Block cipher modes

Mahdollinen alaotsikko

LAB-ammattikorkeakoulu

Insinööri (AMK), Tieto- ja viestintätekniikka

2023

Roope Salminen

|  |  |  |
| --- | --- | --- |
|  | | Tiivistelmä |
| Tekijä(t)  Sukunimi, Etunimi | Julkaisun laji  Opinnäytetyö, AMK | Valmistumisaika  20xx |
| Sivumäärä  xx |  |
| Työn nimi  **Opinnäytetyön otsikko**  Mahdollinen alaotsikko | | |
| Tutkinto  esim. Insinööri (AMK) | | |
| Toimeksiantajan nimi, titteli ja organisaatio  esim. Tarmo Toimeksiantaja, suunnitteluinsinööri DI, Suunnittelutoimisto Oy AB | | |
| Tiivistelmä  Tänne kirjoitetaan tiivistelmä, jossa kerrotaan …. | | |
| Asiasanat  asiasana, asiasana, asiasana | | |

|  |  |  |
| --- | --- | --- |
|  | | Abstract |
| Author(s)  Salminen, Roope | Type of Publication  Thesis, UAS | Published  2023 |
| Number of Pages  xx |  |
| Title of Publication  **Block cipher modes**  Possible subtitle(s) | | |
| Name of Degree  e.g. Engineer (UAS) | | |
| Name, title and organization of the client  e.g. Clint Client, Design Engineer M.Sc. (Eng.), Design Office Ltd. | | |
| Abstract  Type the abstract here…. | | |
| Keywords  keyword 1, keyword 2, keyword 3 | | |

Table of Contents

[1 Introduction 1](#_Toc148359638)

[2 Block ciphers 2](#_Toc148359639)

[2.1 Common general principles 2](#_Toc148359640)

[2.2 ECB – Electronic Code Book 3](#_Toc148359641)

[2.3 CBC – Cipher Block Chaining 4](#_Toc148359642)

[2.4 CTR – Counter Mode 5](#_Toc148359643)

[2.5 GCM – Galois/Counter Mode 7](#_Toc148359644)

[Comparison of modes 9](#_Toc148359645)

[3 Selected flaws and vulnerabilities 10](#_Toc148359646)

[3.1 Lack of diffusion in ECB 10](#_Toc148359647)

[3.2 Padding oracle attack in CBC 10](#_Toc148359648)

[3.3 Counter re-use in CTR 13](#_Toc148359649)

[4 Conclusions 16](#_Toc148359650)

[References 17](#_Toc148359651)

Liitteet

Liite 1. Liitteen otsikko

Liite 2. Liitteen otsikko

1. Introduction

As the number of people and devices connected via the internet grows, so does the amount of data being sent. Much of the data is not meant to be public but only accessible to smaller numbers of people. The internet itself on the other hand is meant to be accessible to everyone. The openness of the internet in a way then goes against the idea private communication. Data flowing from one destination to another can be intercepted in many points along the way by third parties who are not intended to be able to observe the data. This results in a need for ways to communicate securely over a public medium. Since the data being sent is in most cases still interceptable it needs to be sent in a form that is unreadable to unintended recipients. Numerous ways to achieve this have been thought of and tried and some of them have been found to work better than others.

The infrastructure of the internet and the way computers work and handle communications with each other pose some limitations on how secure ways to communicate can be implemented. Ciphering messages is not a new concept and one of the oldest examples is the Caesar cipher where plaintext messages were encrypted by shifting all letters of the alphabet by a fixed amount either to the right or to the left. Ciphers like this are so easy to break that they offer virtually no security in communications.

Cryptography of today has evolved enormously and often uses mathematical discoveries or undiscoveries to provide more secure ciphers. The Diffie-Hellman key exchange is a way to calculate a common shared secret from two pairs of public and private keys. At the core of the method is the assumption that there is no universally effective way to solve the discrete logarithm problem. RSA private-public key encryption on the other hand relies on the assumed difficulty of prime factoring large integers. It is noteworthy that it has not been proven that efficient algorithms for solving both problems couldn’t be discovered in the future. Both rely on the fact that no such algorithms exist today.

Here are some of the terms and abbreviations used throughout the paper.

Bit: the basis of digital data, the smallest unit of data, either 0 or 1

Byte: a collection of eight bits

XOR: exclusive or, a logical commutative operation that returns true if an odd number of its arguments are true. Here we only use two arguments, so it returns true if and only if one of them is true. The symbol used to denote the XOR operation will be .

1. Block ciphers
   1. Common general principles

A block cipher is an invertible deterministic secret-key algorithm that takes a fixed length input block and outputs a block that is the same length.[1] Deterministic here meaning that the same input together with the same key will always result in the same output and invertible meaning that for the encryption function there exists a decrypting function such that . We can write this in the form where is a plaintext block and is the secret key.

Like the name suggests, encryption and decryption are carried out by dissecting the data or message to be encrypted into smaller units called blocks which are handled by the algorithm. This is where different modes come into play as they determine how the block cipher is applied to messages whose length exceeds the block size in use. Block sizes can vary but typical sizes are 64, 128 and 256 bits.[2] Because UTF-8 encoding uses one to four bytes to represent characters, the block size dictates how many characters fit into a single block. For example, the character ‘n’ is 110 in decimal, 6E in hexadecimal and 01101110 in binary, meaning it can be represented by a single byte. If the block size was 64 bits, then eight ‘n’ characters would equal to a single block. This is also the case for most of the common letters in the English language.

In general, a 64-bit block size is nowadays considered to be too small. As a result of the birthday paradox, when the same key is used for large amounts of data, the probability of a collision happening grows too high. A collision meaning a case where two identical ciphertext block are produced. Identical outputs will in CBC mode, for example, imply that the inputs are identical, which will lead to the discovery of the result of the XOR operation of two plaintext blocks.[3] In ECB mode identical cipher blocks will always imply identical plaintext blocks. Both will make code book attacks a viable option for an attacker. As a result, most security protocols today implement larger block sizes. AES which is widely used in TLS in securing HTTPS traffic, for example, uses a 128-bit block size. One could summarize that too small of a block size results in a weak cipher and too large of a block size makes encryption and decryption cumbersome, the calculations require more time and resources. *(Lähde?)*

Key length for the algorithm is a simpler issue. As block ciphers are symmetric key encryptions, the key must be kept secret. Therefore, the most important feature of a key is that it is in practice too long to be brute-forced i.e., it will take an unreasonable amount of time and computing power to test all possible keys. AES supports key sizes of 128, 192 and 256 bits. For a 128-bit key there are possible candidates, which means it would take a huge amount of time and energy to try all the possible combinations of ones and zeroes. *(Etsi validi lähde?)* This is of course assuming that the whole possible key space is utilized by the key generating process as was not the case in the Ubuntu/Debian OpenSSL incident. *(Etsi validi lähde?)*

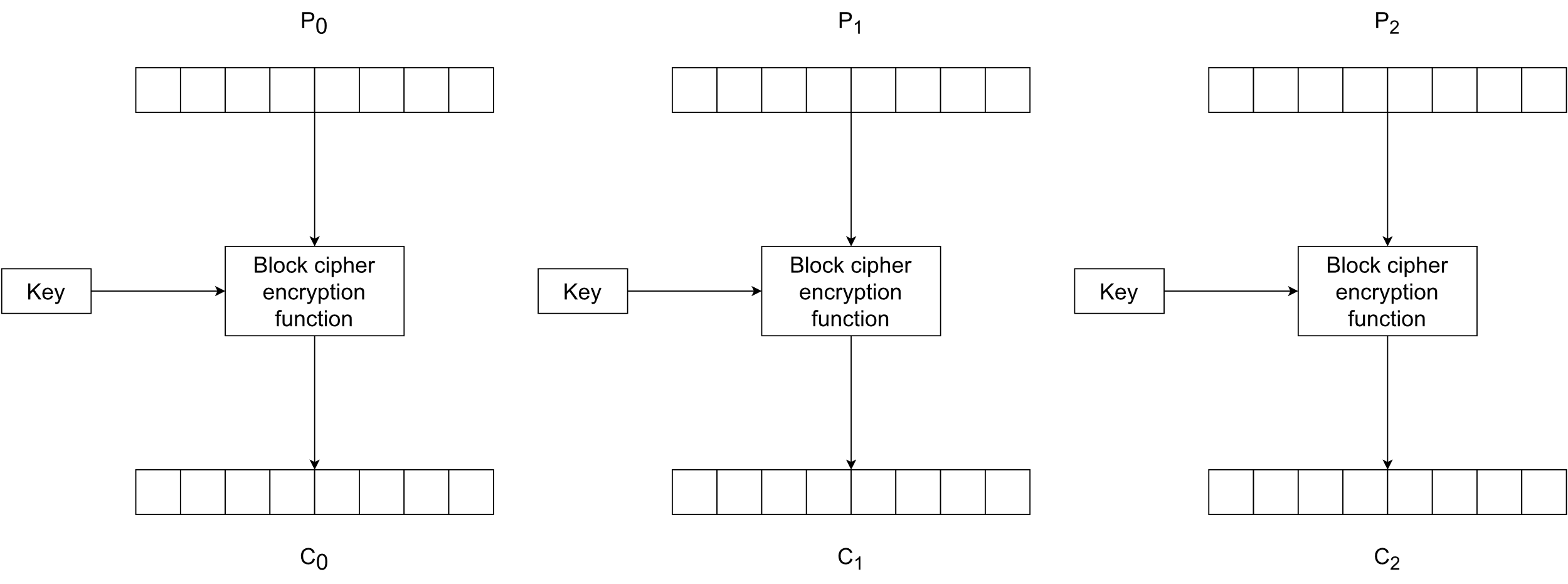
Since block cipher algorithms operate on whole blocks of data some modes require the last block of data to be padded with extra bytes to reach the block size currently in use. After decrypting the message, the extra bytes are discarded as they contain no information relevant to the original message.

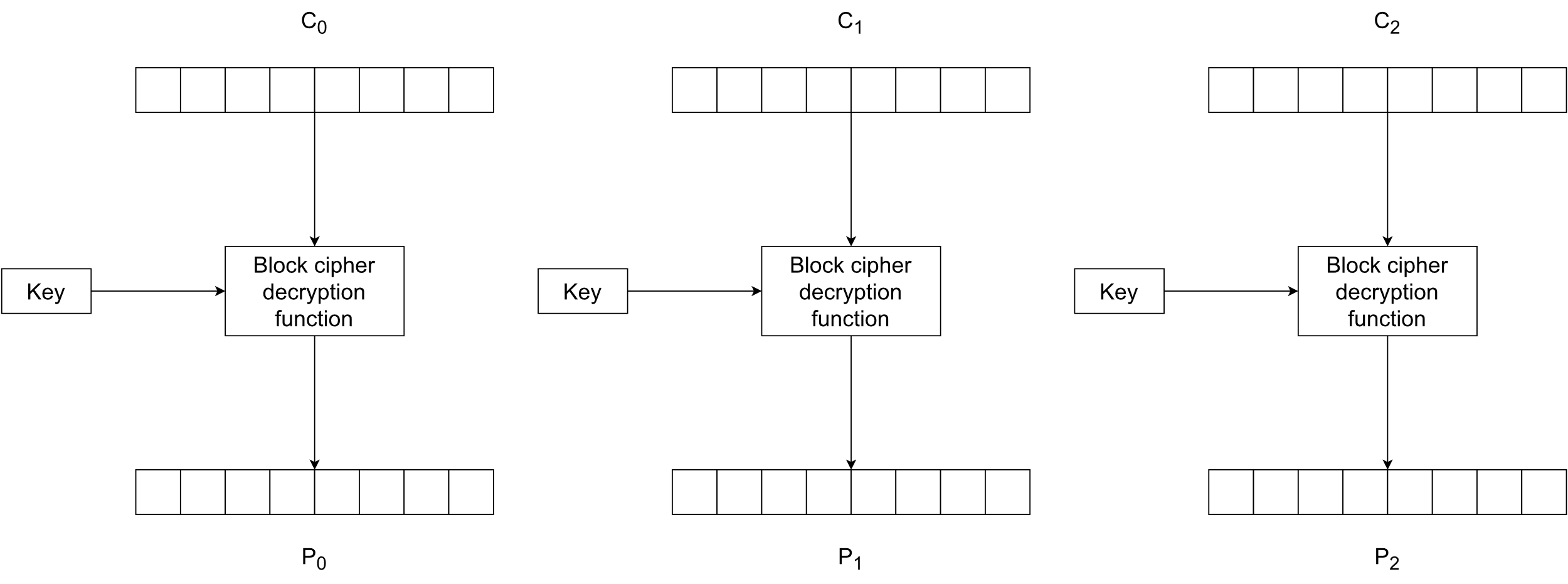
* 1. ECB – Electronic Code Book

This is the simplest of the modes and its name hints to days before digitalization where actual physical code books were used in encrypting messages. ECB is entirely deterministic and lacks diffusion, which means that all blocks are processed individually using the same key. Messages encrypted with the same key produce identical ciphertext blocks for identical plaintext blocks. This results in easy to recognize patterns in the ciphertext as messages often adhere to a standard format. Traditional letters and emails often begin with greetings and end in well-wishes. HTTP requests also follow a standard format so it can be relatively simple to start piecing together what are the plaintext counterparts of certain ciphertext blocks.

Images 1 and 2 demonstrate the principle of encrypting and decrypting a message. The message here is too big to fit into a single block so it is divided into three eight-byte blocks which concatenated together form the original message plus possible padding bytes in . The blocks are each individually run through the encryption algorithm using the same key and transformed into corresponding ciphertext blocks and which concatenated together form the ciphertext.

The decryption process is identical, only now the inputs to the decryption function are the ciphertext blocks, and of course the same key. Once the message is decrypted, the padding is removed, and the original message has now been recovered.





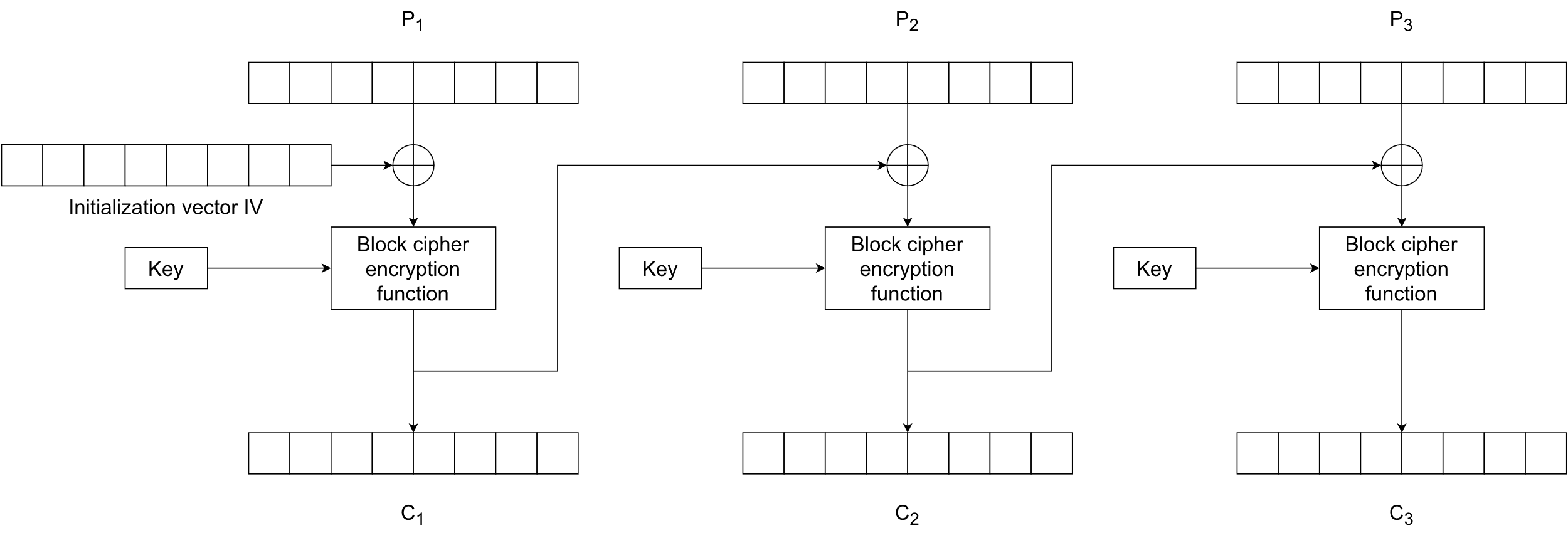
ECB is a mode that should never be used if any sort of proper secrecy is desired. Using even the most robust cipher algorithm will lead to weak encryption if combined with an unsecure mode of operation like ECB.

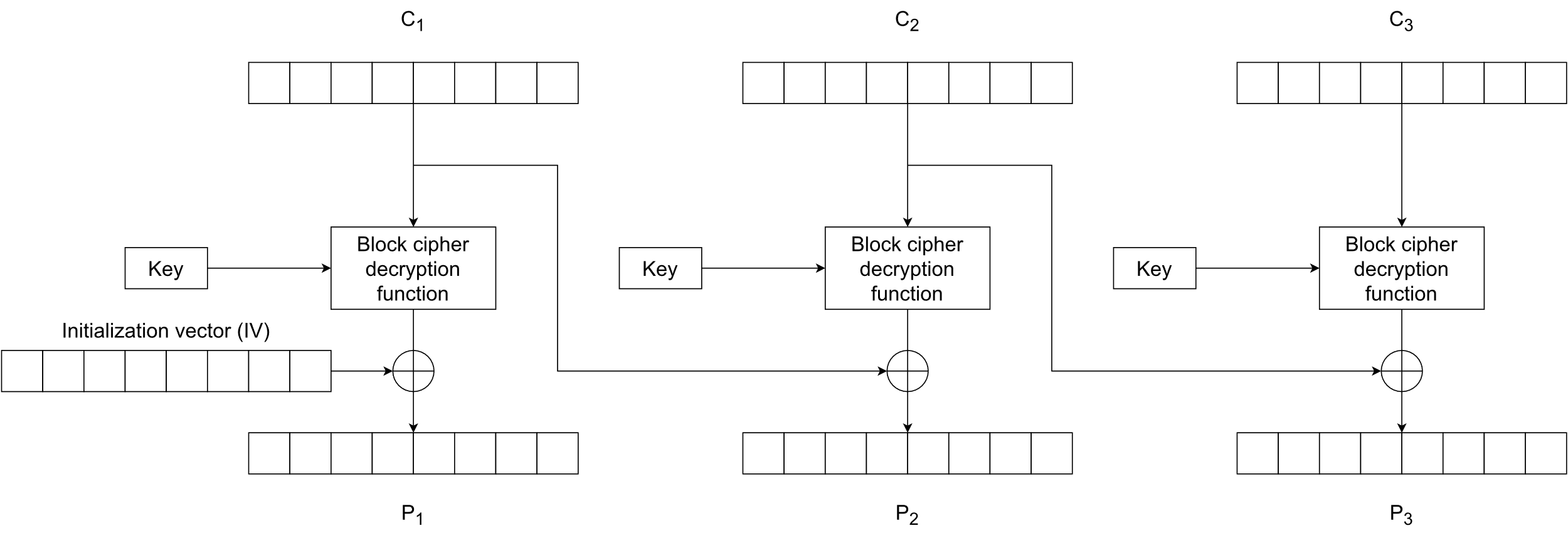
* 1. CBC – Cipher Block Chaining

In CBC each encrypted cipher block is dependent on the previous blocks. This is because before encrypting a plaintext block it is first XORed with the previous ciphertext block and the result is then encrypted. For the first plaintext block there isn’t a previous cipher block, so a random block called the initialization vector (IV) is used instead. The formula for encryption is and for decryption where . CBC also requires all of the blocks to be of the same length so the last block may contain padding bytes. Images 3 and 4 visualize the encryption and decryption processes with 8-byte blocks.

The chaining of blocks with the XOR operation causes an avalanche effect in the ciphertext so that the same message encrypted with the same key results in a totally different ciphertext if a different IV is used. Of course, re-using the same IV reduces the security of CBC to that of ECB mode so the IV has a critical role in CBC and should therefore be generated with care.

The receiver of a message encrypted with CBC mode must receive the IV as well, otherwise they won’t be able to decrypt the first ciphertext block. In general, the IV used for generating a particular ciphertext is sent along the ciphertext to the receiver. For CBC to work as intended, the IV must be random or pseudorandom to prevent possible attackers from predicting or guessing the IV for the next message and it should also be authenticated so attackers can’t tamper with it and make the first block decrypt to what they want. It doesn’t however need to be secret after the encryption process is done. [3]





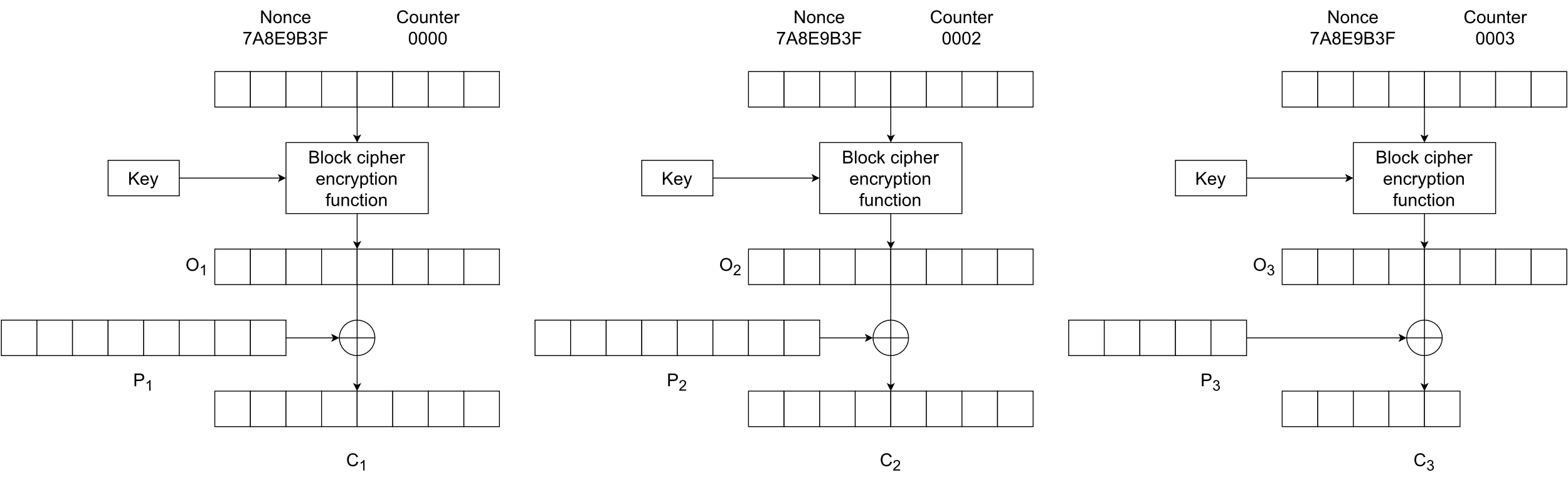
* 1. CTR – Counter Mode

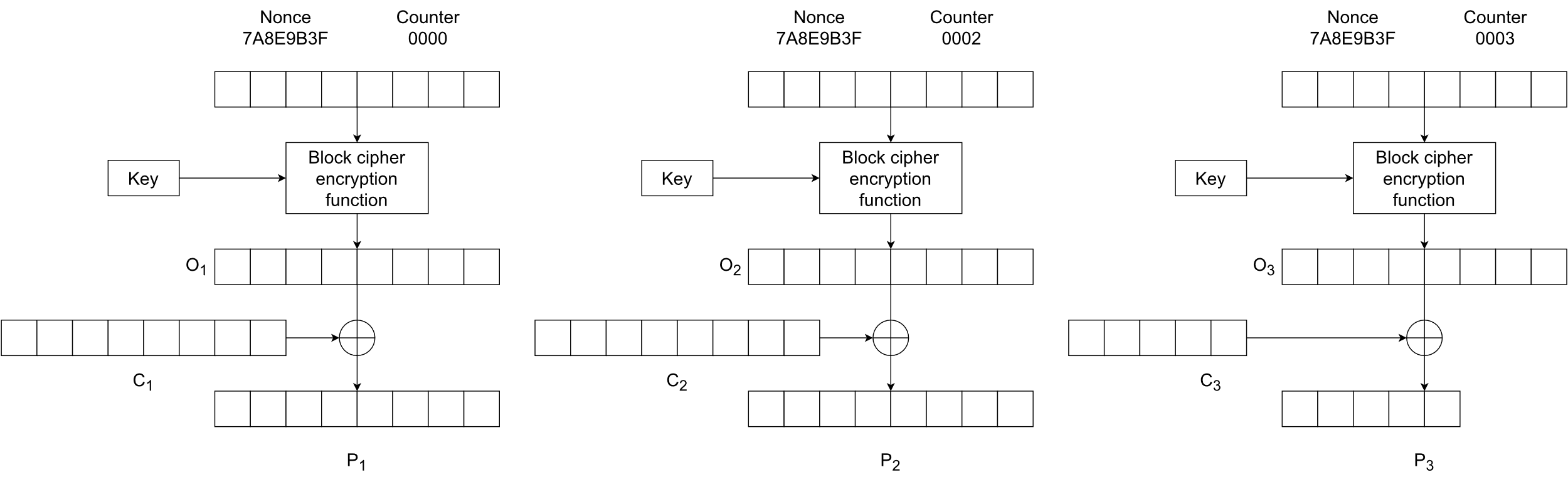
Counter mode is an older mode proposed by Whitfield Diffie and Martin Hellman in 1979. It gained interest in more recent years and was standardized by NIST in 2001. [5] It benefits from the possibility of parallel encryption and decryption since the blocks are processed independently of each other in both cases. [6]

Like CBC, CTR also needs an IV, usually called a counter, which in many cases actually consists of a nonce and a separate counter which are concatenated together, and both are bits long, where is the block length. For a given message the nonce stays the same for all blocks, while the counter is unique, and usually follows a simple increment-by-one procedure. The nonce should then be unique for all messages exchanged in a session and the counter unique for all blocks within a message. This way all nonce and counter combinations will be unique within a session. The nonce-counter combination does not need to be kept secret, only unique, and is often sent along the ciphertext block to the receiver for decryption. Like in the case with IVs in CBC mode, the purpose of the nonce-counter combination in CTR mode is to ensure identical plaintext blocks encrypt to different ciphertext blocks. The security of the cipher algorithm comes from the other input to the encryption function, namely the secret key, while the nonce-counter combination takes care of the uniqueness of the output.

Another useful feature of the CTR mode is that it does not require padding. This is because the last phase of encrypting is a XOR operation between output from the block cipher algorithm and a plaintext block. If the last plaintext block’s length is bits and it’s not the same as the current block size, only the most significant bits from the block cipher’s output are used for the XOR with the last plaintext block. This will result in the last ciphertext block to be of length also. The decryption process is almost identical to the encryption, only now the ciphertext blocks are XORed with the cipher’s output blocks. The same principal of only using the most significant bits of the last ciphertext block for the XOR applies. Images 5 and 6 illustrate the encryption and decryption processes.

Let the nonce/counter combinations be denoted by . The formula for encryption can then be written as , for and , where and denotes the length of the last plaintext block . For decryption it follows from the properties of the XOR operator that for and . Of note is that for both encryption and decryption using CTR mode only the block cipher encryption algorithm is needed. [5]





* 1. GCM – Galois/Counter Mode

Galois/Counter Mode uses encrypts messages in the same way that CTR does but adds a layer of authenticity to the encryption process. The result is a mode that combines the high speed and parallelization properties of CTR and also provides confidentiality and integrity of messages. [7]

Integrity of messages is achieved through calculating a message authentication tag for each encrypted message. The purpose of the tag is to tell the receiver if the received ciphertext was tampered with in transit or not. As it is implemented, the integrity of the message can be checked before decrypting the actual message since the tag is calculated from ciphertext blocks. This is an important security measure as it will help defend from possible chosen-ciphertext attacks.[8]

The basic idea behind the calculation of the tag is a Galois field. A field is a mathematical structure consisting of a set of at least two elements in which the binary operations of addition and multiplication are defined so that for each pair of elements and there exists exactly one and . The addition and multiplication operations must also satisfy other conditions such as commutativity and associativity, the existence of identity elements (0 and 1 in the case of real numbers and usual addition and multiplication) and inverse elements for both operations (and . (Janssen & Lindsey 2019.) Concrete examples of fields are the sets of real and rational numbers combined with the common addition and multiplication operations. Integers on the other hand do not form a field with the afore mentioned operations. The number 9, for example has no inverse element that is a part of .

A Galois field is a field i.e., satisfies all the above conditions, but also has only a finite number of members. An example of a Galois field are the integers modulus a prime number . For example, using the usual addition and multiplication operations, the set of integers modulo 7 is a Galois field and consists of the elements . (Cook.) GCM uses a Galois field of binary polynomials

Comparison of modes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ECB | CBC | CTR | GCM |
| Diffusion | No | Yes | Yes | Yes |
| Requires IV | No | Yes | Yes | Yes |
| Encryption parallelizable | Yes | No | Yes | Yes |
| Decryption parallelizable | Yes | Yes | Yes | Yes |
| Requires padding | Yes | Yes | No | No |
| Built-in authentication | No | No | No | Yes |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Table 1 describes the main properties of the described modes. Out of the four alternatives GCM is by far superior to the rest.

1. Selected flaws and vulnerabilities
   1. Lack of diffusion in ECB

A glaring flaw in ECB mode is the lack of diffusion. Identical plaintext blocks under the same key will encrypt to identical plaintext blocks, which leaves easy to see patterns in the ciphertext. A classic demonstration of this is encrypting a bitmap image with ECB mode and observing the results. In image X we see that even after encryption it is quite easy to see what the original picture was. Similar bit strings representing coloured pixels in the original image will encrypt to similar bit strings in the encrypted message as well. AES cipher user 128-bit, or 16-byte, blocks and the image is a 24-bit bitmap image which means it uses 3 bytes to represent the colour of a pixel. Because of the mismatch there is some distortion in the image, but large areas of uniform colour are still quite easily distinguished.

Using an image to demonstrate the lack of diffusion is an extreme case but patterns in messages consisting of character strings will display as well if the same phrases are repeated in the message.

* 1. Padding oracle attack in CBC

The padding in CBC can be implemented in many ways. One of the more common padding schemes is PKCS#7. This means that blocks requiring padding are padded with bytes of value equal to the number of padding bytes required. If a plaintext block is three bytes short of a full block, then the byte is appended to it three times. If a block is a single byte short of a full block a single byte is appended to it. If the last plaintext block is already a full block, a whole extra block of padding is added to the message. This makes it so that valid messages ending in a full block with the last byte being can be distinguished from ones that are padded. (Wells 2021.)

For this attack to work, the attacker needs access to what is called a padding oracle. The purpose of the oracle is to inform the attacker if a ciphertext they send for decryption decrypts to a plaintext that contains a valid padding or not. A badly configured server could for example respond with an error indicating that the padding was not valid. If an attacker has access to such an oracle and is able to intercept the ciphertext they want to read in plaintext nothing else is needed to decrypt the whole message except for the first block. (Heaton 2013.) In many real world applications the IV is not kept secret and is often sent unencrypted along the ciphertext. If an attacker can intercept the ciphertext it is likely that they will also have the IV in their possession. (Hornsby 2013.)

The basis of the attack is the XOR operation between the decrypted ciphertext block and the previous ciphertext block, which produces the corresponding plaintext block. A simple example case is two intercepted ciphertext blocks and a plaintext block corresponding to that an attacker wants to read. denotes the output of the block cipher decryption with the key and as inputs. The plaintext block in question is then . The attacker has intercepted and and when they send the concatenated message to the oracle they know if has valid padding or not. In order to recover they need to know for the XOR operation.

The attacker can use a modified version of the first ciphertext block denoted to recover byte by byte. For simplicity 8-byte blocks and indexing from 1 to 8 are used in the example. The attacker can set the bytes to zero and test all possible values for to try and find a value so that has valid padding which would generally mean that . There is a possibility that the last byte could be and still be valid padding if the second to last byte in happened to also result in by chance. This can be easily checked by altering and keeping the same.

Continuing with the assumption that will equal that Since is under the control of the attacker they now have recovered , the last byte of . Since they also are in possession of the original block they can now deduce the last byte of the original plaintext block .

The idea behind the attack can be generalized as follows: first find a value so that , where is the block size and , then calculate . From there is easy to obtain. The case , for uncovering the last byte of was covered above.

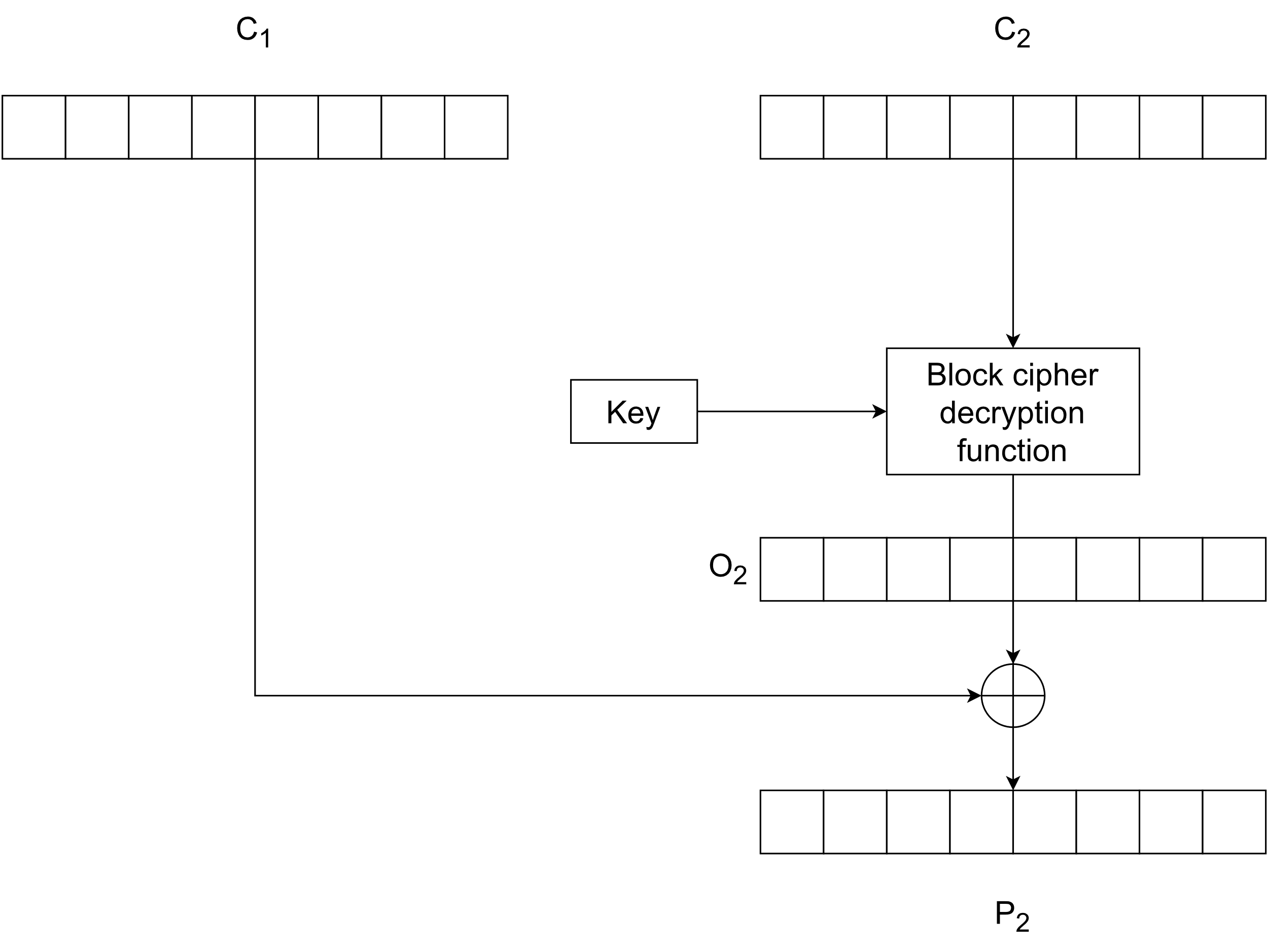
-------

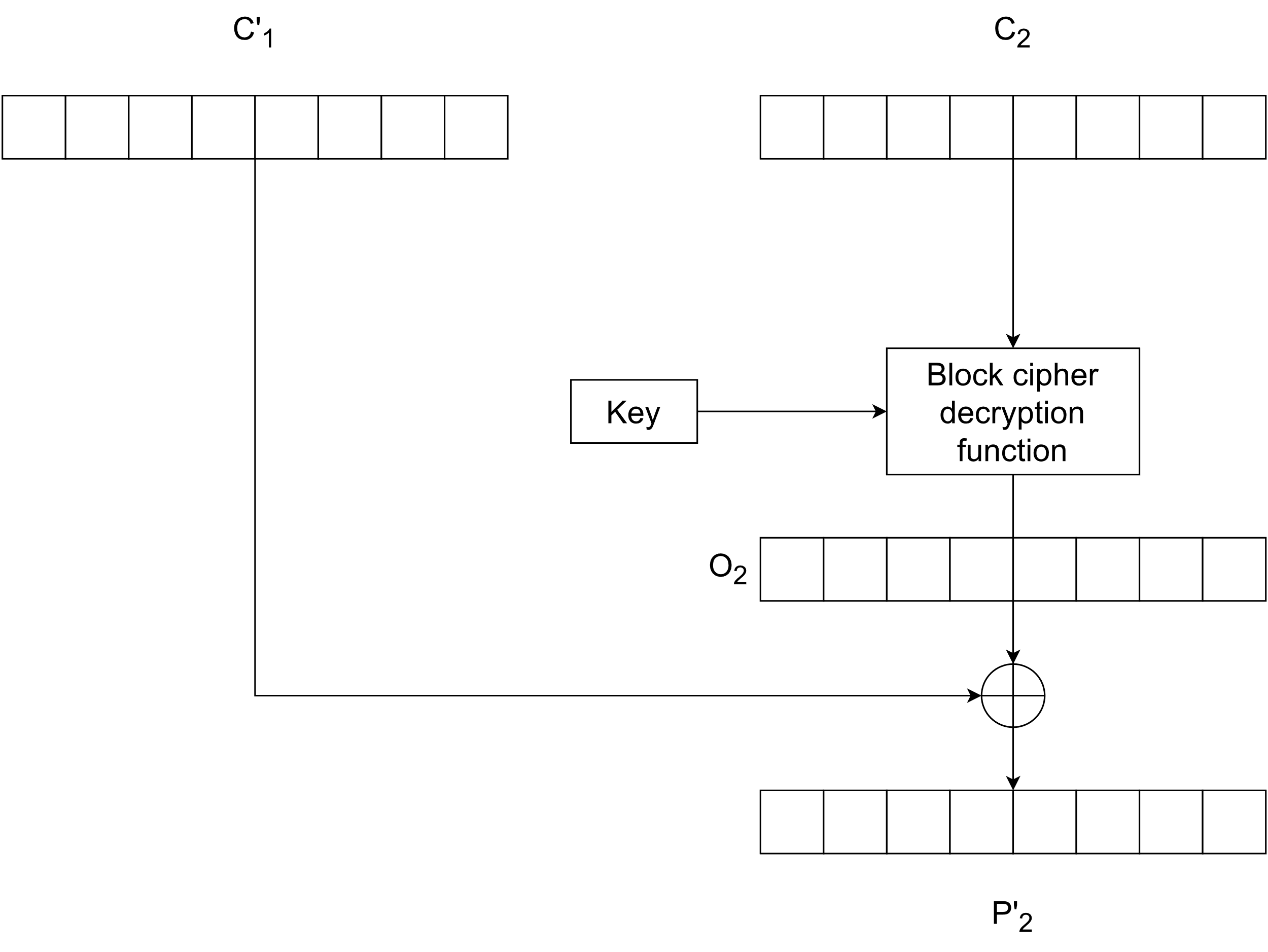
The idea behind the attack can be generalized as follows: find values so that , for all where is the block size and , then calculate . From there is easy to obtain. The case , for uncovering the last byte of was covered above.

-------

For the second to last byte the attacker first needs to set and then look for so that also, as this will result in valid PKCS#7 padding. Setting is done like so: . Now substitute in place of and insert the value of that was obtained earlier. The attacker now knows which value of will result in . Now set to zero, to what was just calculated and go through all possible values of and send to the oracle until it returns that has valid padding. This will now mean that and from here and then eventually are easily obtained.

Diagrams 7 and 8 illustrate the key points in padding oracle attacks. The block stays the same the entire time and consequently the block stays the same as well even if is replaced with to produce a bogus plaintext block . By knowing the padding scheme in use, the attacker can modify of byte by byte to obtain different versions of with valid paddings for each n and can then find out the corresponding byte in relying on the XOR operator.





* 1. Counter re-use in CTR

If an attacker has access to two different ciphertexts and they know that the encryption mode was CTR and that both messages were encrypted using the same key and counter, they can possibly uncover the plaintexts. Let and denote the messages and and denote the corresponding ciphertexts as per CTR modes encryption scheme. Now , which means the attacker is in possession of the XOR of two plaintexts.

For the simplicity of this example assume that the messages are of equal length . Let’s now denote as the concatenation of and as the concatenation of . can then be written in the form . From this form it is easier to see that if the attacker can somehow guess a part of either messages content, they can uncover the corresponding part of the other message using the XOR operation.

Taking the phrases ‘hello there’ and ‘greetings!!’ as and and converting them to hexadecimal strings, a table such as Table 2 can be constructed. The attacker can make a guess as to the content of either message, and quite luckily guesses that one of them starts with ‘hello’, denoted , which is also represented as a hexadecimal string in Table 2. They can then perform and see what the result is. Since they guessed the beginning of correctly the terms and are identical with each other and therefore cancel each other out leaving the terms in plaintext.

By decoding the resulting hexadecimal string one can see that begins with the characters ‘greet’. Now by guessing that the whole word might be ‘greetings‘ they XOR that with . The resulting hexadecimal string ‘68656c6c6f207468655344’ decodes to ‘hello theSD’. From here it is reasonable to suspect that the message could be ‘hello there’. Making that and XORing it with results in the plaintext ‘greetings!!’

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | 68 | 65 | 6c | 6c | 6f | 20 | 74 | 68 | 65 | 72 | 65 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | 67 | 72 | 65 | 65 | 74 | 69 | 6e | 67 | 73 | 21 | 21 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | 01 | 17 | 09 | 09 | 1B | 4E | 13 | 1B | 44 | 53 | 44 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| ‘hello’ |  | 68 | 65 | 6c | 6c | 6f | 00 | 00 | 00 | 00 | 00 | 00 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| ‘greet’ |  | 67 | 72 | 65 | 65 | 74 | 4E | 13 | 1B | 44 | 53 | 44 |

Table 1. First guess of plaintext contents.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | 68 | 65 | 6c | 6c | 6f | 20 | 74 | 68 | 65 | 72 | 65 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | 67 | 72 | 65 | 65 | 74 | 69 | 6e | 67 | 73 | 21 | 21 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | 01 | 17 | 09 | 09 | 1B | 4E | 13 | 1B | 44 | 53 | 44 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| ‘greetings’ |  | 67 | 72 | 65 | 65 | 74 | 69 | 6e | 67 | 73 | 00 | 00 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | 68 | 65 | 6c | 6c | 6f | 20 | 74 | 68 | 65 | 53 | 44 |

Table 2. Second guess of plaintext contents.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | 68 | 65 | 6c | 6c | 6f | 20 | 74 | 68 | 65 | 72 | 65 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | 67 | 72 | 65 | 65 | 74 | 69 | 6e1 | 67 | 73 | 21 | 21 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | 01 | 17 | 09 | 09 | 1B | 4E | 13 | 1B | 44 | 53 | 44 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| ‘hello there’ |  | 68 | 65 | 6c | 6c | 6f | 20 | 74 | 68 | 65 | 72 | 65 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | 67 | 72 | 65 | 65 | 74 | 69 | 6e | 67 | 73 | 21 | 21 |

Table 3. Third guess of plaintext contents.

1. Conclusions

References

[1] <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-107r1.pdf>

[2] <https://www.jscape.com/blog/stream-cipher-vs-block-cipher>

[3] <https://sweet32.info/>

[4] <https://defuse.ca/cbcmodeiv.htm>

[5] <https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38a.pdf>

[6] <https://www.cs.ucdavis.edu/~rogaway/papers/ctr.pdf>

[7]<https://csrc.nist.rip/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-spec.pdf>

[8] <https://www.scottbrady91.com/c-sharp/aes-gcm-dotnet>

Janssen, M., Lindsey, M. 2019. Rings with Inquiry. Referenced 14.10.2023. Available at <https://ringswithinquiry.org/rwi/SubSec-Fields.html>

Cook, J. Finite fields. Referenced 14.10.2023. Available at <https://www.johndcook.com/blog/finite-fields/>

Wells, A. 2021. Cryptography - PKCS#7 Padding Referenced 14.10.2023. Available at <https://node-security.com/posts/cryptography-pkcs-7-padding/>

Heaton, R. 2013. The Padding Oracle Attack. Referenced 14.10.2023. Available at <https://robertheaton.com/2013/07/29/padding-oracle-attack/>

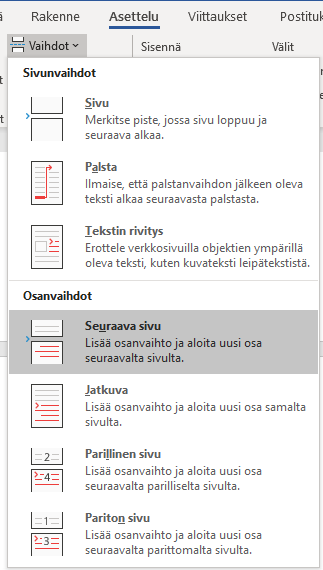
Hornsby, T. 2013. Encryption - CBC Mode IV: Secret or Not? Referenced 15.10.2023. Available at <https://defuse.ca/cbcmodeiv.htm>

Virtanen, V. 2011. Esimerkkilähteen otsikko. LAB-ammattikorkeakoulu. Viitattu 1.1.2018. Saatavissa <http://esimerkki.fi>

Liite 1. Otsikko

Jos liite on yksisivuinen, siihen ei laiteta sivunumeroa.

Seuraavalla sivulla alkaa liite, joka on useampi sivuinen – tälle sivulle tulee tehdä osan vaihto: Asettelu/Vaihdot/Seuraava sivu.

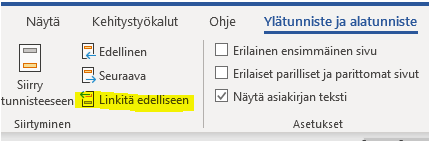


Kun piilomerkit ovat näkyvissä sivunvaihto näkyy dokumentissa seuraavasti:



Liite 2. Liitteen otsikko

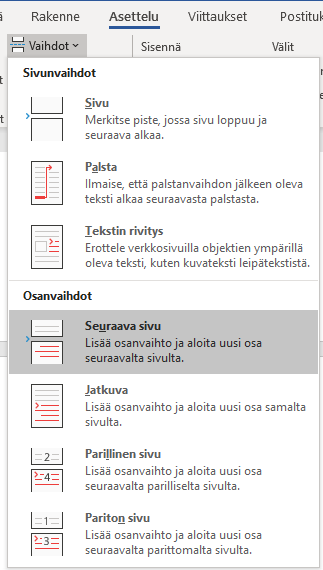
Jos liite on useampi sivuinen, sille täytyy laittaa sivunumerointi. Huom. ylätunnisteesta tulee ottaa Linkitä edelliseen pois päältä. (**Huom. edelliselle sivulle on tehty osan vaihto**)



Liite 2:n tarina jatkuu …

Liite 2:n kolmas sivu

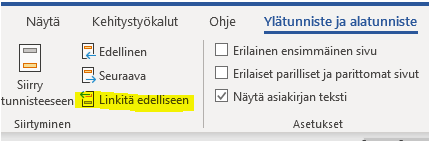
Jos tämän jälkeen tulee kolmas liite, niin tänne tulee tehdä osan vaihto. Asettelu/Vaihdot/Seuraava sivu.



Liite 3. Otsikko

Liite 3:n tarina alkaa tästä. Jos liitteessä on vain yksi sivu, sivunumeroa ei laiteta.

Tämä tarkoittaa, ylätunnisteesta otetaan pois Linkitä edelliseen



ja vasta tämän jälkeen poistetaan sivunumerointi.