and then XOR the result with the first ciphertext block to recover the second plaintext block. This process continues for all blocks, to recover any plaintext block(except the first), it applies the inverse function to the corresponding ciphertext block and XOR it with the previous ciphertext block. CBC Encryption cannot happen in parallel because each block depends on the result of the previous block, but in decryption, it can process multiple ciphertext block in parallel since all ciphertext blocks are available at the same time.

Cipher Feedback Mode(CFB)

CBF mode encrypts plaintext using previous ciphertext segments to encrypt the next block of plaintext, it requires an Initialization Vector(IV) to start the process. The IV can also not be secret, but should be unpredictable. CBF Mode uses a parameter called s, which determines the size of the segments(in bits) for both plaintext and ciphertext(e.g: 1-bit, 8-bit or 64-bit CFB Mode).

CFB Encryption

For encryption, the first input block is the IV. It applies the encryption function to the IV to get the first output block. The first ciphertext segment is created by XORing the first plaintext segment with the most significant bits of the output block, the remaining bits of the output block are discarded. For the next input blocks, like the second input block, it takes the least significant bits of the IV and combine them with the first ciphertext segment. This process continues, each new input block is formed by combining the previous ciphertext segment with the least significant bits of the previous input block. This process is repeated for all plaintext segments, using output from the previous step to create the next input block.

CFB Decryption

For decryption, the first input block is the IV, it applies the encryption function to the IV to get the output block. To recover the first plaintext segment, it XORs the most significant bits of the output block with the first ciphertext segment. For subsequent plaintext segments, it continues to use the previous ciphertext segments in the same way. In CBF encryption, you cannot process multiple blocks in parallel because each block depends on the previous one. In CBC decryption, you can process multiple blocks in parallel once the input blocks are constructed from the IV and ciphertext. So... CBF Mode uses previous ciphertext to encrypt the next block of plaintext, making it more secure. The IV is crucial for starting the process, encryption cannot be parallel, decryption can.

Output Feedback Mode(OFB)

the OFB Mode generates a series of output blocks from an initialization vector(IV) and uses these blocks to encrypt plaintext, it also works in reverse for decryption. OFB requires an IV, the IV must be unique for each encryption execution and not reused.

OFB Encryption

For the first output block, it starts with the IV and applies the encryption function to create the first output block(O1). Then XORs the first output block(O1) with the first plaintext block(P1) to get the first ciphertext block(C1). For next output block(O2), it uses the previous output block (O1) as input to the encryption function, then it XORs the output with the second plaintext block(P2) to get the second ciphertext block(C2). This process is repeated for all plaintext blocks, using the previous output block to generate the next one. The Last block, if a partial block, only the most significant bits are used for the XOR operation, and the rest are discarded.

OFB Decryption

For the first output block, it starts with the IV and generate the first output block(O1), XOR the output with the first ciphertext block(C1) to recover the first plaintext block(P1). For subsequent output blocks, it uses the previous output block to generate the next output block(O2) and XOR it with the second ciphertext block(C2) to recover the second plaintext block(P2). For the Last block, it uses the most significant bits of the last output block for the XOR operation, and discard the rest. In both encryption and decryption, each output block depends on the previous one so cannot process multiple blocks in parallel, however if IV is known, it can generate the output blocks before it has the plaintext or ciphertext. It is recommended to use a unique IV for every message. If the same IV is used for all messages, an attacker who knows one plaintext block, can potentially recover corresponding blocks from other messages encrypted with the same IV.

Counter Mode(CTR)

CTR Mode encrypts data with a series of unique input blocks called counters. These counter are processed to create output blocks, which are then combined with a plaintext to produce ciphertext. Each counter in the sequence must be different from all the others, not just within a single message, but across all messages encrypted with the same key, this ensures that the encryption remains secure.

CTR Encryption

For each counter(T1, T2, ..., Tn), it applies the encryption function to generate output blocks(O1, O2...). To generate ciphertext, it XORs each output block with the corresponding plaintext block to create the ciphertext blocks. For the last block, if it's a partial block, only the most significant bits are used for XOR operation, and the rest are discarded.

CTR Decryption

Like encryption, it applies encryption function to each counter to generate the output blocks. To recover plaintext, it XORs each output block with the corresponding ciphertext block to recover the plaintext blocks. For the last block, uses the most significant bits of the output block for the XOR operation, discarding the rest. In both encryption and decryption, the output blocks can be generated in parallel, allowing for processing multiple counters at the same time. It can also recover any plaintext block independently from others, as long as it has the corresponding counter. So... CTR Mode uses unique counters to generate output blocks that are XORed with plaintext to create ciphertext, it allows for parallel processing, making it efficient, each counter must be unique across all messages to maintain security.

Padding

In encryption modes like ECB, CBC, and CFB, the plaintext must be a complete set of data blocks. This means that the total number of bits in the plaintext must be a multiple of the block size. If the plaintext doesn't meet this requirement, it needs to add extra bits to make it fit, this process is called padding. A common padding method is to add a single '1' bit to the end of the plaintext, followed by enough '0' bits to fill out the last block. For example, if your plaintext ends with some bits and you need to complete a block, you would add a '1' bit and then add '0' bits until the block is full. The padding bits can be removed easily as long as the receiver knows that the message is padded. To avoid confusion, it's a good idea for the sender to always pad messages, even if the last block is already complete, this way, the receiver can tell that padding exists. Instead of always padding, you could send messages without padding if there's a reliable way for the receiver to know whether padding is present, such as including a message length indicator. So... padding is necessary to ensure that plaintext fits into complete blocks for encryption. A common method is to add '1' bit followed by '0' bits.

Generation of Counter Blocks

Each plaintext block encrypted with a given key must have a unique counter block, if the same counter block is used more than once, it can compromise the security of the encrypted data. If an attacker knows any plaintext block encrypted with a counter, they can find the output and recover other plaintexts encrypted with that same counter. To ensure that counter blocks are unique, an incrementing function generates new counter blocks from an initial counter block to ensure that they do not repeat within a single message. Also, the starting counter blocks must be chosen to ensure uniqueness across all messages encrypted with the same key. The standard incrementing function works by taking an initial counter block, the next counter blocks are created by applying an incrementing function, this function can work on the entire block or just part of it. For example, if you have a block of 8 bits and you want to increment the last 5 bits, you treat those bits as a number and add 1, wrapping around if necessary. example: with a block like "****11110", applying the incrementing function four times would give you a sequence of blocks that changes the last 5 bits. As long as the number of blocks in a message is less than or equal to 2^m , where m is the number of bits being incremented, the counter blocks will be unique within that message. One way to ensure unique counter blocks is to encrypt messages sequentially. The initial counter block for the first message can be any string of bits, for subsequent messages, apply the incrementing function to the last counter block of the previous message, this message requires keeping track of the last counter block used. Another method is to assign a unique identifier (Nonce), to each message. The Nonce is included in every counter block, and the incrementing function is applied to the remaining bits. In CTR Mode, each counter block must be unique to ensure security.

Generation of Initialization Vectors

An Initialization Vector (IV) is needed for encryption modes like CBC, CFB, and OFB along the plaintext. Each encryption requires a unique IV, which must also be used for decryption. The IV does not need to be kept secret and can be sent with the ciphertext. For CBC and CFB modes, IVs must be unpredictable, in OFB mode, the IV must be unique but does not have to be unpredictable. There are 2 main methods of generating IVs: 1 is to use a Nonce and apply the encryption function to it with the same key, method 2 is to create a random data block using a FIPS-approved random number generator. Using the same IV for multiple messages in OFB mode can compromise security. If an attacker knows the plaintext block, they can recover corresponding plaintext from other messages encrypted with the same IV. To ensure security, implementation of CBC, CFB and OFB modes should be checked to confirm that they generate IVs that are unique and unpredictable.

Error Properties

If there is a bit error in a ciphertext block, the decrypted output will be incorrect. In CFB, OFB and CTR modes, the bit errors appear in the same positions in the decrypted block as in the ciphertext.

In ECB and CBC modes, bit errors can occur in any position of the decrypted block, with about 50% of chance of error. In ECB mode, errors in one block do not affect others. In CBC mode, error in one block affects the next block's decryption, errors in the IV affect lead to incorrect decryption of the first block, with errors in the same positions as in the IV. In CFB mode, errors in a segment affect the next few segments, errors in the IV affect the first segment and possibly subsequent segments. In OFB mode, errors in one block affect all subsequent blocks, errors in the IV affect all ciphertext blocks. In CTR mode, errors in a counter block can lead to random errors in the corresponding ciphertext.

The CBC mode is particularly vulnerable to intentional bit errors in the IV, which can affect the first ciphertext block. OFB and CTR modes are vulnerable to errors in their ciphertext blocks if the integrity of those blocks is not protected. Inserting or deleting bits disrupts the synchronization of blocks, leading to errors in subsequent blocks. In 1-bit CFB mode, synchronization is restored automatically after a few bits, but for other modes, it must be fixed externally.

GCM(Galois-Counter Mode)

GCM is an authenticated encryption mode that provides both confidentiality and integrity. It combines the Counter(CTR) mode with a Galois field multiplication for authentication. GCM is efficient and allows for parallel processing.

GCM Encryption

GCM requires a nonce(unique value for each encryption) and a secret key. A counter is initialized and incremented for each block of plaintext. Each plaintext block is XORed with the output of the encryption of the counter value using the secret key. An Authentication Tag is generated using the Galois field multiplication of the encrypted data and additional authenticated data(AAD). The final authentication Tag(T) is computed based on the ciphertext and AAD.

GCM Decryption

Counter Generation --> Decryption --> Tag Verification.

The same counter values used during encryption are generated. Each ciphertext block is XORed with the output of the encryption of the counter value to recover the plaintext. The authentication tag is verified to ensure data integrity. GCM allows for parallel encryption and decryption.

EAX Mode

EAX mode is another authenticated encryption mode that combines Counter(CTR) with MAC(Message Authentication Code) for integrity. Is Designed to be simple and efficient while providing both confidentiality and integrity.

Eax Encryption

EAX requires a nonce and a secret key. A counter is initialized and incremented for each block of plaintext. Each Plaintext block encrypted is encrypted using CTR mode. A MAC(Message Authentication Code) is generated for both the plaintext and any additional authenticated data(AAD) to ensure integrity.

EAX Decryption

The same counter values used during encryption are generated. Each ciphertext block is decrypted using CTR Mode. The MAC is verified to ensure data integrity. EAX mode allows for parallel processing.

So... GCM combines CTR mode for encryption with Galois field multiplication for authentication. Allows for parallel processing. EAX combines CTR mode for encryption with a MAC for integrity. Allows for parallel processing.

[EDIT] since i noticed the formatting kind of sucks, this is the gh address for it but using better formatting and dissection: https://github.com/Somorpher/BSOM_NOTES