# Computer and Network Security (2023-2024)

## Part 2: Symmetric encryption

**Jeroen Famaey**
jeroen.famaey@uantwerpen.be

University of Antwerp
Faculty of Science

# Part 2 overview

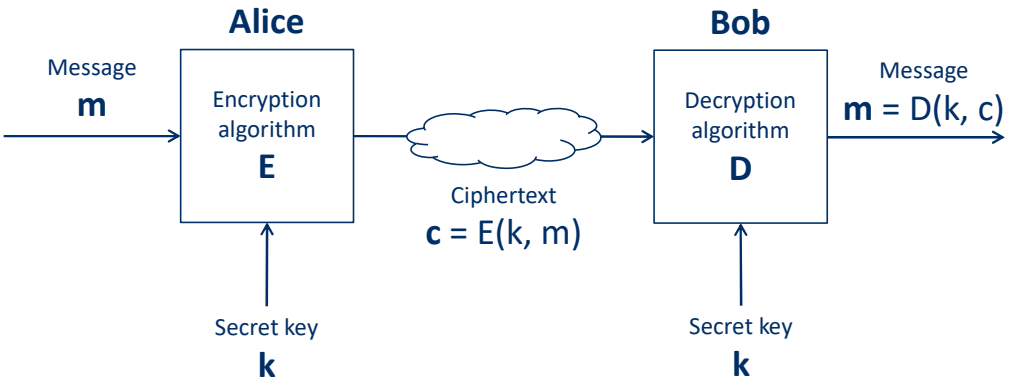
1. Symmetric encryption basics
2. Steam ciphers
   - One Time Pad
   - Practical Stream Ciphers
   - Modern Stream Cipher Examples
3. Block ciphers
   - General Block Cipher Structure
   - Advanced Encryption Standard (AES)
   - Finite Field Arithmetic
   - AES details
4. Block operation modes

University of Antwerp
Faculty of Science

# Symmetric Encryption Basics

## Ensuring data confidentiality using encryption

- Also called conventional or single-key encryption
- The only type of encryption used before the 1970s
- Still the most widely used type of encryption (DES, AES, etc.)

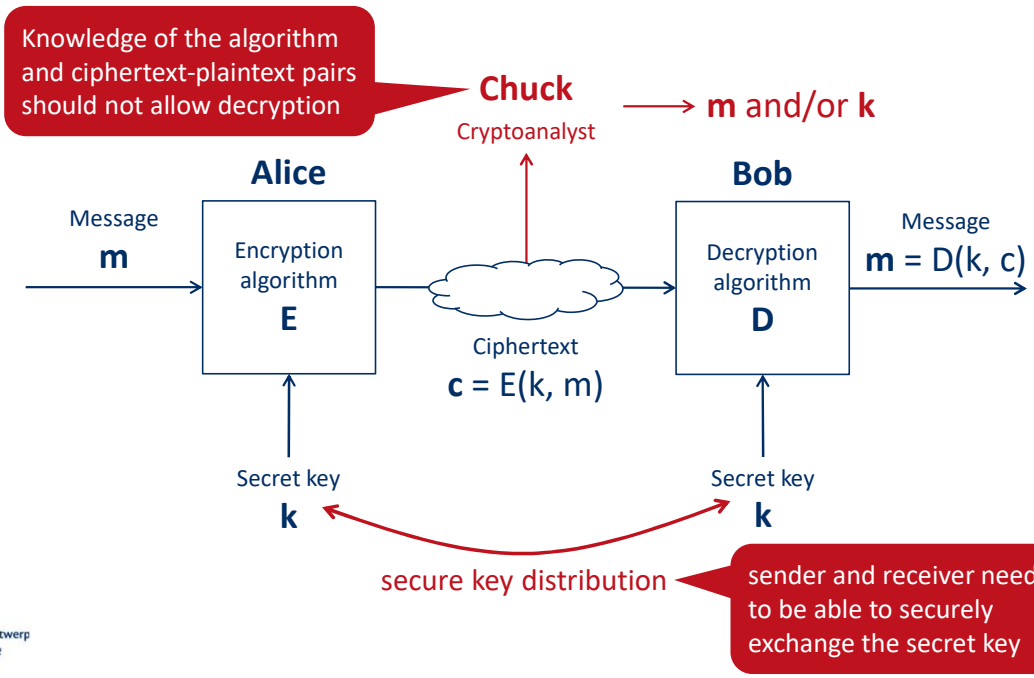



University of Antwerp
Faculty of Science



Before beginning, we define some terms. An original message is known as the **plaintext**, while the coded message is called the **ciphertext**. The process of converting from plaintext to ciphertext is known as **enciphering** or **encryption**; restoring the plaintext from the ciphertext is **deciphering** or **decryption**. The many schemes used for encryption constitute the area of study known as **cryptography**. Such a scheme is known as a **cryptographic system** or a **cipher**. Techniques used for deciphering a message without any knowledge of the enciphering details fall into the area of **cryptanalysis**. Cryptanalysis is what the layperson calls “breaking the code.” The areas of cryptography and cryptanalysis together are called **cryptology**.

## Two requirements for encryption algorithms




University of Antwerp
Faculty of Science

5

There are two requirements for secure use of conventional encryption:

1. We need a strong encryption algorithm. At a minimum, we would like the algorithm to be such that an opponent who knows the algorithm and has access to one or more ciphertexts would be unable to decipher the ciphertext or figure out the key. This requirement is usually stated in a stronger form: The opponent should be unable to decrypt ciphertext or discover the key even if he or she is in possession of a number of ciphertexts together with the plaintext that produced each ciphertext.
2. Sender and receiver must have obtained copies of the secret key in a secure fashion and must keep the key secure. If someone can discover the key and knows the algorithm, all communication using this key is readable.

## Mathematical notation

- Symmetric encryption is defined over a triple $(\boldsymbol{K}, \boldsymbol{M}, \boldsymbol{C})$:
  - Key space $\boldsymbol{K}$ contains the set of all keys
  - Message space $\boldsymbol{M}$ contains the set of all message
  - Ciphertext space $\boldsymbol{C}$ contains the set of all ciphertexts
- It consists of a set of "efficient" algorithms $(\boldsymbol{E}, \boldsymbol{D})$
  - $\boldsymbol{E}: \boldsymbol{K} \times \boldsymbol{M} \rightarrow \boldsymbol{C}$
  - $\boldsymbol{D}: \boldsymbol{K} \times \boldsymbol{C} \rightarrow \boldsymbol{M}$
- Subject to correctness property:
  - $\forall m \in \boldsymbol{M}, k \in \boldsymbol{K}: \boldsymbol{D}\big(k, \boldsymbol{E}(k, m)\big) = m$
- "Efficient" may refer to theoretical (e.g., polynomial time complexity) or practical (e.g., a maximum execution time) efficiency.

University of Antwerp
Faculty of Science



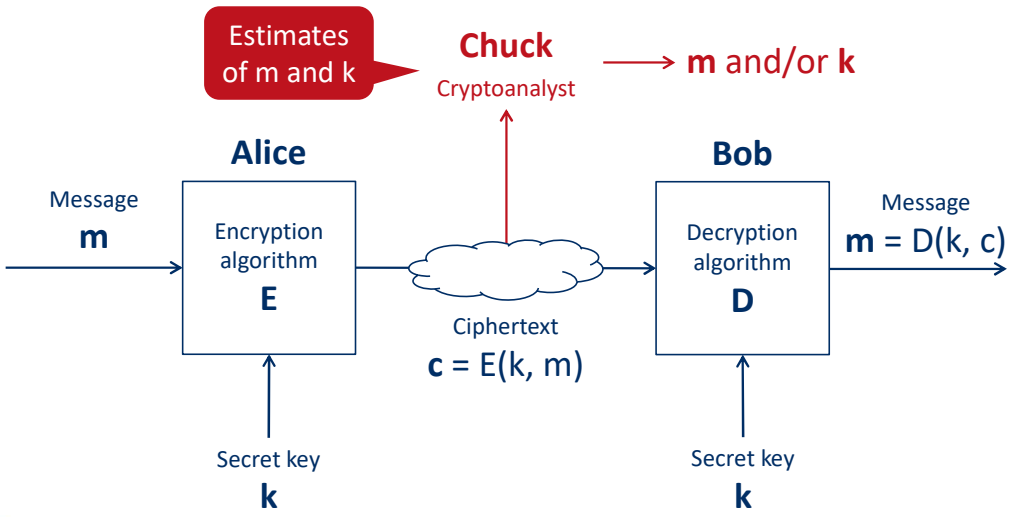
A symmetric cipher is a pair (E, D) of functions that is defined over (K, M, C):

- The function E (the encryption function) takes as input a key k and a message m (also called a plaintext), and produces as output a ciphertext c. That is, c = E(k,m), and we say that c is the encryption of m under k.
- The function D (the decryption function) takes as input a key k and a ciphertext c, and produces a message m. That is, m = D(k, c), and we say that m is the decryption of c under k.
- We require that decryption "undoes" encryption; that is, the cipher must satisfy the following correctness property: for all keys k and all messages m, we have D(k, E(k, m) ) = m.

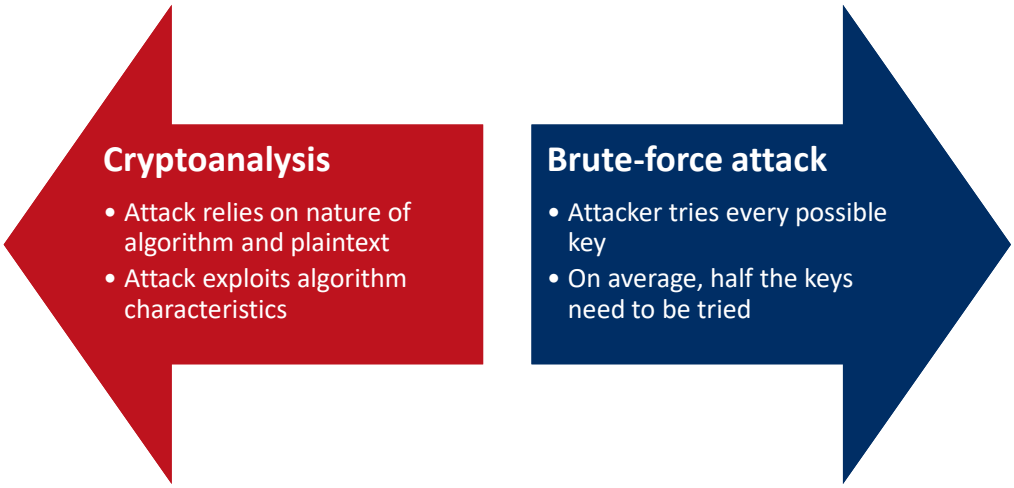
To be slightly more formal, let us assume that K is the set of all keys (the key space), M is the set of all messages (the message space), and that C is the set of all ciphertexts (the ciphertext space). With this notation, we can write: E : K x M → C, and D : K x C → M.

## Cracking the code

The goal is to recover the plaintext and/or key based on knowledge of c, a set of (m, c) pairs, and/or E



## Two categories of cryptography attacks




University of Antwerp
Faculty of Science

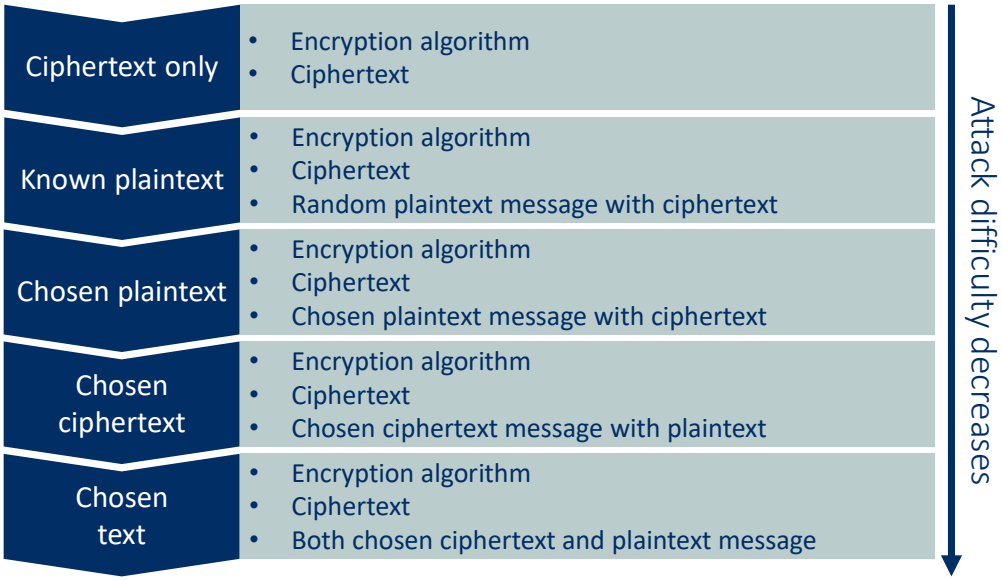
8

Typically, the objective of attacking an encryption system is to recover the key in use rather than simply to recover the plaintext of a single ciphertext. There are two general approaches to attacking a conventional encryption scheme:

- **Cryptanalysis:** Cryptanalytic attacks rely on the nature of the algorithm plus perhaps some knowledge of the general characteristics of the plaintext or even some sample plaintext–ciphertext pairs. This type of attack exploits the characteristics of the algorithm to attempt to deduce a specific plaintext or to deduce the key being used.
- **Brute-force attack:** The attacker tries every possible key on a piece of cipher- text until an intelligible translation into plaintext is obtained. On average, half of all possible keys must be tried to achieve success.

If either type of attack succeeds in deducing the key, the effect is catastrophic: All future and past messages encrypted with that key are compromised.

## Attacker knowledge




University of Antwerp | Faculty of Science

9

This slide summarizes the various types of **cryptanalytic attacks** based on the amount of information known to the cryptanalyst. The most difficult problem is presented when all that is available is the *ciphertext only*. In some cases, not even the encryption algorithm is known, but in general, we can assume that the opponent does know the algorithm used for encryption. One possible attack under these circumstances is the brute-force approach of trying all possible keys. If the key space is very large, this becomes impractical. Thus, the opponent must rely on an analysis of the ciphertext itself, generally applying various statistical tests to it. To use this approach, the opponent must have some general idea of the type of plaintext that is concealed, such as English or French text, an EXE file, a Java source listing, an accounting file, and so on.
The ciphertext-only attack is the easiest to defend against because the opponent has the least amount of information to work with. In many cases, however, the analyst has more information. The analyst may be able to capture one or more plaintext messages as well as their encryptions. Or the analyst may know that certain plaintext patterns will appear in a message. For example, a file that is encoded in the Postscript format always begins with the same pattern, or there may be a standardized header or banner to an electronic funds transfer message, and so on. All these are examples of *known plaintext*. With this knowledge, the analyst may be able to deduce the key on the basis of the way in which the known plaintext is transformed.

Closely related to the known-plaintext attack is what might be referred to as a probable-word attack. If the opponent is working with the encryption of some general prose message, he or she may have little knowledge of what is in the message. However, if the opponent is after some very specific information, then parts of the message may be

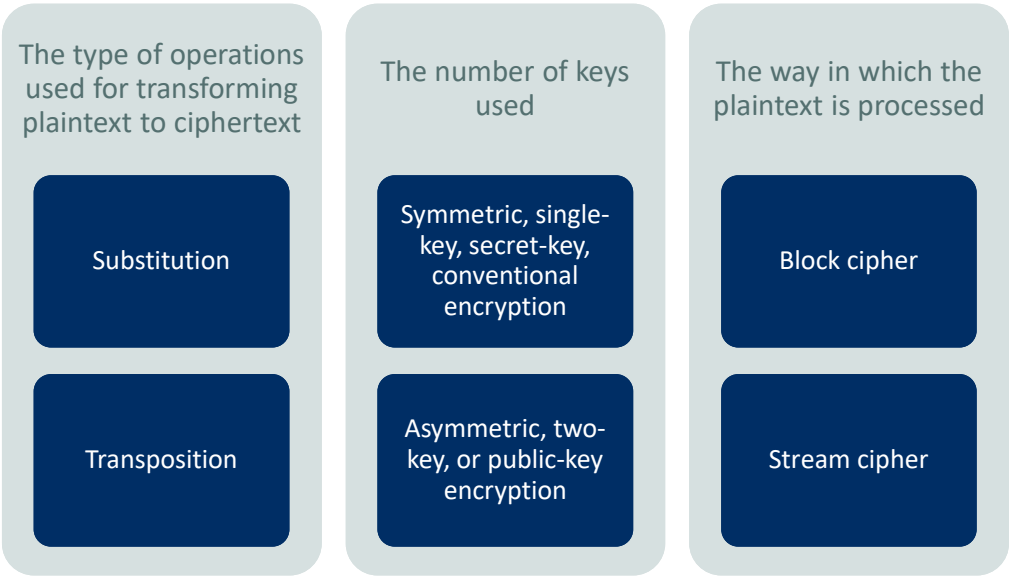
known. For example, if an entire accounting file is being transmit- ted, the opponent may know the placement of certain key words in the header of the file. As another example, the source code for a program developed by Corporation X might include a copyright statement in some standardized position.

If the analyst is able somehow to get the source system to insert into the system a message chosen by the analyst, then a *chosen-plaintext* attack is possible. An example of this strategy is differential cryptanalysis. In general, if the analyst is able to choose the messages to encrypt, the analyst may deliberately pick patterns that can be expected to reveal the structure of the key.

The slide lists two other types of attack: chosen ciphertext and chosen text. These are less commonly employed as cryptanalytic techniques but are nevertheless possible avenues of attack.

Only relatively weak algorithms fail to withstand a ciphertext-only attack. Generally, an encryption algorithm is designed to withstand a known-plaintext attack.

## Characterization of cryptographic systems




University of Antwerp | Faculty of Science

10

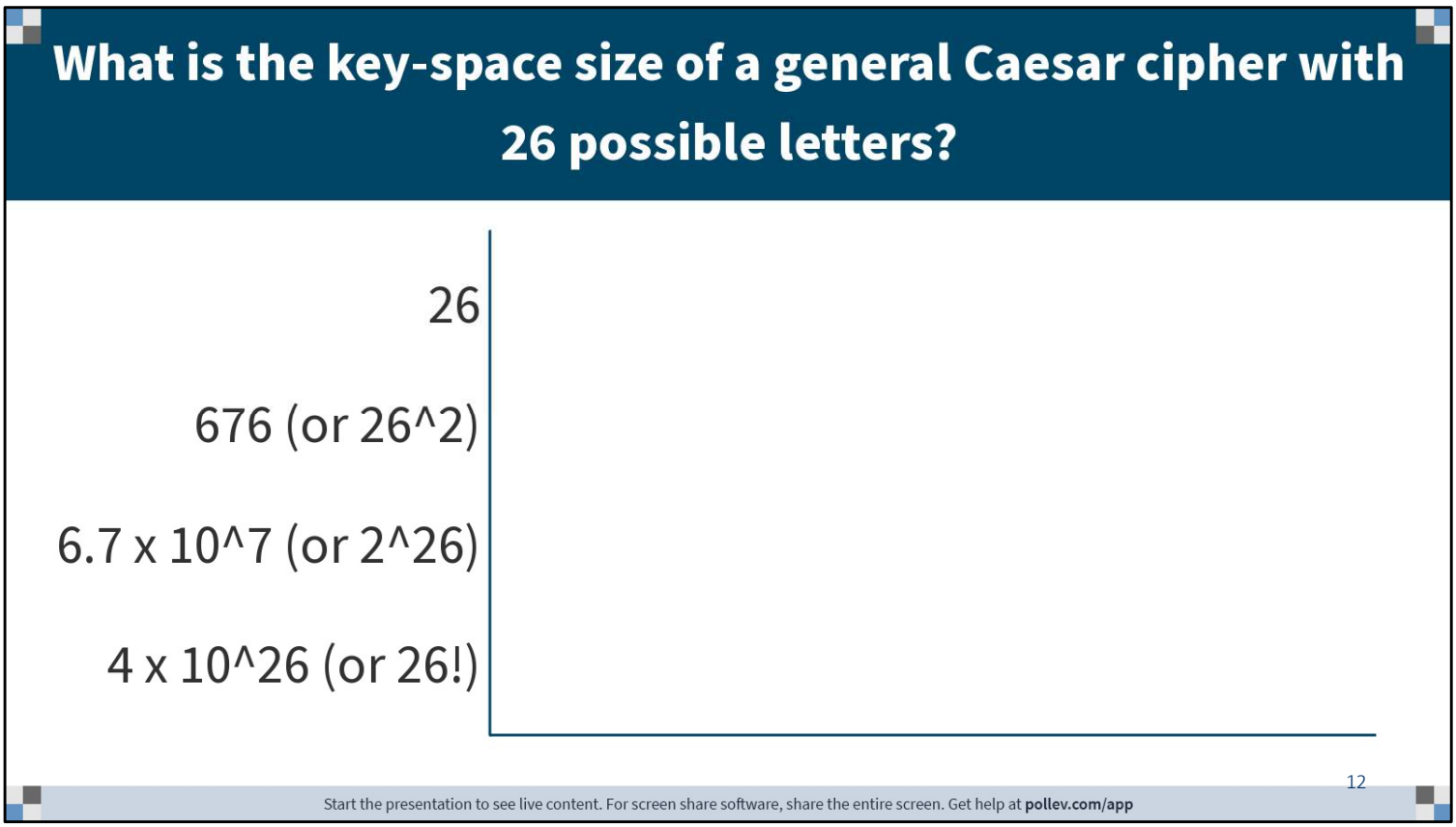
Cryptographic systems are characterized along three independent dimensions:

1. The type of operations used for transforming plaintext to ciphertext. All encryption algorithms are based on two general principles: substitution, in which each element in the plaintext (bit, letter, group of bits or letters) is mapped into another element, and transposition, in which elements in the plaintext are rearranged. The fundamental requirement is that no information be lost (i.e., that all operations are reversible). Most systems, referred to as product systems , involve multiple stages of substitutions and transpositions.
2. The number of keys used. If both sender and receiver use the same key, the system is referred to as symmetric, single-key, secret-key, or conventional encryption. If the sender and receiver use different keys, the system is referred to as asymmetric, two-key, or public-key encryption.
3. The way in which the plaintext is processed. A block cipher processes the input one block of elements at a time, producing an output block for each input block. A stream cipher processes the input elements continuously, producing output one element at a time, as it goes along.

## Some history: the Caesar cipher

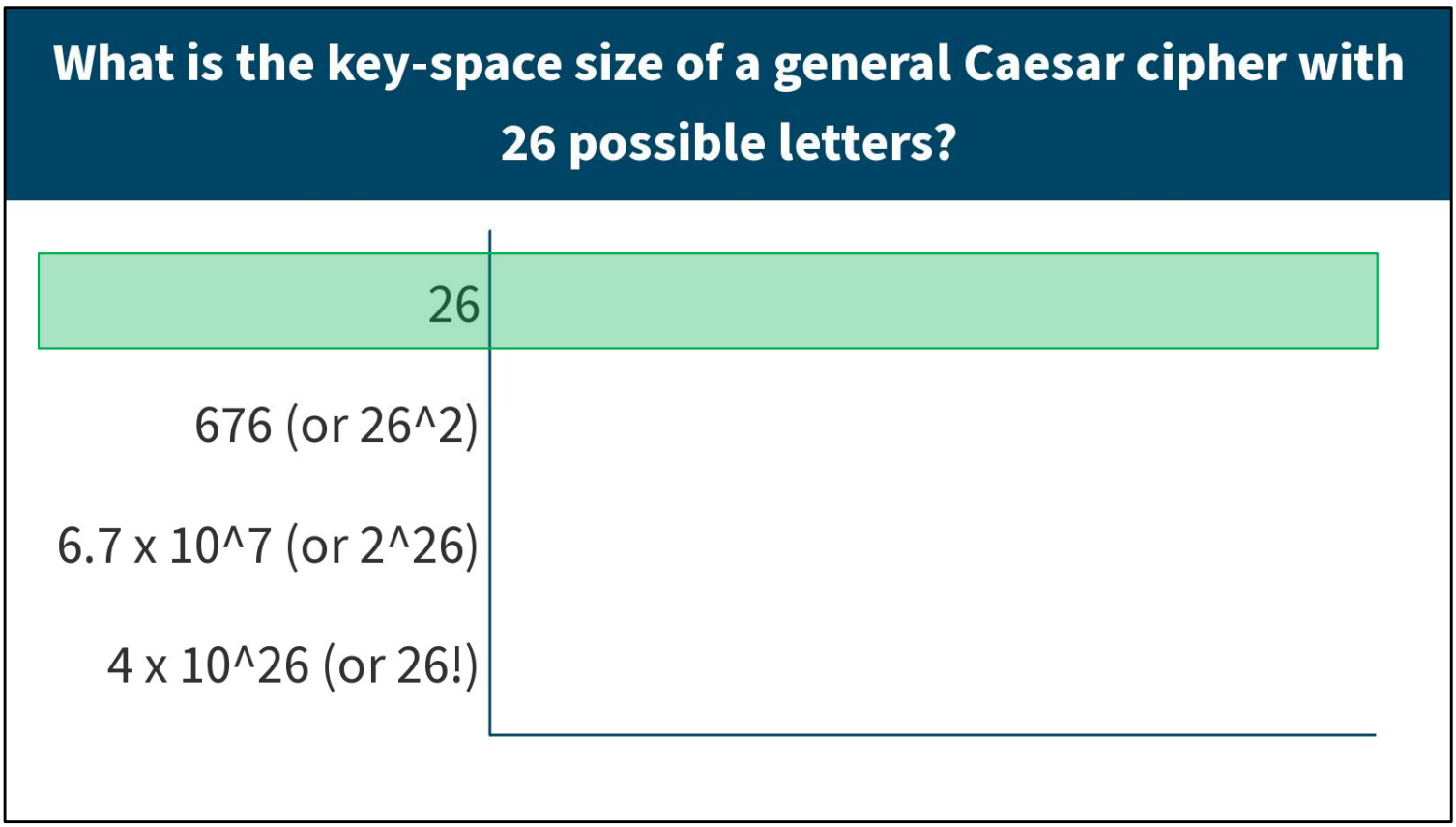
Julius Caesar
(100BC - 44BC)

- Replace each letter in the alphabet with the letter 3 places further (round robin)
- **Mapping**
  - plain: a b c d e f g h i j k l m n o p q r s t u v w x y z
  - cipher: D E F G H I J K L M N O P Q R S T U V W X Y Z A B C
- **Mathematical representation**
  - Assign each sorted letter a number in the range *[0, 25]*
  - $c = E(3, p) = (p + 3) \bmod 26$ $p = D(3, c) = (c - 3) \bmod 26$
- **General Caesar algorithm**
  - $c = E(k, p) = (p + k) \bmod N$ with $N$ letters and $k \in [0, N-1]$




Poll Title: Do not modify the notes in this section to avoid tampering with the Poll Everywhere activity.
More info at polleverywhere.com/support

What is the key-space size of a general Caesar cipher with 26 possible letters?
https://www.polleverywhere.com/multiple_choice_polls/5y2tJUCbpk5RYoS



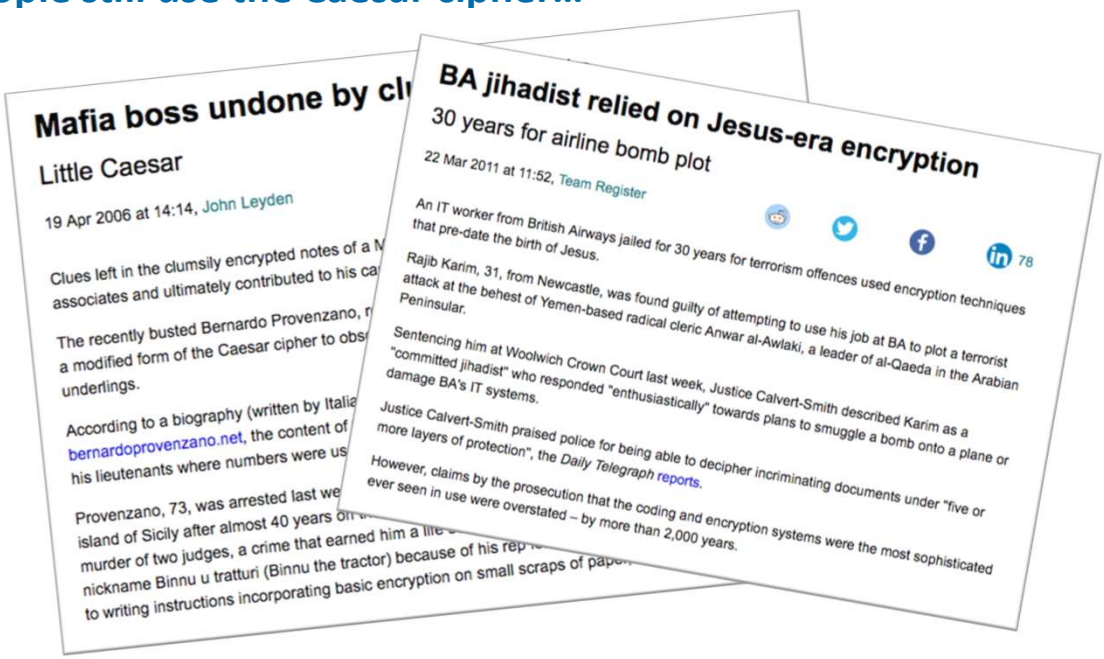

Poll Title: What is the key-space size of a general Caesar cipher with 26 possible letters?

# What makes it easy to brute force Caesar ciphers?

| KEY | PHHW PH DIWHU WKH WRJD SDUWB |
|---|---|
| 1 | oggv og chvgt vjg vqic rctva |
| 2 | nffu nf bgufs uif uphb gbsuz |
| 3 | meet me after the toga party |
| 4 | ldds ld zesdq sgd snfz ozqsx |
| 5 | kccr kc ydrcp rfc rmey nyprw |
| 6 | jbbq jb xcqbo qeb qldx mxoqv |
| 7 | iaap ia wbpan pda pkcw lwnpu |
| 8 | hzzo hz vaozm ocz ojbv kvmot |
| 9 | gyyn gy uznyl nby niau julns |
| 10 | fxxm fx tymxk max mhzt itkmr |
| 11 | ewwl ew sxlwj lzw lgys hsjlq |
| 12 | dvvk dv rwkvi kyv kfxr grikp |
| 13 | cuuj cu qvjuh jxu jewq fqhjo |
| 14 | btti bt puitg iwt idvp epgin |
| 15 | assh as othsf hvs hcuo dofhm |
| 16 | zrrg zr nsgre gur gbtn cnegl |
| 17 | yqqf yq mrfqd ftq fasm bmdfk |
| 18 | xppe xp lqepc esp ezrl alcej |
| 19 | wood wo kpdob dro dyqk zkbdi |
| 20 | vnnc vn jocna cqn cxpj yjach |
| 21 | ummb um inbmz bpm bwoi xizbg |
| 22 | tlla tl hmaly aol avnh whyaf |
| 23 | skkz sk glzkx znk zumg vgxze |
| 24 | rjjy rj fkyjw ymj ytlf ufwyd |
| 25 | qiix qi ejxiv xli xske tevxc |

1. The algorithm is known
2. There are only 26 keys (or 25 if you exclude the trivial 0 key)
3. The plaintext language is easily recognized

## Yet people still use the Caesar cipher…



**Mafia boss undone by cl**

Little Caesar

19 Apr 2006 at 14:14, John Leyden

Clues left in the clumsily encrypted notes of a M
associates and ultimately contributed to his ca

The recently busted Bernardo Provenzano, r
a modified form of the Caesar cipher to obs
underlings.

According to a biography (written by Italia
bernardoprovenzano.net, the content of
his lieutenants where numbers were us

Provenzano, 73, was arrested last we
island of Sicily after almost 40 years on
murder of two judges, a crime that earned him a life
nickname Binnu u tratturi (Binnu the tractor) because of his rep
to writing instructions incorporating basic encryption on small scraps of pape

**BA jihadist relied on Jesus-era encryption**

30 years for airline bomb plot

22 Mar 2011 at 11:52, Team Register

78

An IT worker from British Airways jailed for 30 years for terrorism offences used encryption techniques that pre-date the birth of Jesus.

Rajib Karim, 31, from Newcastle, was found guilty of attempting to use his job at BA to plot a terrorist attack at the behest of Yemen-based radical cleric Anwar al-Awlaki, a leader of al-Qaeda in the Arabian Peninsular.

Sentencing him at Woolwich Crown Court last week, Justice Calvert-Smith described Karim as a "committed jihadist" who responded "enthusiastically" towards plans to smuggle a bomb onto a plane or damage BA's IT systems.

Justice Calvert-Smith praised police for being able to decipher incriminating documents under "five or more layers of protection", the *Daily Telegraph* reports.

However, claims by the prosecution that the coding and encryption systems were the most sophisticated ever seen in use were overstated – by more than 2,000 years.

## General substitution (or monoalphabetic) cipher

- Increases the key-space size compared to the Caesar cipher
- Instead of shifting the letters k spots, each plain letter is assigned a unique random letter from the alphabet

- Example substitution cipher
  - Plain: a b c d e f g h i j k l m n o p q r s t u v w x y z
  - Cipher: D H U J E I X K M Q N T F Y O B P G A R L Z S V C W

- Encryption example
  - Message: hello my name is jeroen
  - Ciphertext: KETTO FC YDFE MA QEGOEY

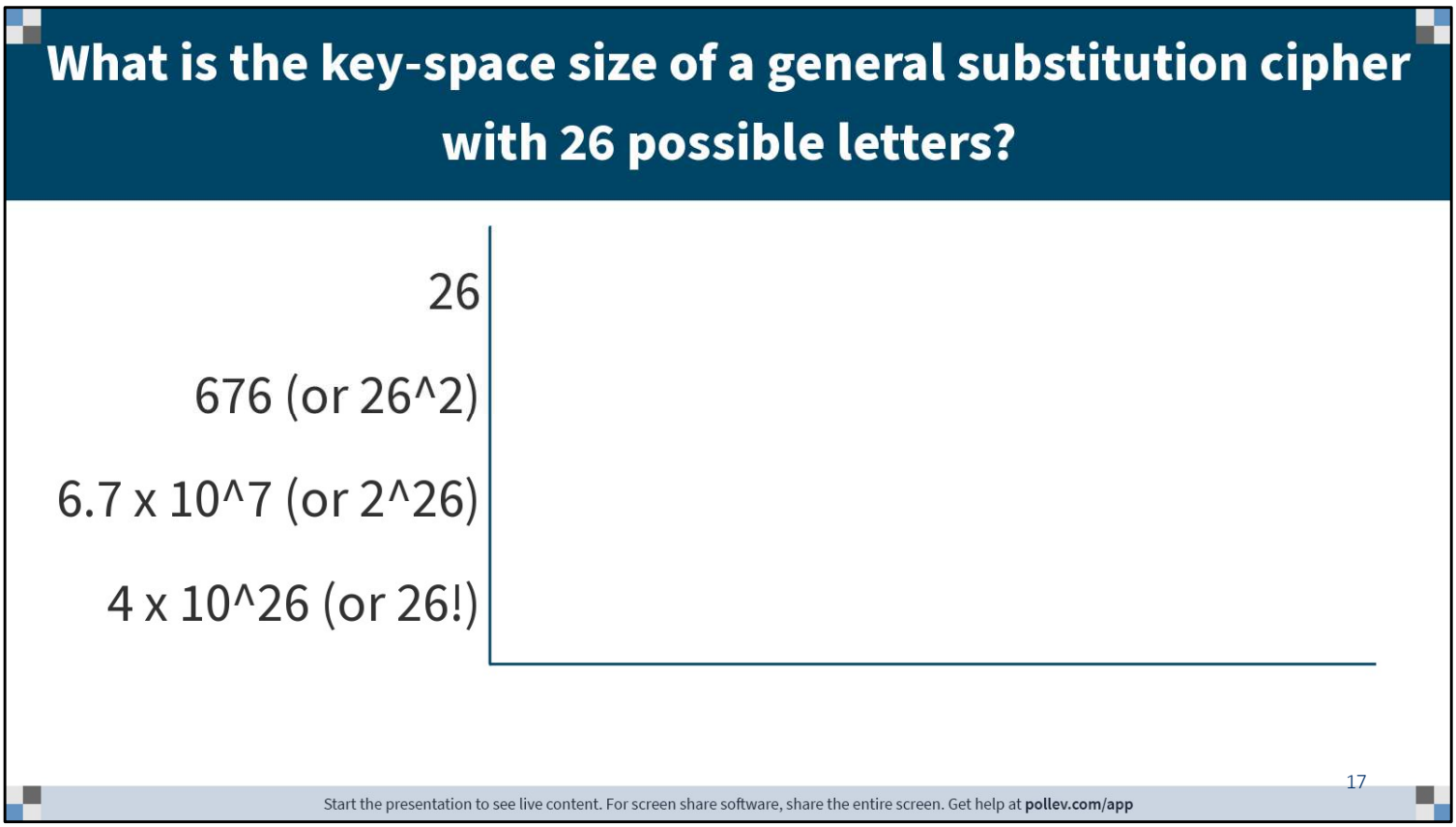


With only 25 possible keys, the Caesar cipher is far from secure. A dramatic increase in the key space can be achieved by allowing an arbitrary substitution. Before proceeding, we define the term permutation . A permutation of a finite set of elements S is an ordered sequence of all the elements of S , with each element appearing exactly once.

For example, if S = {a, b, c}, there are six permutations of S : abc, acb, bac, bca, cab, cba

In general, there are n ! permutations of a set of n elements, because the first element can be chosen in one of n ways, the second in n - 1 ways, the third in n - 2 ways, and so on.

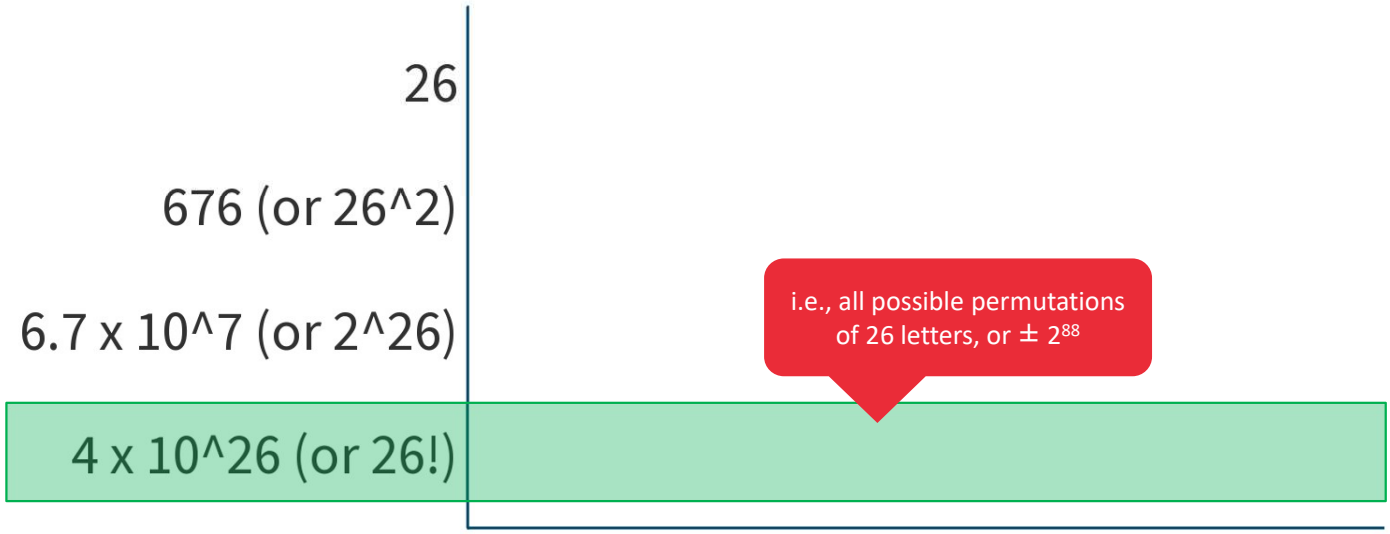
If, instead, the “cipher” line can be any permutation of the 26 alphabetic characters, then there are 26! or greater than 4 * 1026 possible keys. This is 10 orders of magnitude greater than the key space for DES and would seem to eliminate brute-force techniques for cryptanalysis. Such an approach is referred to as a monoalphabetic substitution cipher , because a single cipher alphabet (mapping from plain alphabet to cipher alphabet) is used per message.




Poll Title: Do not modify the notes in this section to avoid tampering with the Poll Everywhere activity.
More info at polleverywhere.com/support

What is the key-space size of a general substitution cipher with 26 possible letters?
https://www.polleverywhere.com/multiple_choice_polls/XJ0QX8HlSaltneo

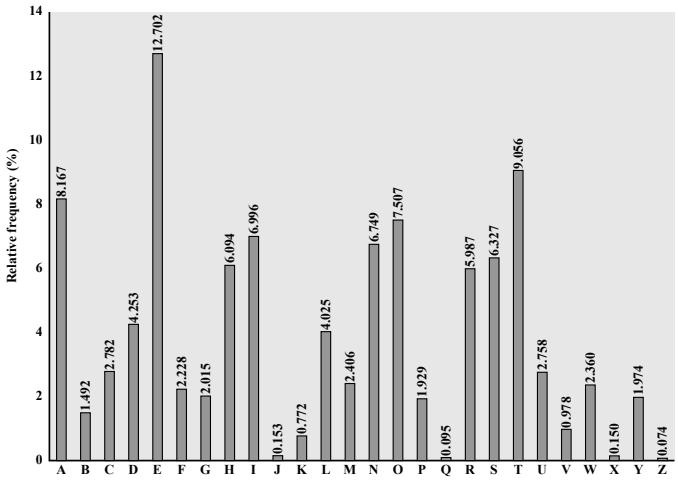
## What is the key-space size of a general substitution cipher with 26 possible letters?




Poll Title: What is the key-space size of a general substitution cipher with 26 possible letters?

## Breaking the substitution cipher

The relative frequency of the letters can be used to map cipher letters onto one or a few potential plain letters






There is, however, another line of attack. If the cryptanalyst knows the nature of the plaintext (e.g., noncompressed English text), then the analyst can exploit the regularities of the language.

# An example

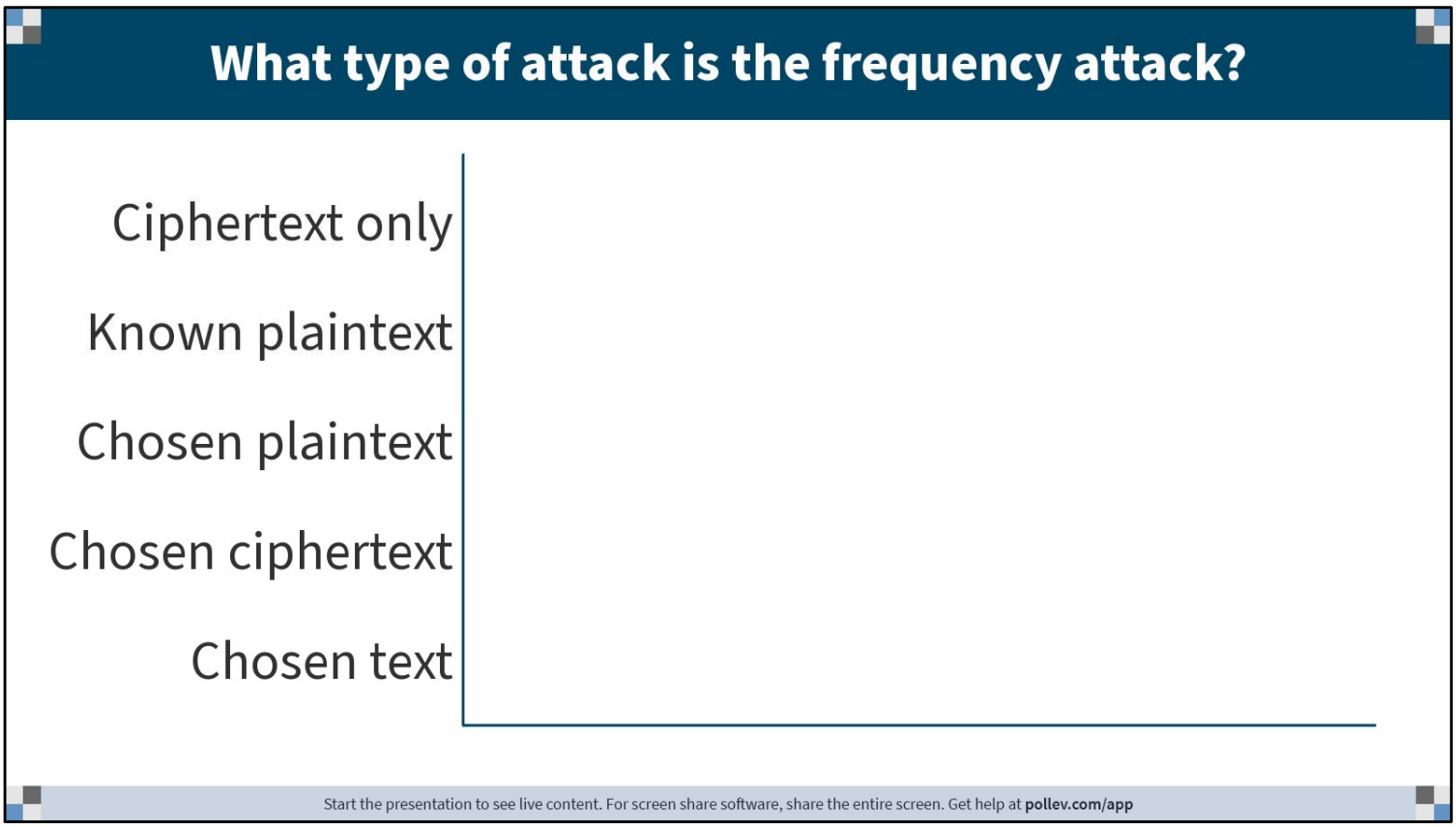
- Ciphertext

```
UzqSovUoHxmoPvgPozPevSgzWSzoPfPeSxUDBmeT
SxaIzvUePHzHmDzSHzoWSfPaPPDTSvPqUzWymxUz
UHSxePyePoPDzSzUfPomBzWPfUPzHmDJUDTmoHmq
```

- Frequencies of letters in ciphertext

| | | | | |
|---|---|---|---|---|
| P 13.33 | H 5.83 | F 3.33 | B 1.67 | C 0.00 |
| Z 11.67 | D 5.00 | W 3.33 | G 1.67 | K 0.00 |
| S 8.33 | E 5.00 | Q 2.50 | Y 1.67 | L 0.00 |
| U 8.33 | V 4.17 | T 2.50 | I 0.83 | N 0.00 |
| O 7.50 | X 4.17 | A 1.67 | J 0.83 | R 0.00 |
| M 6.67 | | | | |

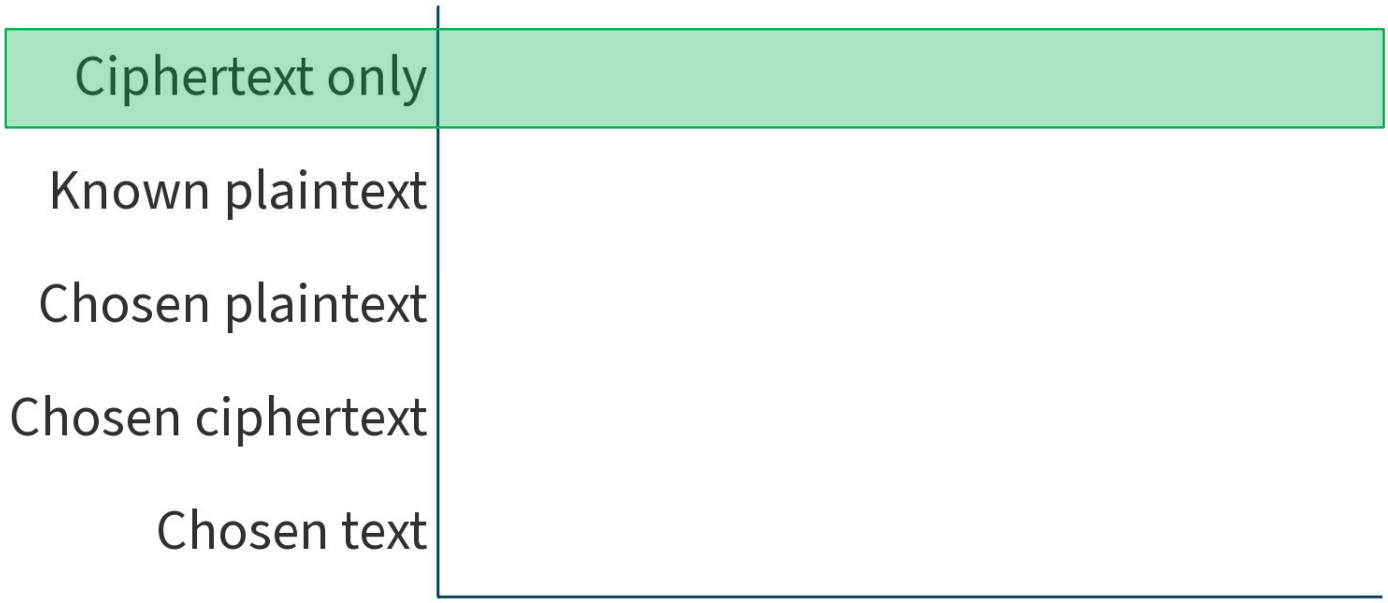
- One-on-one mapping with the English language letter frequencies
  - P = e  Z = t  S = a  …
- Requires some trial and error, as it becomes harder to guess for less frequent letters




Poll Title: Do not modify the notes in this section to avoid tampering with the Poll Everywhere activity.
More info at polleverywhere.com/support

What type of attack is the frequency attack?
https://www.polleverywhere.com/multiple_choice_polls/V5j1cfQDdgUnvoC

## What type of attack is the frequency attack?




Poll Title: What type of attack is the frequency attack?

# Vignère: a polyalphabetic cipher (Rome, 16th century)

- Use all 26 mono-alphabetic general Caesar cipher substitution rules
- A key represents the order in which the 26 substitution rules are used to encode subsequent plaintext letters

Plaintext: $P = p_0, p_1, \ldots, p_{N-1}$

Key: $K = k_0, k_1, \ldots, k_{M-1}$

Ciphertext: $C = c_0, c_1, \ldots, c_{N-1}$

Key is generally (much) shorter than Plaintext (i.e., M << N)

Encryption: $c_i = (p_i + k_{i \bmod M}) \bmod N$

Decryption: $p_i = (c_i - k_{i \bmod M}) \bmod N$

## An example of the Vignère cipher

**Exercise:** Derive the plaintext, given the cipher and key below

Key: `pass`

Ciphertext: `ihwscsowg`

| a | b | c | d | e | f | g | h | i | j | k | l | m |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| **n** | **o** | **p** | **q** | **r** | **s** | **t** | **u** | **v** | **w** | **x** | **y** | **z** |
| 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |

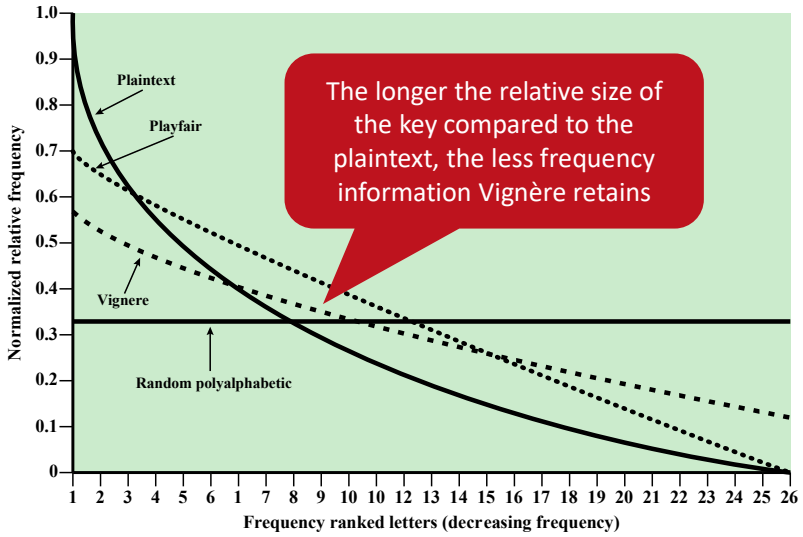
## Solution: An example of the Vignère cipher

**Exercise:** Derive the plaintext, given the cipher and key below

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $c_i$ | i | h | w | s | c | s | o | w | g |
| | 8 | 7 | 22 | 18 | 2 | 18 | 14 | 22 | 6 |
| $k_i$ | p | a | s | s | p | a | s | s | p |
| | 15 | 0 | 18 | 18 | 15 | 0 | 18 | 18 | 15 |
| $p_i = (c_i - k_{i \bmod m}) \bmod 26$ | 19 | 7 | 4 | 0 | 13 | 18 | 22 | 4 | 17 |
| | t | h | e | a | n | s | w | e | r |

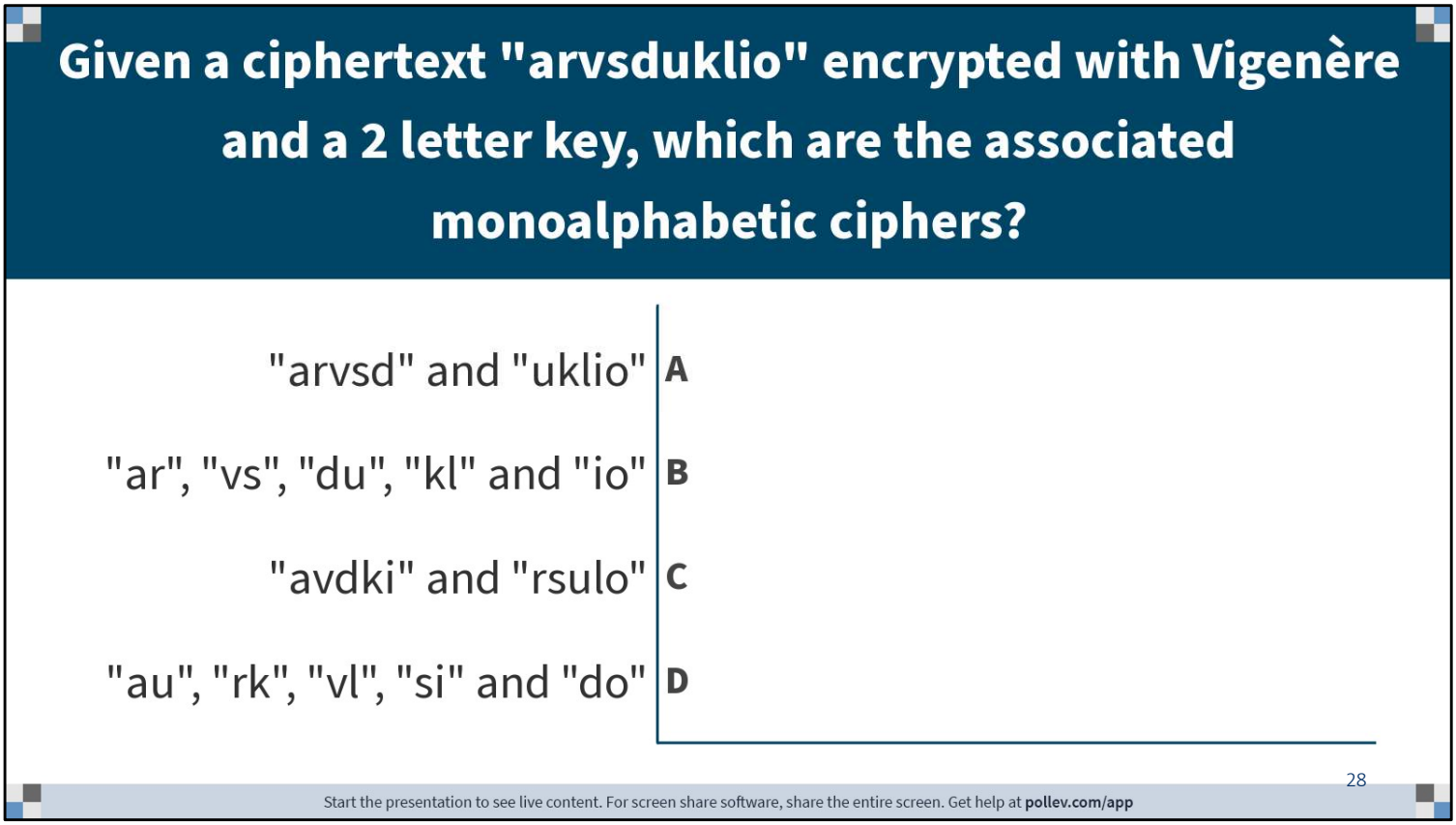
# Vignère cipher still retains frequency information

Which makes it vulnerable to some extent to frequency attacks



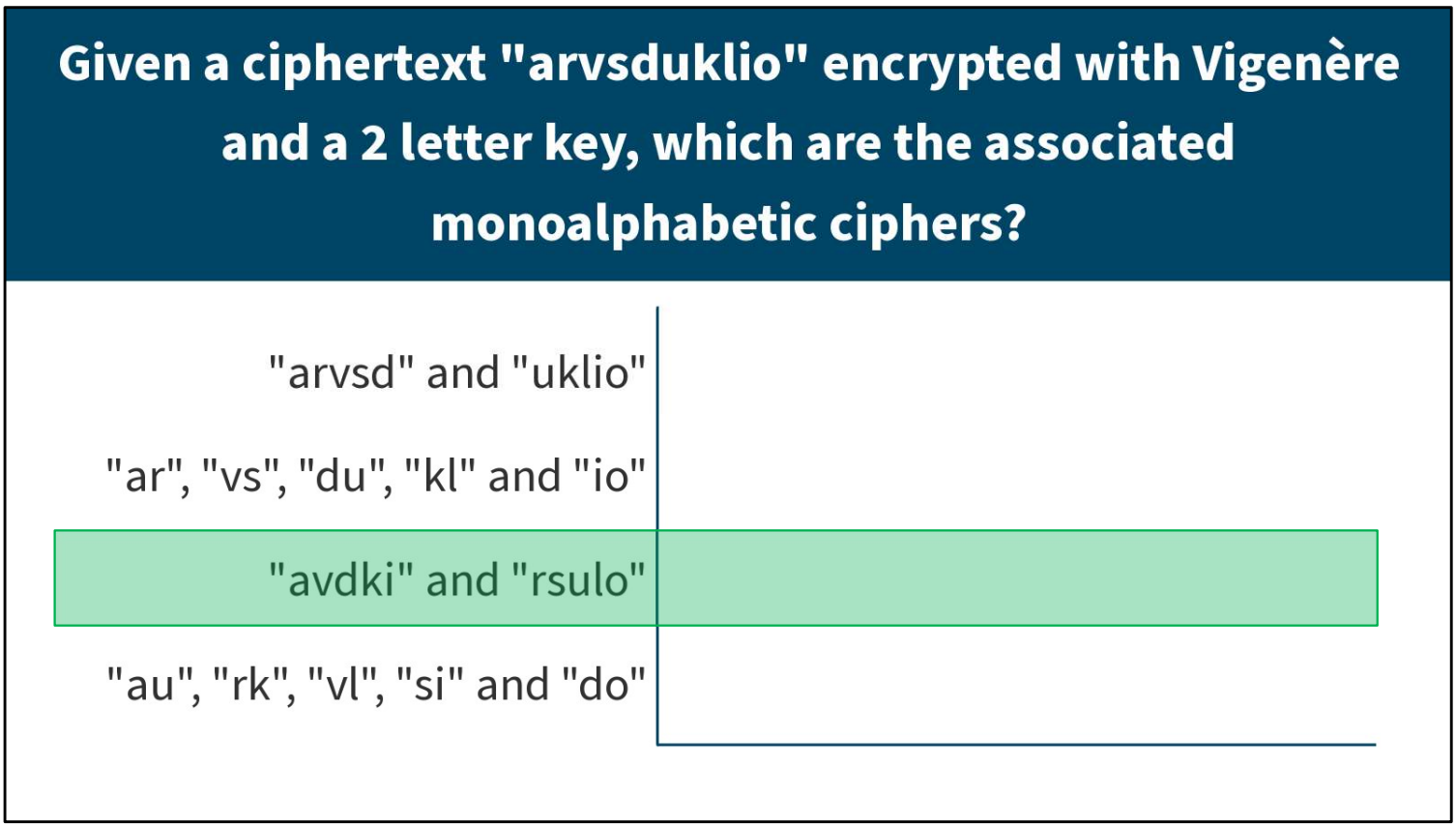
# Cracking the Vigenère cipher

- **Determining key length**
  - When the same sequence occurs at a distance that is an integer multiple of the key length, it will generate identical ciphertext
- **Break up the cipher into m mono-alphabetic ciphers**
  - The letters encoded with the same letter from the key together form a monoalphabetic cipher that can easily be cracked with a frequency attack




Poll Title: Do not modify the notes in this section to avoid tampering with the Poll Everywhere activity.
More info at polleverywhere.com/support

Given a ciphertext "arvsduklio" encrypted with Vigenère and a 2 letter key, which are the associated monoalphabetic ciphers?
https://www.polleverywhere.com/multiple_choice_polls/3U9sTdsAa7h0E5y




Poll Title: Given a ciphertext "arvsduklio" encrypted with Vigenère and a 2 letter key, which are the associated monoalphabetic ciphers?

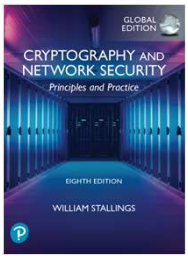
## Rotor machines (1870 – 1970)

- Mechanical rotating disks substitute each typed letter based on polyalphabetic substitution cipher

- The most famous example is the German Enigma machine, whose messages were deciphered by the Allies during World War II

- Also vulnerable to frequency attacks

# Further reading material




Chapter 3: Classical Encryption Techniques


Week 1: What is cryptography?

- History of cryptography

# Stream ciphers

1 One Time Pad

2 Practical Stream Ciphers

3 Modern Stream Cipher Examples

University of Antwerp
Faculty of Science

## Classical ciphers rely on simple substitution

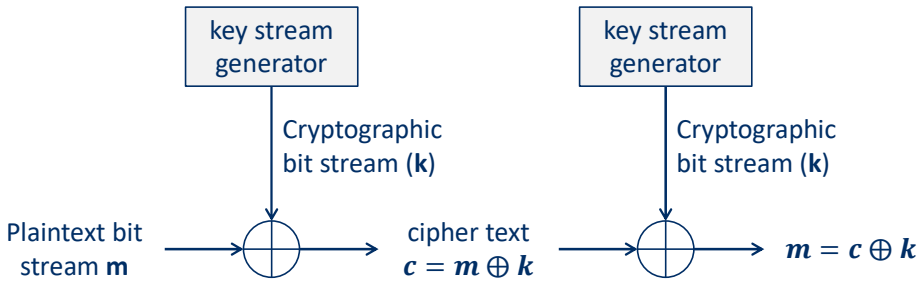
- Caesar cipher
- General substitution cipher
- Vigenere cipher
- Rotor machine
- …

**vulnerable to brute force or frequency attacks**

- The first secure cipher was developed by Vernam in 1917
  - Called the One Time Pad
  - $\boldsymbol{M} = \boldsymbol{C} = \{0,1\}^n$ and $\boldsymbol{K} = \{0,1\}^n$
  - Key is a random bit string with the same length as the message
  - Key is only used one time (no key reuse)

# One Time Pad (OTP)

Relies on bitwise XOR operation of message and key bit streams



## One Time Pad example and proof of correctness

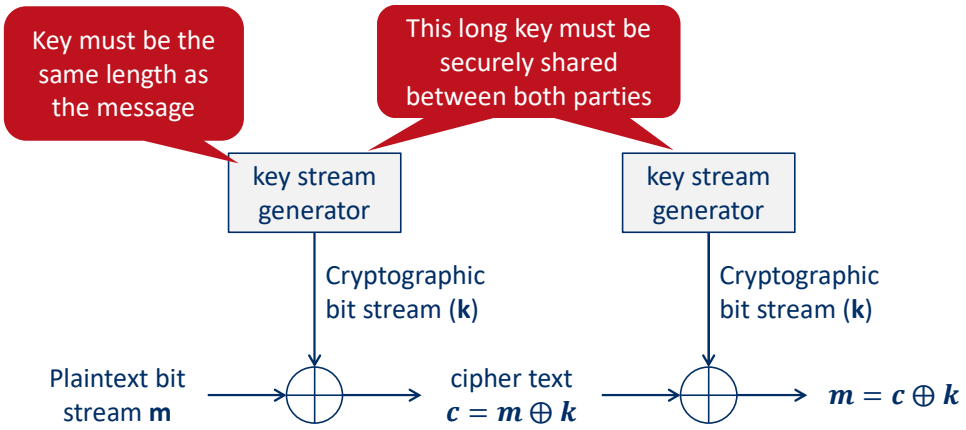
| | | |
|---|---|---|
| Message: | | 0 1 1 0 1 1 1 |
| Key: | $\oplus$ | 1 0 1 1 0 0 1 |
| Ciphertext: | | 1 1 0 1 1 1 0 |

Proof of correctness:

$$\forall m \in \boldsymbol{M}, k \in \boldsymbol{K}: \boldsymbol{D}\big(k, \boldsymbol{E}(k,m)\big) = m$$

$$D\big(k, E(k,m)\big) = D(k, k \oplus m) = k \oplus (k \oplus m) = (k \oplus k) \oplus m = m$$

## Practical problems with OTP



## Security definitions for encryption algorithms

**Unconditional or perfect security**

- The ciphertext generated by the scheme does not contain enough information to determine uniquely the corresponding plaintext, no matter how much ciphertext is available.
- $\forall m_0, m_1 \in \boldsymbol{M}, \forall c \in \boldsymbol{C}: \Pr[E(k, m_0) = c] = \Pr[E(k, m_1) = c]$
- One Time Pad is the only known unconditionally secure algorithm

**Computational or semantic security**

- The cost of breaking the cipher exceeds the information value, or
- The time required to break the cipher exceeds information lifetime
- $\forall m_0, m_1 \in \boldsymbol{M}, \forall c \in \boldsymbol{C}: |\Pr[E(k, m_0) = c] - \Pr[E(k, m_1) = c]| \leq \epsilon$

1 One Time Pad

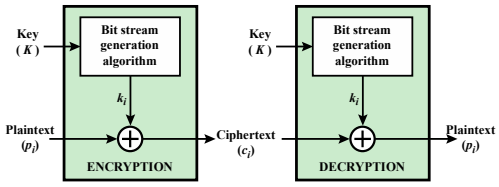
2 Practical Stream Ciphers

3 Modern Stream Cipher Examples

University of Antwerp
Faculty of Science

# Stream vs. block ciphers

## Stream ciphers

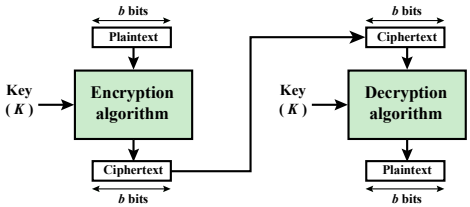



- Encrypts data one element (bit, byte, letter) at a time
- Practically, the key must be generated algorithmically

**Examples**

Vigenère, One Time Pad

## Block ciphers



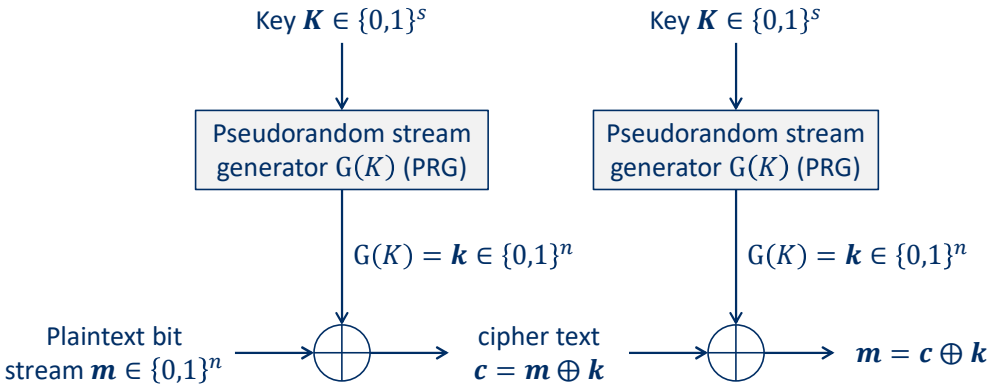

- Equal-size blocks of plaintext are encrypted as a whole
- Used by the vast majority of cryptographic applications

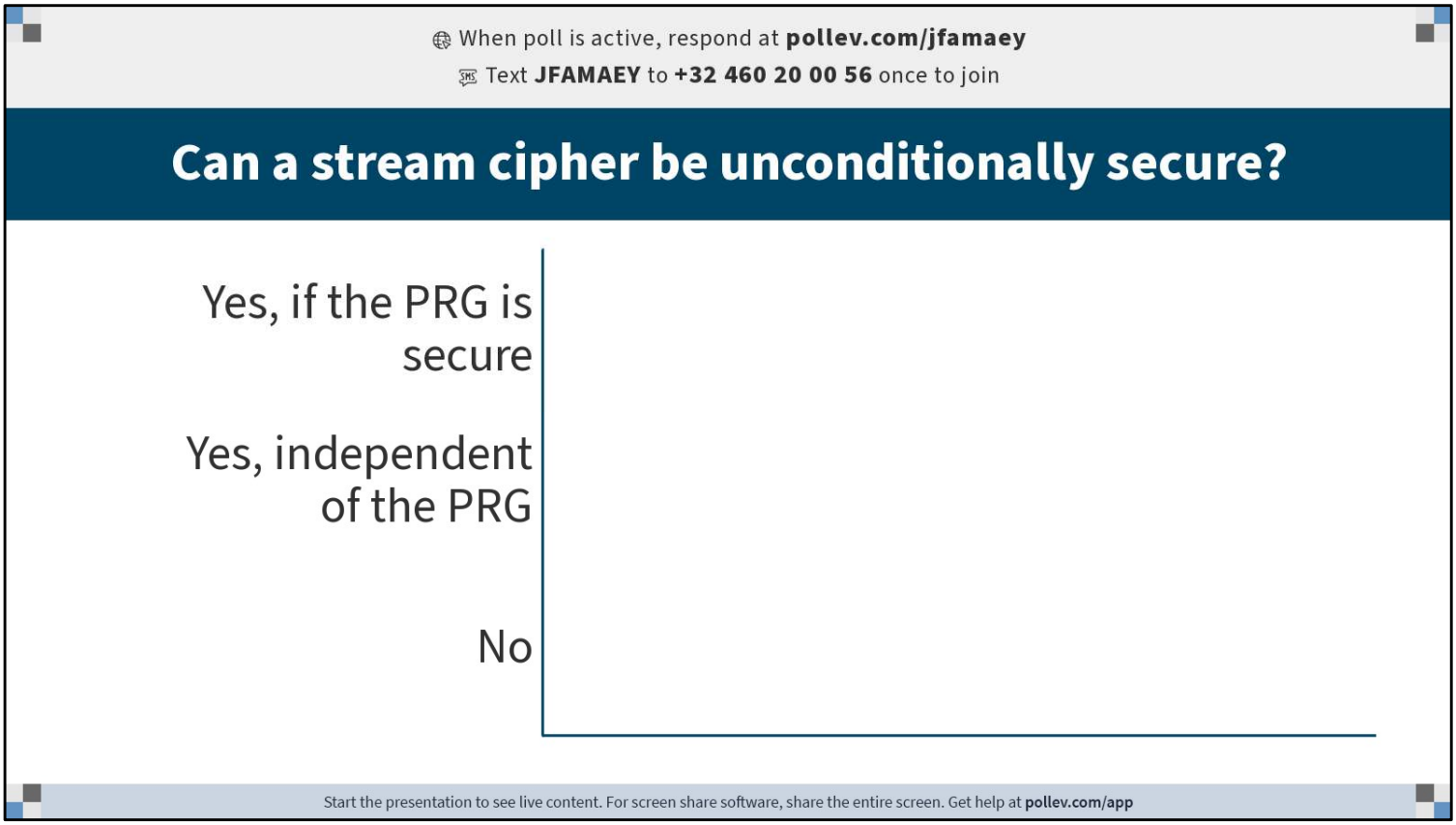
**Examples**

DES, AES

## Stream ciphers: a practical version of OTP

- Idea: replace “random” key by “pseudorandom” key
- Use Pseudo-Random Generator (PRG) that turns a seed into a pseudorandom bitstream: $G: \{0,1\}^s \rightarrow \{0,1\}^n$, with $n \gg s$

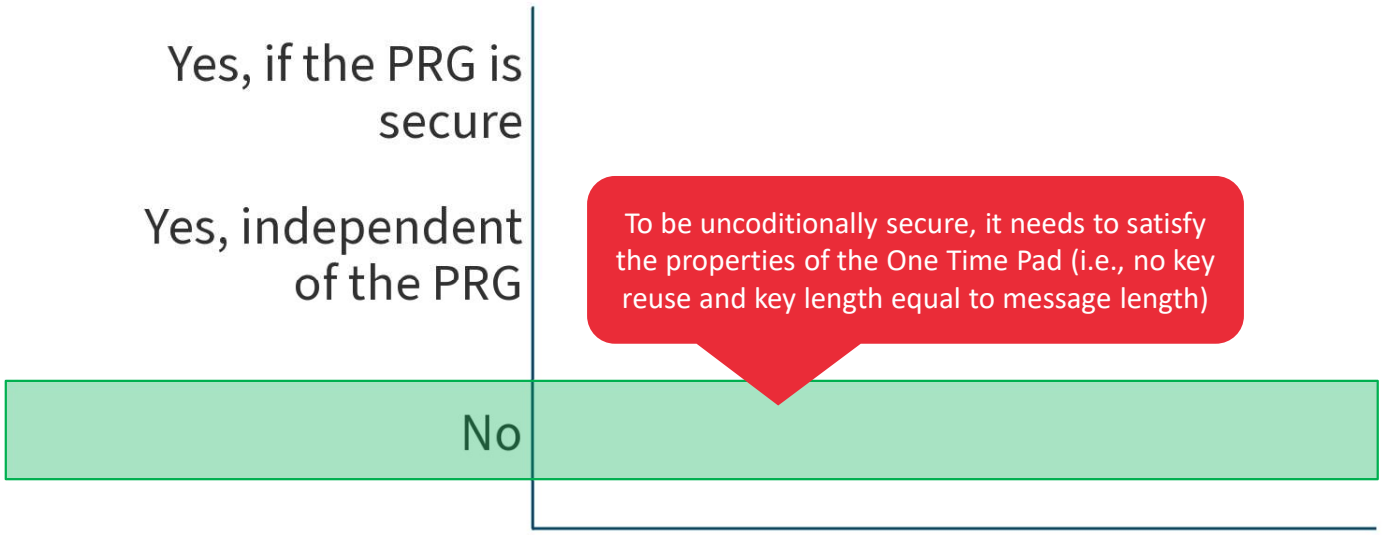





Poll Title: Do not modify the notes in this section to avoid tampering with the Poll Everywhere activity.
More info at polleverywhere.com/support

Can a stream cipher be unconditionally secure?
https://www.polleverywhere.com/multiple_choice_polls/XBfDCH947s9xs08?state=opened&flow=Default&onscreen=persist

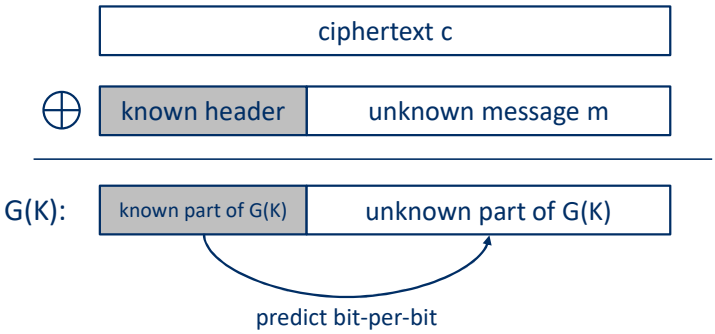
## Can a stream cipher be unconditionally secure?




Poll Title: Can a stream cipher be unconditionally secure?

## PRG must be unpredictable

- We say that G: $K \rightarrow \{0,1\}^n$ is predictable if:
  - There exists an "efficient" algorithm that, given the first i bits of G(K), is able to predict bit i + 1 with a "non-negligible" probability $\varepsilon$
  - How to exploit predictability?




- PRG is unpredictable if it is not predictable



In applications such as reciprocal authentication, session key generation, and stream ciphers, the requirement is not just that the sequence of numbers be statistically random but that the successive members of the sequence are unpredictable. With "true" random sequences, each number is statistically independent of other numbers in the sequence and therefore unpredictable. Although true random numbers are used in some applications, they have their limitations, such as inefficiency, as is discussed shortly. Thus, it is more common to implement algorithms that generate sequences of numbers that appear to be random. In this latter case, care must be taken that an opponent not be able to predict future elements of the sequence on the basis of earlier elements.

1 One Time Pad

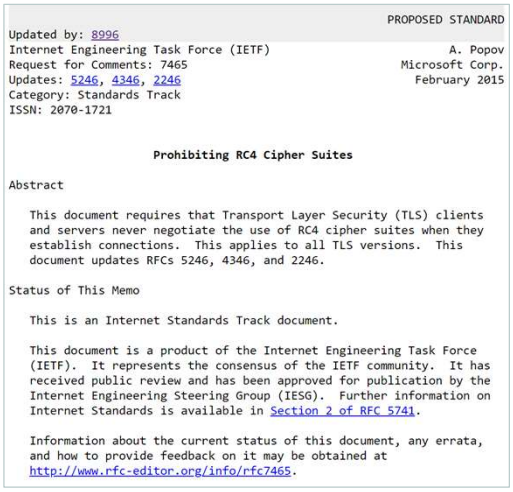
2 Practical Stream Ciphers

3 Modern Stream Cipher Examples

University of Antwerp
Faculty of Science

## RC4 stream cipher

- Designed by Ron Rivest in 1987 (one of the inventors of RSA)
- Variable key-size stream cipher with byte-oriented operations
- Was used in Wi-Fi WEP, but found to be insecure due to incorrect key generation method
- Currently still used in WPA2 and SSH
- Recent vulnerabilities (2015) have made RC4 prohibited for TLS



PROPOSED STANDARD
Updated by: 8996
Internet Engineering Task Force (IETF) — A. Popov
Request for Comments: 7465 — Microsoft Corp.
Updates: 5246, 4346, 2246 — February 2015
Category: Standards Track
ISSN: 2070-1721

**Prohibiting RC4 Cipher Suites**

Abstract

This document requires that Transport Layer Security (TLS) clients and servers never negotiate the use of RC4 cipher suites when they establish connections. This applies to all TLS versions. This document updates RFCs 5246, 4346, and 2246.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7465.

***Source***: *https://www.rfc-editor.org/rfc/rfc7465.html*



RC4 is a stream cipher designed in 1987 by Ron Rivest for RSA Security. It is a variable key size stream cipher with byte-oriented operations. The algorithm is based on the use of a random permutation. Analysis shows that the period of the cipher is overwhelmingly likely to be greater than 10100. Eight to sixteen machine operations are required per output byte, and the cipher can be expected to run very quickly in software. RC4 is used in the WiFi Protected Access (WPA) protocol that are part of the IEEE 802.11 wireless LAN standard. It is optional for use in Secure Shell (SSH) and Kerberos. RC4 was kept as a trade secret by RSA Security. In September 1994, the RC4 algorithm was anonymously posted on the Internet on the Cypherpunks anonymous remailers list.

Strength of RC4

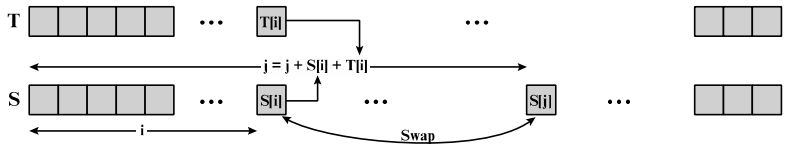
A number of papers have been published analyzing methods of attacking RC4. None of these approaches is practical against RC4 with a reasonable key length, such as 128 bits. A more serious problem was reported in 2001. It was demonstrated that the WEP protocol, intended to provide confidentiality on 802.11 wireless LAN networks, is vulnerable to a particular attack approach. In essence, the problem is not with RC4 itself but the way in which keys are generated for use as input to RC4. This particular problem does not appear to be relevant to other applications using RC4 and can be remedied in WEP by changing the way in which keys are generated. This problem points out the difficulty in designing a secure system that involves both cryptographic functions and protocols that make use of them.

More recently, a more fundamental vulnerability in the RC4 key scheduling algorithm was revealed that reduces the amount of effort to discover the key. Recent cryptanalysis

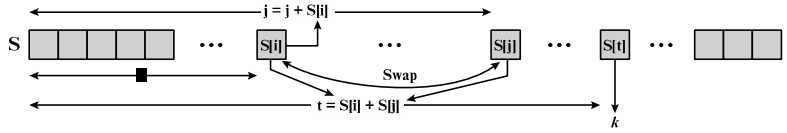
results exploit biases in the RC4 keystream to recover repeatedly encrypted plaintexts. As a result of the discovered weaknesses, the IETF issued RFC 7465 prohibiting the use of RC4 in TLS (*Prohibiting RC4 Cipher Suites*, February 2015). In its latest TLS guidelines, NIST also prohibited the use of RC4 for government use (SP 800-52, *Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations,* September 2013).

## How does the RC4 PRG work?

- The RC4 PRG maintains an internal state S of 256 bytes
- Initialization
  - Each byte S[i] of S is initialized to the value i
  - The key K is repeated until it has 256 bytes (called T)
  - T is used to create an initial permutation of S




- Stream generation
  - Each byte in S is swapped, and S[t] is selected as next output byte k




University of Antwerp
Faculty of Science

47

The RC4 algorithm is remarkably simple and quite easy to explain. A variable-length key of from 1 to 256 bytes (8 to 2048 bits) is used to initialize a 256-byte state vector S, with elements S[0],S[1], . . . ,S[255]. At all times, S contains a permutation of all 8-bit numbers from 0 through 255. For encryption and decryption, a byte *k* is generated from S by selecting one of the 255 entries in a systematic fashion. As each value of *k* is generated, the entries in S are once again permuted.

Initialization of S

To begin, the entries of S are set equal to the values from 0 through 255 in ascending order; that is, S[0] = 0, S[1] = 1, c , S[255] = 255. A temporary vector, T, is also created. If the length of the key K is 256 bytes, then K is transferred to T. Otherwise, for a key of length *keylen* bytes, the first *keylen* elements of T are copied from K, and then K is repeated as many times as necessary to fill out T. These preliminary operations can be summarized as:

```
/* Initialization */
for i = 0 to 255 do
        S[i] = i;
        T[i] = K[i mod keylen];
```

Next, we use T to produce the initial permutation of S. This involves starting with S[0] and going through to S[255], and for each S[i], swapping S[i] with another byte in S according to a scheme dictated by T[i]:

```
/* Initial Permutation of S */
j = 0;
for i = 0 to 255 do
        j = (j + S[i] + T[i]) mod 256;
        Swap (S[i], S[j]);
```

Because the only operation on S is a swap, the only effect is a permutation. S still contains all the numbers from 0 through 255.

Stream Generation

Once the S vector is initialized, the input key is no longer used. Stream generation involves cycling through all the elements of S[i], and for each S[i], swapping S[i] with another byte in S according to a scheme dictated by the current configuration of S. After S[255] is reached, the process continues, starting over again at S[0]:

```
/* Stream Generation */
i, j = 0;
while (true)
        i = (i + 1) mod 256;
        j = (j + S[i]) mod 256;
        Swap (S[i], S[j]);
        t = (S[i] + S[j]) mod 256;
        k = S[t];
```

To encrypt, XOR the value k with the next byte of plaintext. To decrypt, XOR the value *k* with the next byte of ciphertext.

## Why is RC4 not secure?

- RC4 does not use a nonce, which results in vulnerabilities due to key reuse
- Like other steam ciphers, RC4 is malleable (see further)
- The first several bytes of the keystream are strongly correlated with the key
- Keystream is biased towards various sequences
  - e.g., the second output byte has a 1/128 (instead of 1/256) chance to be zero
- NOMORE (Numerous Occurrence MOnitoring & Recovery Exploit) in TLS (HTTPS) and WPA-TKIP implementation of RC4 discovered in 2014
  - First practically feasible attack against RC4 (it takes about 75 hours to execute)
  - Relies on statistical biases in the keystream (as highlighted above)



- More information about the NOMORE attack can be found at http://www.rc4nomore.com/

## Salsa20 & ChaCha20: A more secure stream cipher

- Designed by Daniel J. Berstein in 2005
- No attacks exist against the full 20 round variant
  - The variant performing only 8 rounds can be broken in 2250 operations [Shi et al., 2012]
- A modified version called ChaCha was published in 2008
  - Uses a new round function with better diffusion
  - Increases performance on some hardware architectures
  - Published in IETF RFC 7539
- PRG is based on add-rotate-XOR (ARX) operations, combining
  - 32-bit addition
  - bitwise addition (XOR)
  - Fixed binary rotation (circular shift)
- Inputs
  - 256-bit key (128-bit key version also exists)
  - 64-bit nonce (unique (pseudo-)random number)
  - 64-bit counter



Source: Zhenqing Shi, Bin Zhang, Dengguo Feng, Wenling Wu (2012). "Improved Key Recovery Attacks on Reduced-Round Salsa20 and ChaCha". ICISC'12 Proceedings of the 15th International Conference on Information Security and Cryptology. Lecture Notes in Computer Science. 7839. pp. 337–351. doi:10.1007/978-3-642-37682-5_24. ISBN 978-3-642-37681-8.

## Advantages of add-rotate-XOR (ARX) ciphers

- Only use 3 operations in their round functions
  - Modular addition: $a + b \bmod 2^n$
  - Exclusive OR: $a \oplus b$
  - Fixed binary rotation: a <<< b
- These operations are computationally efficient to execute both in hardware and software
- They run in constant time
  - Avoids timing attacks (cf., lecture on asymmetric encryption for details on this)

## Salsa20 PRG overview

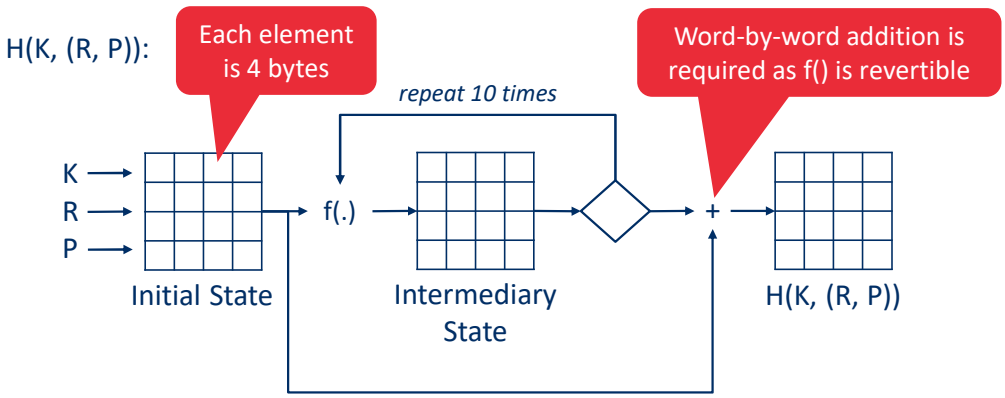
- Transform a key K and nonce R into key stream of plaintext length n
- $\{0,1\}^{256} \times \{0,1\}^{64} \rightarrow \{0,1\}^{n}$

  Key K — Nonce R
- Expansion function H(K, (R, P)):
  - Salsa20(K, R) = H(K, (R, 0)) || H(K, (R, 1)) || …

Repeated until desired number of bytes is reached

Each expansion function call increases position counter value

## Expansion function H() generates a 64-byte key block



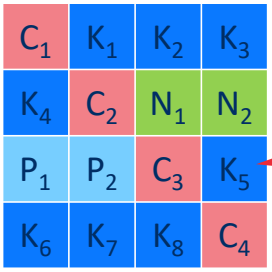

University of Antwerp
Faculty of Science

52

The expansion function H takes as input an initial state of 64 bytes, consisting of 16 (represented as a 4x4 matrix) words of 4 bytes each. The initial state is transformed using the round function f() into an intermediary state. This process is repeated 10 times, each of which consisting of two rounds (one column- and one row-transformation). Afterwards, it is summed word-by-word with the initial state, resulting in the 64-byte output. This word-by-word summation is performed because the round function f() is invertible, and could thus be reversed to obtain the original 4x4 matrix, and therefore the key K. Summing with the initial state, makes it impossible to revert the operation without knowing K, R, and P. To get the required number of n output bits, the entire process is repeated several times with different values for the counter P.

## Salsa20 PRG uses internal state of 16 words

Initial state (16 words of 4 bytes):




If K is only 128 bits, then $(K_5, K_6, K_7, K_8) = (K_1, K_2, K_3, K_4)$

128-bit constant C ("expand 32-byte k" in ASCII)

256-bit key K

64-bit nonce N (pseudo-random number)

64-bit position P (counter value)

P is increased by 1 every time H() is called

**Some notes:**

- All 4-byte words are represented using little endian notation (i.e., least significant byte comes first)
- The 8-byte counter P is represented with the least significant byte first
  - Thus $P = [p_0, p_1, p_2, p_3, p_4, p_5, p_6, p_7] = p_0 + 2^8\, p_1 + 2^{16}\, p_2 + \ldots + 2^{56}\, p_7$
  - In little endian form, the counter values $P_1$ and $P_2$ thus become $P_1 = p_3\, p_2\, p_1\, p_0$ and $P_2 = p_7\, p_6\, p_5\, p_4$
  - For example:
    $P = 300 = 0x12C = [2C, 01, 00, 00, 00, 00, 00, 00]$
    with (converting to little endian):
    $P_1 = 0x0000012C$ and $P_2 = 0x00000000$

University of Antwerp | Faculty of Science

53

The initial state is represented by a 4x4 matrix of 4-byte words (64 bytes in total) and is composed of a 128 bit constant (the ASCII string "expand 32-byte k"), a 256 bit key K (or a 128 bit key repeated twice), a 64-bit nonce that is pseudo-randomly generated, and a 64-bit position counter P. Between iterations of the expansion function H(), only the counter changes. When encrypting a different plaintext message, also the nonce should be different. The counter P is represented as an 8 byte sequence $[p_0, p_1, p_2, p_3, p_4, p_5, p_6, p_7]$, with $P_1 = [p_0, p_1, p_2, p_3]$ and $P_2 = [p_4, p_5, p_6, p_7]$. Note that the leftmost byte of the counter represents the least significant byte, and the counter value p of P thus represents the integer $p_0 + 2^8\, p_1 + 2^{16}\, p_2 + \ldots + 2^{56}\, p_7$. Additionally, all individual words are represented using little endian notation in the state matrix, as this is more efficient when performing calculations. As such, each 4-byte word $w = [b_3, b_2, b_1, b_0]$ is represented as $b_0\, b_1\, b_2\, b_3$ in the state matrix.

# Exercise: Calculating the initial 64-byte state

**Exercise**: Calculate the initial 4x4 64-byte state, given the following 128-bit key K, 64-bit nonce N, and position P.

Given:

- Key K = 0x4ff69c8e1e610aa5c72875cd8c7c69c8
- Nonce N = 0x341f8a2905ca7af8
- Position P = 17
- (note that K and N are in hexadecimal, while P is in decimal)

Determine the initial 4x4 state matrix:

(use little endian notation!)

| | | | |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |

| | | | |
|---|---|---|---|
| $C_1$ | $K_1$ | $K_2$ | $K_3$ |
| $K_4$ | $C_2$ | $N_1$ | $N_2$ |
| $P_1$ | $P_2$ | $C_3$ | $K_5$ |
| $K_6$ | $K_7$ | $K_8$ | $C_4$ |

128-bit constant C ("expand 32-byte k" in ASCII)

256-bit key K

64-bit nonce N (pseudo-random number)

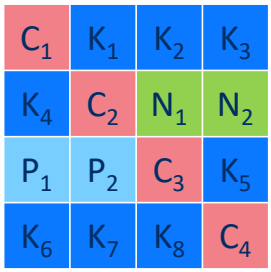
64-bit position P (counter value)

## Solution: Calculating the initial 64-byte state

- C = 0x657870616e642033322d62797465206b
- K = 0x4ff69c8e1e610aa5c72875cd8c7c69c8
- N = 0x341f8a2905ca7af8
- P = 17 = 0x1100000000000000

Least significant byte first!

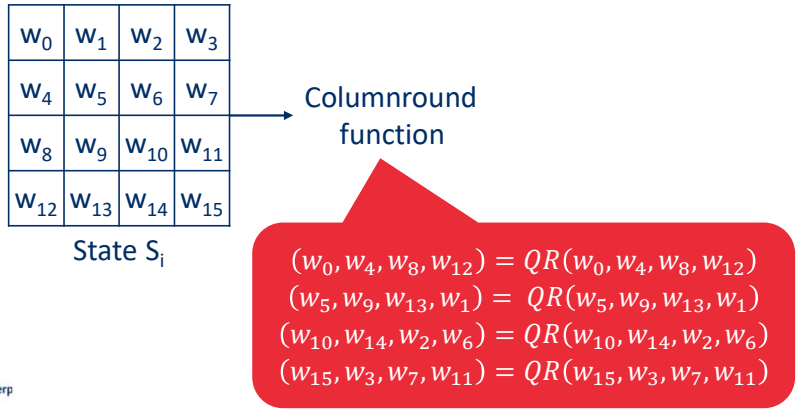
Little Endian!




| | | | |
|---|---|---|---|
| 0x61707865 | 0x8e9cf64f | 0xa50a611e | 0xcd7528c7 |
| 0xc8697c8c | 0x3320646e | 0x298a1f34 | 0xf87aca05 |
| 0x00000011 | 0x00000000 | 0x79622d32 | 0x8e9cf64f |
| 0xa50a611e | 0xcd7528c7 | 0xc8697c8c | 0x6b206574 |

## Round function f( )

- Each round applies the Quarterround function QR( ) twice
- QR( ) takes a 4-word input and generates a 4-word output
- First, QR( ) is applied column-per-column on the input state $S_i$
- Second, QR( ) is applied row-per-row



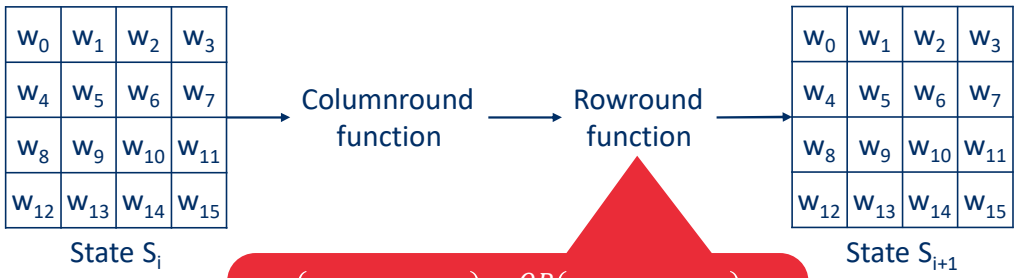

University of Antwerp
Faculty of Science

56

The round function is always applied two times in a row (referred to as a double-round). First, the Quarterround function QR() is applied to each column separately, transforming each 4-word (16 byte) column into a 4-word output column.

## Round function f( )

- Each round applies the Quarterround function QR( ) twice
- QR( ) takes a 4-word input and generates a 4-word output
- First, QR( ) is applied column-per-column on the input state Si
- Second, QR( ) is applied row-per-row




$$(w_0, w_1, w_2, w_3) = QR(w_0, w_1, w_2, w_3)$$
$$(w_5, w_6, w_7, w_4) = QR(w_5, w_6, w_7, w_4)$$
$$(w_{10}, w_{11}, w_8, w_9) = QR(w_{10}, w_{11}, w_8, w_9)$$
$$(w_{15}, w_{12}, w_{13}, w_{14}) = QR(w_{15}, w_{12}, w_{13}, w_{14})$$

University of Antwerp
Faculty of Science

57

Subsequently, the intermediary output state is again transformed, by applying the function QR to each 4-word row separately. The result is a new state $S_{i+1}$. Each of these transformations actually counts as two rounds. During odds rounds QR is thus applied to the columns of S, while during even rounds it's applied to its rows.

## Quarter round function QR( ) using ARX

$$(y_0, y_1, y_2, y_3) = QR(x_0, x_1, x_2, x_3)$$

Where:

Word-by-word addition modulo $2^{32}$

$$y_1 = x_1 \oplus ((x_0 + x_3)<<<7)$$
$$y_2 = x_2 \oplus ((y_1 + x_0)<<<9)$$
$$y_3 = x_3 \oplus ((y_2 + y_1)<<<13)$$
$$y_0 = x_0 \oplus ((y_3 + y_2)<<<18)$$

Bitwise XOR

Binary left rotation (leftmost bits are moved to the right

More information: https://cr.yp.to/snuffle/salsafamily-20071225.pdf

University of Antwerp | Faculty of Science

58

The quarter round function uses only three types of operation: 32-bit addition (add), bitwise addition (XOR), and bitwise left rotation (rotate), due to their computational efficiency. It calculates the 4 output words starting with $y_1$, then $y_2$, $y_3$ and $y_0$. When using this order, the original words do not need to be kept in memory after their associated output word has been calculated (i.e., $x_1$ can be discarded after calculating $y_1$, etc.). This reduces the memory footprint of the algorithm.

## Exercise: Solving the quarter round function QR

**Exercise**: Given the four 32-bit words $x_0$, $x_1$, $x_2$, and $x_3$, calculate QR($x_0$, $x_1$, $x_2$, $x_3$).

Given:

$$x_0 = 0xE4971786$$
$$x_1 = 0x0096E9C1$$
$$x_2 = 0x932f7534$$
$$x_3 = 0x27F0D9BB$$

Determine:

$$(y_0, y_1, y_2, y_3) = QR(x_0, x_1, x_2, x_3)$$

$$(y_0, y_1, y_2, y_3) = QR(x_0, x_1, x_2, x_3)$$

Where:

$$y_1 = x_1 \oplus ((x_0 + x_3)\text{<<<}7)$$
$$y_2 = x_2 \oplus ((y_1 + x_0)\text{<<<}9)$$
$$y_3 = x_3 \oplus ((y_2 + y_1)\text{<<<}13)$$
$$y_0 = x_0 \oplus ((y_3 + y_2)\text{<<<}18)$$

## Solution: Solving the quarter round function QR (1)

$x_0$ = 0xe4971786, $x_1$ = 0x0096e9c1,
$x_2$ = 0x932f7534, $x_3$ = 0x27f0d9bb

**Solution**

$$y_1 = x_1 \oplus \left((x_0 + x_3)\text{<<<}7\right)$$

| | |
|---|---|
| $x_0$ = 0xe4971786 | = 1110 0100 1001 0111 0001 0111 1000 0110 |
| $x_3$ = 0x27f0d9bb | = 0010 0111 1111 0000 1101 1001 1011 1011 |
| $x_0$ + $x_3$ | = 0000 1100 1000 0111 1111 0001 0100 0001 |
| <<< 7 | = 0100 0011 1111 1000 1010 0000 1000 0110 |
| $x_1$ | = 0000 0000 1001 0110 1110 1001 1100 0001 |
| XOR | = 0100 0011 0110 1110 0100 1001 0100 0111 |
| $y_1$ | = 0x436e4947 |

## Solution: Solving the quarter round function QR (2)

$x_0$ = 0xe4971786, $x_1$ = 0x0096e9c1,
$x_2$ = 0x932f7534, $x_3$ = 0x27f0d9bb

**Solution**

$$y_1 = x_1 \oplus \big((x_0 + x_3)\text{<<<}7\big), y_2 = x_2 \oplus \big((y_1 + x_0)\text{<<<}9\big)$$
$$y_3 = x_3 \oplus \big((y_2 + y_1)\text{<<<}13\big), y_0 = x_0 \oplus \big((y_3 + y_2)\text{<<<}18\big)$$

$y_1$ = 0100 0011 0110 1110 0100 1001 0100 0111 = 0x436e4947
$y_2$ = 1001 1001 1110 1110 1110 1111 0110 0100 = 0x99eeef64
$y_3$ = 1000 0000 1110 0101 1010 0010 0001 0000 = 0x80e5a210
$y_0$ = 1010 0001 0100 0111 0111 1100 1101 0100 = 0xa1477cd4

## Changes in ChaCha20 variant

- Initial state consists of the same 128-bit constant and 256-bit key, but a smaller counter (32 bits) and larger nonce (96 bits), ordered differently:

| | | | |
|---|---|---|---|
| $C_1$ | $C_2$ | $C_3$ | $C_4$ |
| $K_1$ | $K_2$ | $K_3$ | $K_4$ |
| $K_5$ | $K_6$ | $K_7$ | $K_8$ |
| $P_1$ | $N_1$ | $N_2$ | $N_3$ |

- The quarter-round function $QR(x_0, x_1, x_2, x_3)$ is replaced with:

$$x_0 \mathrel{+}= x_1; x_3 = x_0 \oplus x_3; x_3 = x_3 \lll 16;$$
$$x_2 \mathrel{+}= x_3; x_1 = x_1 \oplus x_2; x_2 = x_2 \lll 12;$$
$$x_0 \mathrel{+}= x_1; x_3 = x_0 \oplus x_3; x_3 = x_3 \lll 8;$$
$$x_2 \mathrel{+}= x_3; x_1 = x_1 \oplus x_2; x_2 = x_2 \lll 7;$$

- Finally, QR is applied to diagonals in even rounds instead of rows



Notice that this version updates each word twice, while Salsa20's quarter round updates each word only once. In addition, the ChaCha quarter-round diffuses changes more quickly. On average, after changing 1 input bit the Salsa20 quarter-round will change 8 output bits while ChaCha will change 12.5 output bits.

## Stream cipher pitfall 1: You should never reuse keys

- Never use stream cipher key more than once (!!)
- Two ciphertexts encrypted with the same key k

$$\begin{cases} c_1 \leftarrow m_1 \oplus \text{PRG}(k) \\ c_2 \leftarrow m_2 \oplus \text{PRG}(k) \end{cases}$$

- Eavesdropper can use ciphertexts to get information about plaintext:

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

- By exploiting characteristics of plaintexts, attacker can recover them (e.g., using a dictionary attack):

$$m_1 \oplus m_2 \rightarrow \; m_1, m_2$$

- By adding a unique nonce to the key, the key itself can be reused

## Stream cipher pitfall 2: Stream ciphers are malleable

- **Definition of malleability:**
  It is possible to transform a ciphertext into another ciphertext which decrypts to a related plaintext
- For a steam cipher, it is possible to calculate ciphertext $E(k, m \oplus t)$, given only the ciphertext $E(k, m)$, and without knowing the key stream $S(k)$, as:

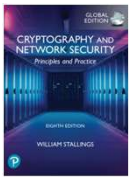
$$E(k, m) \oplus t = m \oplus S(k) \oplus t = E(k, m \oplus t)$$

- This allows performing a **bit-flipping attack**, where the attacker selects t to flip specific bits in the original message m to change its meaning
  - For example, an electronic fund transfer containing the ASCII string "1000€" can be changed into "9000€", by calculating:

$$E(k, \text{"1000€"}) \oplus (\text{"1000€"} \oplus \text{"9000€"})$$
$$= S(k) \oplus \text{"1000€"} \oplus (\text{"1000€"} \oplus \text{"9000€"}) = S(k) \oplus \text{"9000€"}$$
$$= \mathrm{E}(k, \text{"9000€"})$$

  - Can be prevented by including a **message authentication code** to detect tampering

# Further reading




Chapter 8 (Section 8.4—8.5)

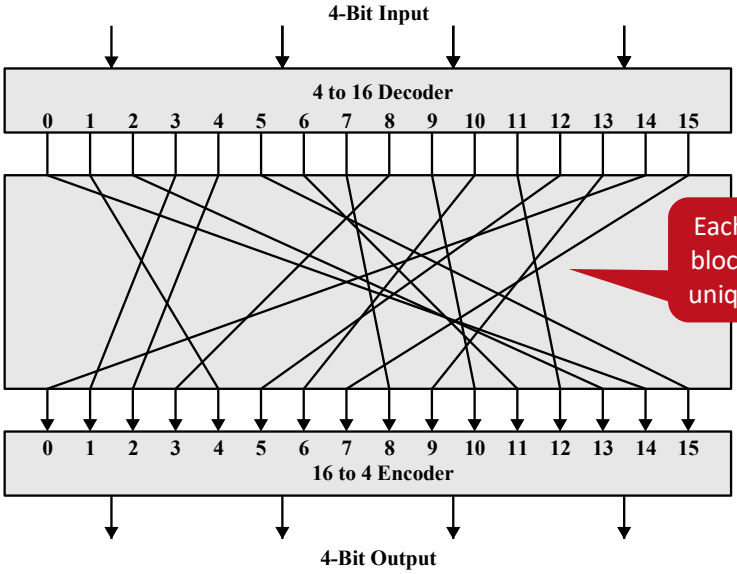
Stream ciphers (videos 1, 2, 3 and 4)

# Block ciphers

1. General block cipher structure
2. Advanced Encryption Standard (AES)
3. Detour: Finite field arithmetic
4. AES details

University of Antwerp
Faculty of Science

## General substitution: the ideal block cipher




University of Antwerp
Faculty of Science

68

A block cipher operates on a plaintext block of $n$ bits to produce a ciphertext block of $n$ bits. There are $2^n$ possible different plaintext blocks and, for the encryption to be reversible (i.e., for decryption to be possible), each must produce a unique ciphertext block. Such a transformation is called reversible, or nonsingular. The following examples illustrate nonsingular and singular transformations for $n = 4$.

# The ideal block cipher has some problems

### Small block size (e.g., n = 4)

It is equivalent to traditional substitution ciphers, and is vulnerable to frequency and other statistical attacks

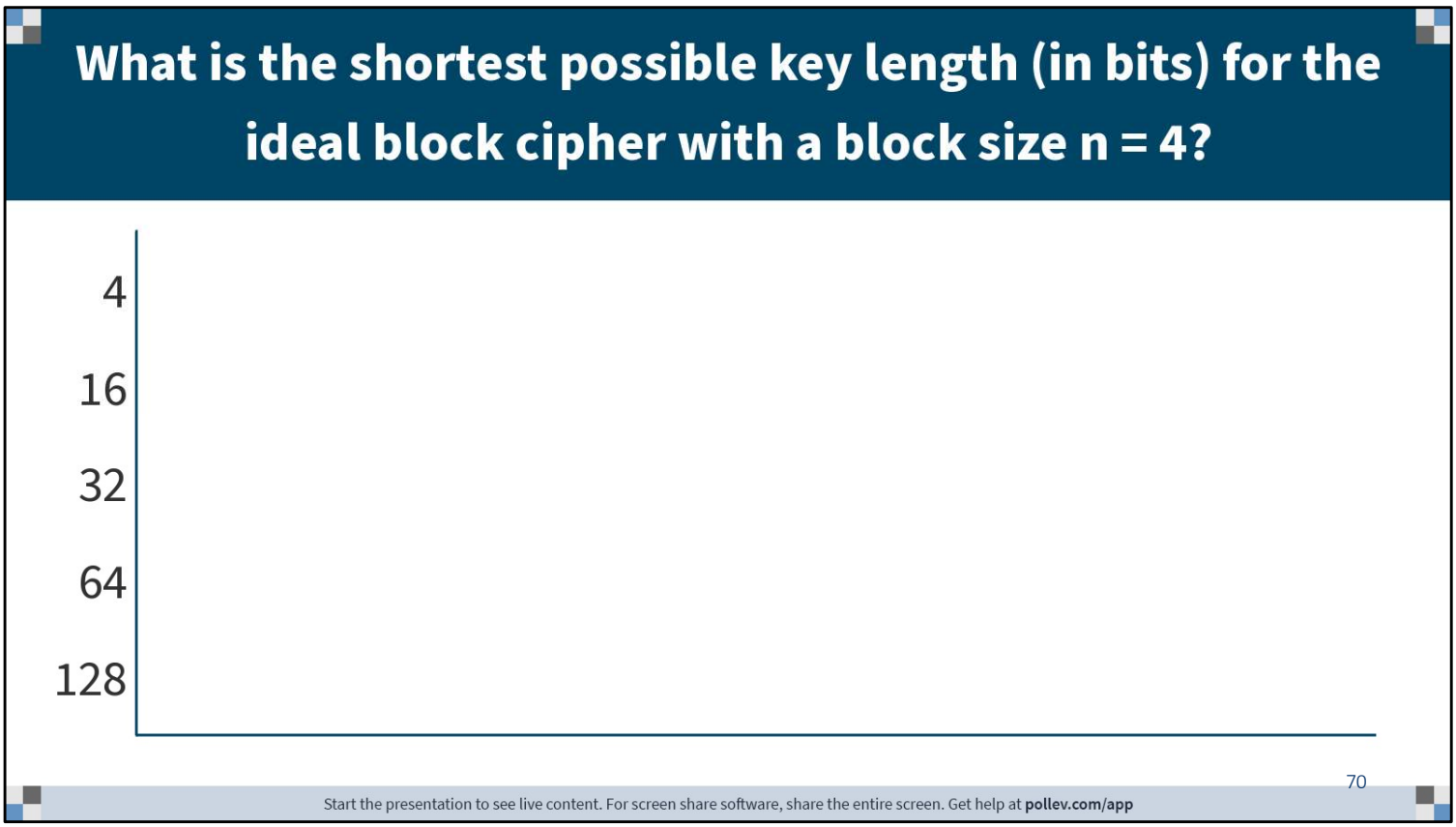
### Large block size

It is impractical, as the mapping of plain- onto ciphertext blocks needs to be captured by the key, which results in long keys

Example mapping for n = 4:

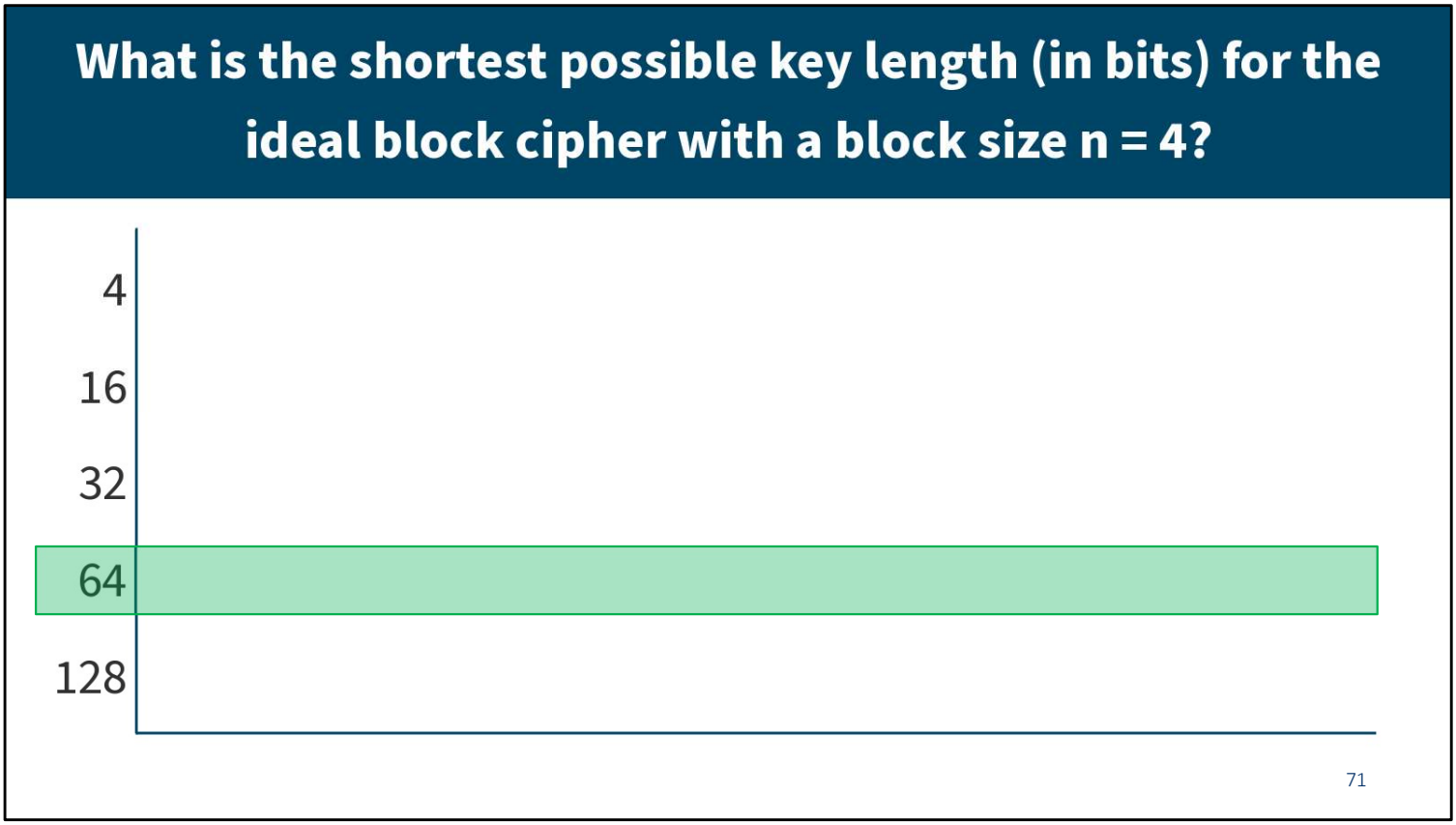
| Plaintext | Ciphertext |
|---|---|
| 0000 | 1110 |
| 0001 | 0100 |
| 0010 | 1101 |
| 0011 | 0001 |
| 0100 | 0010 |
| 0101 | 1111 |
| 0110 | 1011 |
| 0111 | 1000 |
| 1000 | 0011 |
| 1001 | 1010 |
| 1010 | 0110 |
| 1011 | 1100 |
| 1100 | 0101 |
| 1101 | 1001 |
| 1110 | 0000 |
| 1111 | 0111 |

| Ciphertext | Plaintext |
|---|---|
| 0000 | 1110 |
| 0001 | 0011 |
| 0010 | 0100 |
| 0011 | 1000 |
| 0100 | 0001 |
| 0101 | 1100 |
| 0110 | 1010 |
| 0111 | 1111 |
| 1000 | 0111 |
| 1001 | 1101 |
| 1010 | 1001 |
| 1011 | 0110 |
| 1100 | 1011 |
| 1101 | 0010 |
| 1110 | 0000 |
| 1111 | 0101 |




Poll Title: Do not modify the notes in this section to avoid tampering with the Poll Everywhere activity.
More info at polleverywhere.com/support

What is the shortest possible key length (in bits) for the ideal block cipher with a block size n = 4?
https://www.polleverywhere.com/multiple_choice_polls/FIfVyL2ostgHhJN




Poll Title: What is the shortest possible key length (in bits) for the ideal block cipher with a block size n = 4?

# Solution: Key length for the ideal block cipher

- The key can be uniquely defined as the ciphertext blocks in order:

  *1110 0100 1101 0001 …*

- Each block is 4 bits, and there are 24 (= 16) different block:

  *4 x 16 = 64 bits*

- Generalisation to n bit blocks:

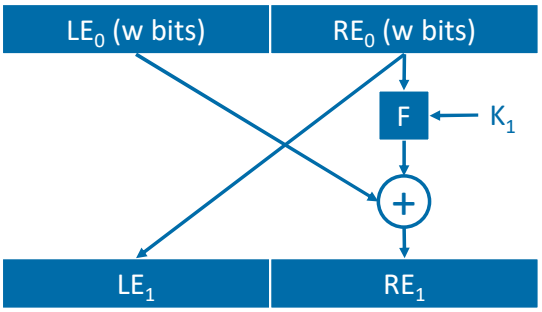
  key length = n x $2^n$

| Plaintext | Ciphertext |
|---|---|
| 0000 | 1110 |
| 0001 | 0100 |
| 0010 | 1101 |
| 0011 | 0001 |
| 0100 | 0010 |
| 0101 | 1111 |
| 0110 | 1011 |
| 0111 | 1000 |
| 1000 | 0011 |
| 1001 | 1010 |
| 1010 | 0110 |
| 1011 | 1100 |
| 1100 | 0101 |
| 1101 | 1001 |
| 1110 | 0000 |
| 1111 | 0111 |

| Ciphertext | Plaintext |
|---|---|
| 0000 | 1110 |
| 0001 | 0011 |
| 0010 | 0100 |
| 0011 | 1000 |
| 0100 | 0001 |
| 0101 | 1100 |
| 0110 | 1010 |
| 0111 | 1111 |
| 1000 | 0111 |
| 1001 | 1101 |
| 1010 | 1001 |
| 1011 | 0110 |
| 1100 | 1011 |
| 1101 | 0010 |
| 1110 | 0000 |
| 1111 | 0101 |

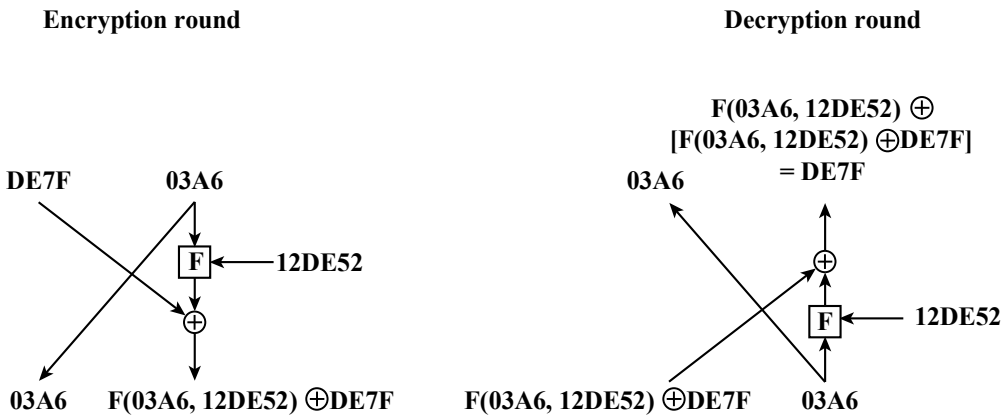
## Feistel cipher approximates the ideal block cipher

- Each block is 2w bits long
- The round function F uses the key $K_1$ (derived from the main key K)
- A substitution is performed on $LE_0$ using XOR with $F(RE_0, K_1)$
- A permutation is performed by switching the two data block halves



## A Feistel cipher example

Assume blocks of 32 bits and a 24-bit key (represented as hexadecimal numbers in the example)





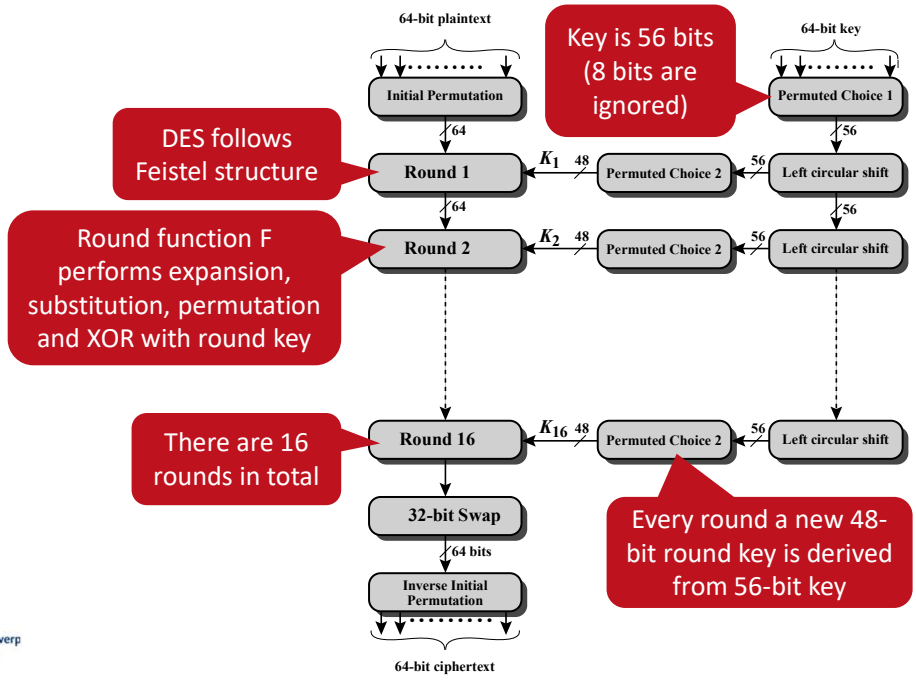

To help clarify the preceding concepts, let us look at a specific example. Suppose that the blocks at each stage are 32 bits (two 16-bit halves) and that the key size is 24 bits. Suppose that at the end of the previous encryption round, the value of the intermediate block (in hexadecimal) is DE7F03A6. Then $LE_0$ = DE7F and $RE_0$ = 03A6. Also assume that the value of $K$ is 12DE52. After the round, we have $LE_1$ = 03A6 and $RE_1$ = F(03A6, 12DE52) $\oplus$ DE7F.

Now let's look at the decryption. We assume that $LD_0 = RE_1$ and $RD_0 = LE_1$, and we want to demonstrate that $LD_1 = RE_0$ and $RD_1 = LE_0$. So, we start with $LD_0$ = F(03A6, 12DE52) $\oplus$ DE7F and $RD_0$ = 03A6. Then, $LD_1$ = 03A6 = $RE_0$ and $RD_1$ = F(03A6, 12DE52) $\oplus$ [F(03A6, 12DE52) $\oplus$ DE7F] = DE7F = $LE_0$.

## Data Encryption Standard (DES)

- Issued in 1977 by the National Institute of Standards and Technology (NIST)
- Most used encryption scheme until introduction of AES in 2001
- DES uses 64-bit blocks and 56-bit keys, deemed unsafe
- Triple DES (3DES), which increases security through repeating the DES algorithm 3 times, was introduced in 1999

# High level overview of DES




Key is 56 bits (8 bits are ignored)

DES follows Feistel structure

Round function F performs expansion, substitution, permutation and XOR with round key

There are 16 rounds in total

Every round a new 48-bit round key is derived from 56-bit key

# Why use a 56 bit key?

~~(FOUO)~~ In 1973 NBS solicited private industry for a data encryption standard (DES). The first offerings were disappointing, so NSA began working on its own algorithm. Then Howard Rosenblum, deputy director for research and engineering, discovered that Walter Tuchman of IBM was working on a modification to Lucifer for general use. NSA gave Tuchman a clearance and brought him in to work jointly with the Agency on his Lucifer modification.

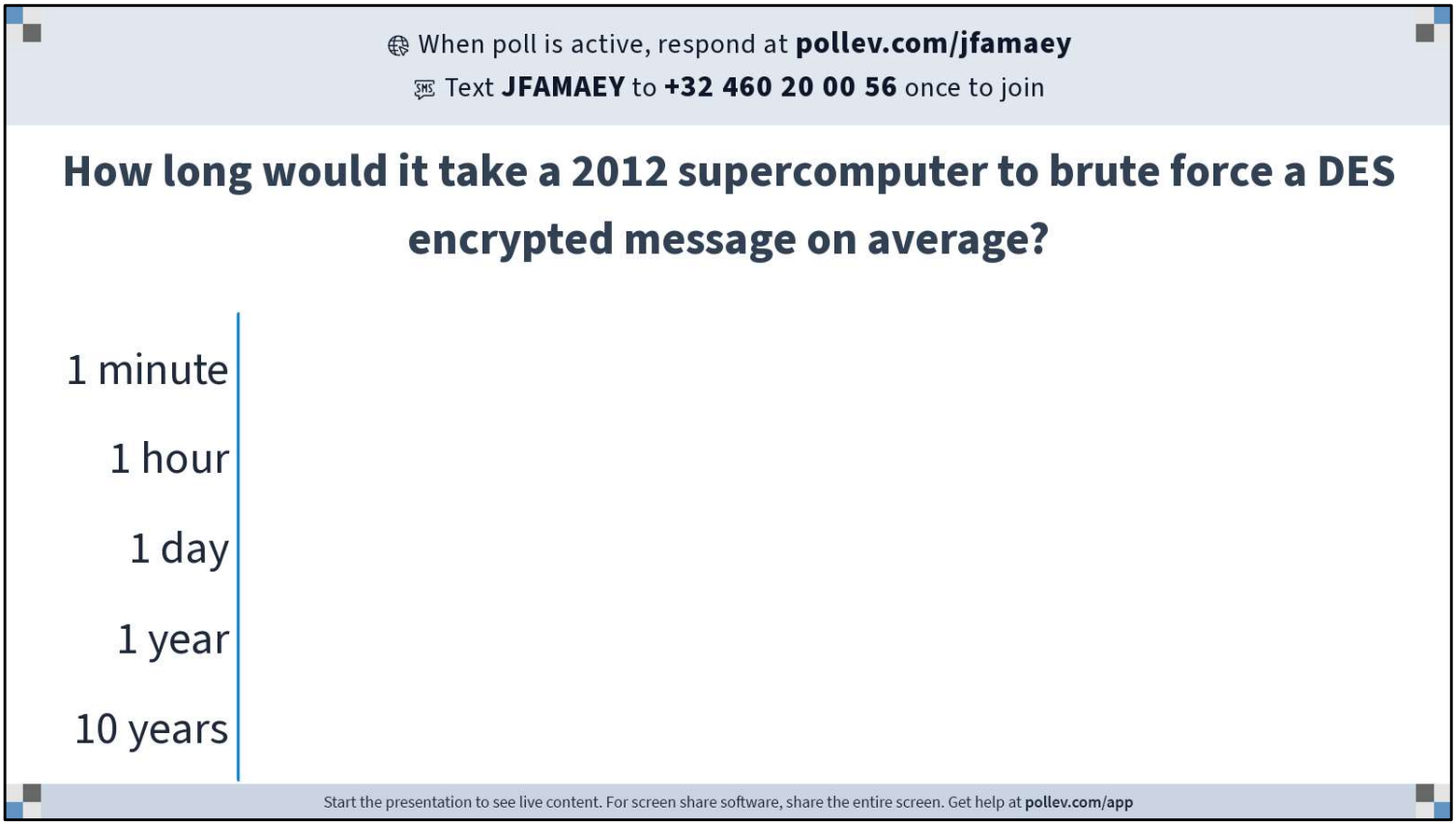
~~(S-CCO)~~ The decision to get involved with NBS was hardly unanimous. From the SIGINT standpoint, a competent industry standard could spread into undesirable areas, like Third World government communications, narcotics traffickers, and international terrorism targets.

This argued the opposite case – that, as Frank Rowlett had contended since World War II, in the long run it was more important to secure one's own communications than to exploit those of the enemy.[121]

~~(FOUO)~~ Once that decision had been made, the debate turned to the issue of minimizing the damage. Narrowing the encryption problem to a single, influential algorithm might drive out competitors, and that would reduce the field that NSA had to be concerned about.

NSA worked closely with IBM to strengthen the algorithm against all except brute force attacks and to strengthen substitution tables, called S-boxes. Conversely, NSA tried to convince IBM to reduce the length of the key from 64 to 48 bits. Ultimately, they compromised on a 56-bit key.[122]

*Source: https://www.nsa.gov/news-features/declassified-documents/gulf-of-tonkin/articles/assets/files/release-2/rel2_american_cryptology.pdf (page 232)*




Poll Title: Do not modify the notes in this section to avoid tampering with the Poll Everywhere activity.
More info at polleverywhere.com/support

How long would it take a 2012 supercomputer to brute force a DES encrypted message on average?
https://www.polleverywhere.com/multiple_choice_polls/FzcRMUHLIqIiRxEE0kZSn?state=opened&flow=Default&onscreen=persist

# Main shortcoming of DES: Key size

- There are $2^{56}$ possible keys, which would mean a brute-force attack with 1 attempted decryption per microsecond would take 1000 years
- However, modern hardware is much faster than that!

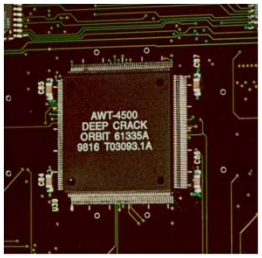
average 2009 PC

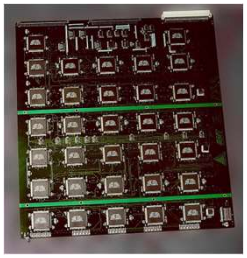
2012 supercomputer

| Key Size (bits) | Cipher | Number of Alternative Keys | Time Required at $10^9$ Decryptions/s | Time Required at $10^{13}$ Decryptions/s |
|---|---|---|---|---|
| 56 | DES | $2^{56} \approx 7.2 \times 10^{16}$ | $2^{55}$ ns = 1.125 years | 1 hour |
| 128 | AES | $2^{128} \approx 3.4 \times 10^{38}$ | $2^{127}$ ns = $5.3 \times 10^{21}$ years | $5.3 \times 10^{17}$ years |
| 168 | Triple DES | $2^{168} \approx 3.7 \times 10^{50}$ | $2^{167}$ ns = $5.8 \times 10^{33}$ years | $5.8 \times 10^{29}$ years |
| 192 | AES | $2^{192} \approx 6.3 \times 10^{57}$ | $2^{191}$ ns = $9.8 \times 10^{40}$ years | $9.8 \times 10^{36}$ years |
| 256 | AES | $2^{256} \approx 1.2 \times 10^{77}$ | $2^{255}$ ns = $1.8 \times 10^{60}$ years | $1.8 \times 10^{56}$ years |
| 26 characters (permutation) | Monoalphabetic | $2! = 4 \times 10^{26}$ | $2 \times 10^{26}$ ns = $6.3 \times 10^{9}$ years | $6.3 \times 10^{6}$ years |

## History of brute-force attacks against DES

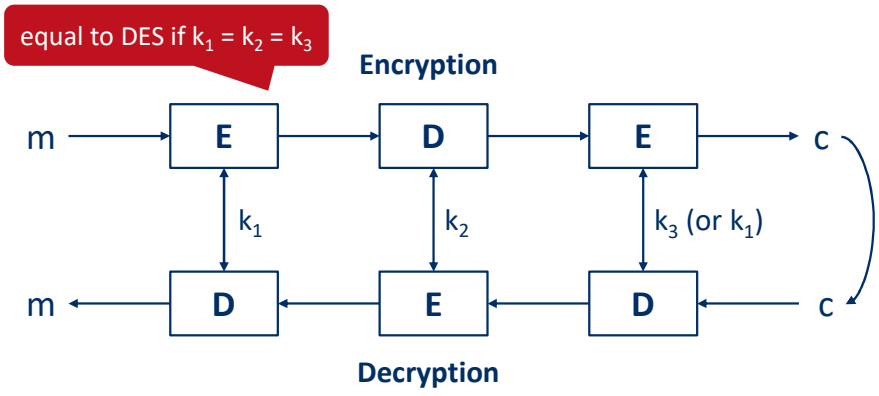
- Successful brute-force attacks have been conducted as early as 1997
- 1977: Diffie and Hellman propose a dedicated DES cracking machine which would cost $20 million and crack DES in 1 day
- 1993: Wieter proposes a key-search machine that costs $1 million and finds the key in 7 hours
- 1997: DESCHALL Project wins $10.000 prize as the first to decrypt DES using idle cycles of thousands of PCs across the Internet
- 1998: Electronic Frontier Foundation (EFF) builds first actual DES cracker for $250.000, which finds a key in about 2 days



## Triple DES (3DES): A secure alternative to DES

- Employs three subsequent stages with 2 or 3 keys
- The decrypt stage allows 3DES to decrypt regular DES ciphertext
- 2-key variant reuses first key in third stage (112 bit key)
- 3-key variant uses unique key in each stage (168 bit key)




University of Antwerp | Faculty of Science

81

There are two versions of 3DES; one using two keys and one using three keys. NIST SP 800-67 (*Recommendation for the Triple Data Encryption Block Cipher,* January 2012) defines the two-key and three-key versions. We look first at the two-key version and then examine the three-key version.

Two-key triple encryption was first proposed by Tuchman in 1979. The function follows an encrypt-decrypt-encrypt (EDE) sequence:

$C = E(K1, D(K2, E(K1, P)))$

$P = D(K1, E(K2, D(K1, C)))$

There is no cryptographic significance to the use of decryption for the second stage. Its only advantage is that it allows users of 3DES to decrypt data encrypted by users of the older single DES:

$C = E(K1, D(K1, E(K1, P))) = E(K1, P)$

$P = D(K1, E(K1, D(K1, C))) = D(K1, C)$

3DES with two keys is a relatively popular alternative to DES and has been adopted for use in the key management standards ANSI X9.17 and ISO 8732.1.

In SP 800-57, Part 1 (*Recommendation for Key Management—Part 1: General,* July 2012) NIST recommends that 2-key 3DES be retired as soon as practical and replaced with 3-key 3DES. Three-key 3DES is defined as

$C = E(K3, D(K2, E(K1, P)))$

Backward compatibility with DES is provided by putting $K3 = K2$ or $K1 = K2$. One might expect that 3TDEA would provide 56 * 3 = 168 bits of strength. However, there is an attack on 3TDEA that reduces the strength to the work that would be involved in

exhausting a 112-bit key. A number of Internet-based applications have adopted three-key 3DES, including PGP and S/MIME.

## Summary up to now...

| Cipher | Type | Characteristics |
|---|---|---|
| Caesar | Substitution | Extremely vulnerable to **brute force attacks** |
| Monoalphabetic | Substitution | Vulnerable to statistical **frequency attacks** |
| Vigenère | Substitution | Limited vulnerability to **frequency attacks** |
| One-time pad | Substitution | **Unconditionally secure**, but practically infeasible |
| RC4 | Substitution | Not secure due to **bias in PRG** |
| Salsa20 | Substitution | **Computationally secure** with key size of 256 bits |
| DES | Hybrid | Vulnerable to **brute force** due to 56-bit key size |
| Triple DES | Hybrid | **Computationally secure** with key size of 168 bits |

1. General block cipher structure
2. Advanced Encryption Standard (AES)
3. Detour: Finite field arithmetic
4. AES details

University of Antwerp
Faculty of Science

## Advanced Encryption Standard history

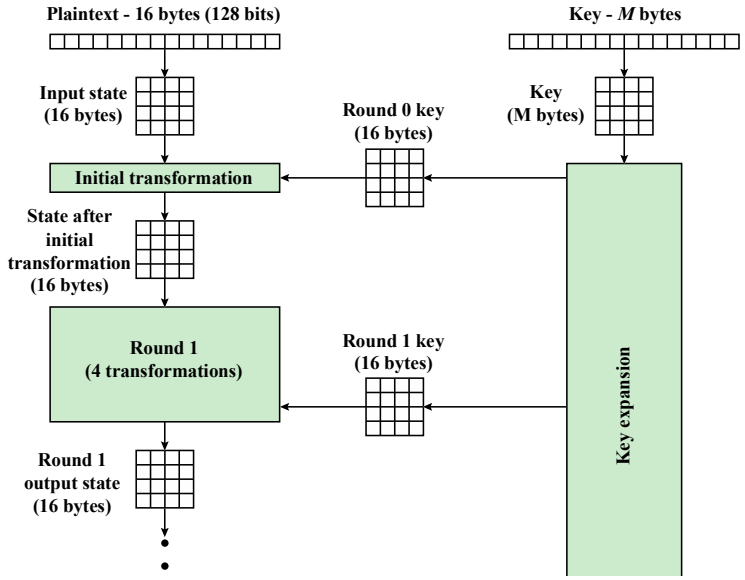
- NIST launched an open call in 1997, and received 15 submissions
- The 5 best ones were selected in terms of:
  - Cryptographic security
  - RAM and CPU performance on PCs, embedded systems and in hardware
- In 2000 NIST announced Rijndael as the winner, which was invented by Joan Daemen and Vincent Rijmen at KU Leuven
- AES is based on a version of Rijndael with block size 128 bits and key lengths 128, 192 and 256 bits

Joan Daemen en Vincent Rijmen



The Advanced Encryption Standard (AES) was published by the National Institute of Standards and Technology (NIST) in 2001. AES is a symmetric block cipher that is intended to replace DES as the approved standard for a wide range of applications.

## AES general structure (1/2)



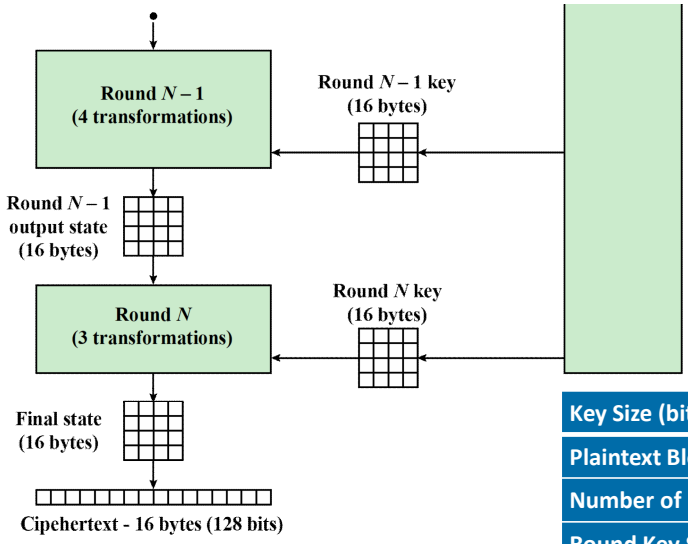

| No. of rounds | Key Length (bytes) |
|---|---|
| 10 | 16 |
| 12 | 24 |
| 14 | 32 |



The cipher takes a plaintext block size of 128 bits, or 16 bytes. The key length can be 16, 24, or 32 bytes (128, 192, or 256 bits). The algorithm is referred to as AES-128, AES-192, or AES-256, depending on the key length. The input to the encryption and decryption algorithms is a single 128-bit block. In FIPS PUB 197, this block is depicted as a 4 * 4 square matrix of bytes. This block is copied into the **State** array, which is modified at each stage of encryption or decryption. After the final stage, **State** is copied to an output matrix. Similarly, the key is depicted as a square matrix of bytes. This key is then expanded into an array of key schedule words. Each word is four bytes, and the total key schedule is 44 words for the 128-bit key. Note that the ordering of bytes within a matrix is by column. So, for example, the first four bytes of a 128-bit plaintext input to the encryption cipher occupy the first column of the **in** matrix, the second four bytes occupy the second column, and so on. Similarly, the first four bytes of the expanded key, which form a word, occupy the first column of the **w** matrix.

The cipher consists of *N* rounds, where the number of rounds depends on the key length: 10 rounds for a 16-byte key, 12 rounds for a 24-byte key, and 14 rounds for a 32-byte key. The first *N* - 1 rounds consist of four distinct transformation functions: SubBytes, ShiftRows, MixColumns, and AddRoundKey, which are described subsequently. The final round contains only three transformations, and there is an initial single transformation (AddRoundKey) before the first round, which can be considered Round 0. Each transformation takes one or more 4 * 4 matrices as input and produces a 4 * 4 matrix as output.

# AES general structure (2/2)



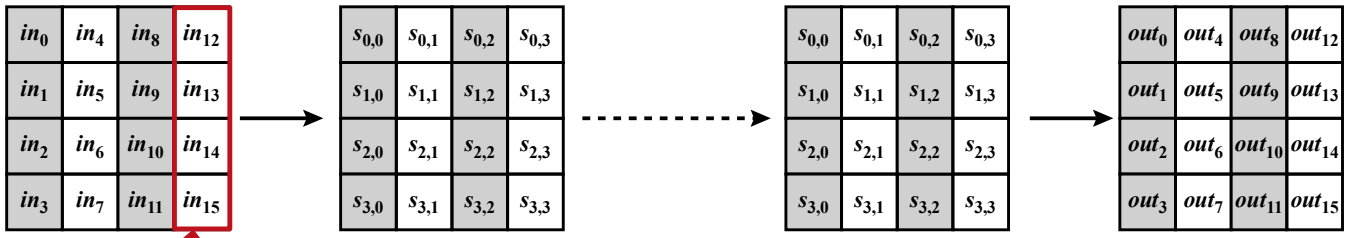

| No. of rounds | Key Length (bytes) |
|---|---|
| 10 | 16 |
| 12 | 24 |
| 14 | 32 |

number of rounds depends on key size

| Key Size (bits) | 128 | 192 | 256 |
|---|---|---|---|
| Plaintext Block Size (bits) | 128 | 128 | 128 |
| Number of Rounds | 10 | 12 | 14 |
| Round Key Size (bits) | 128 | 128 | 128 |
| Expanded Key Size (bytes) | 176 | 208 | 240 |

## The input is represented as a 4x4 byte matrix

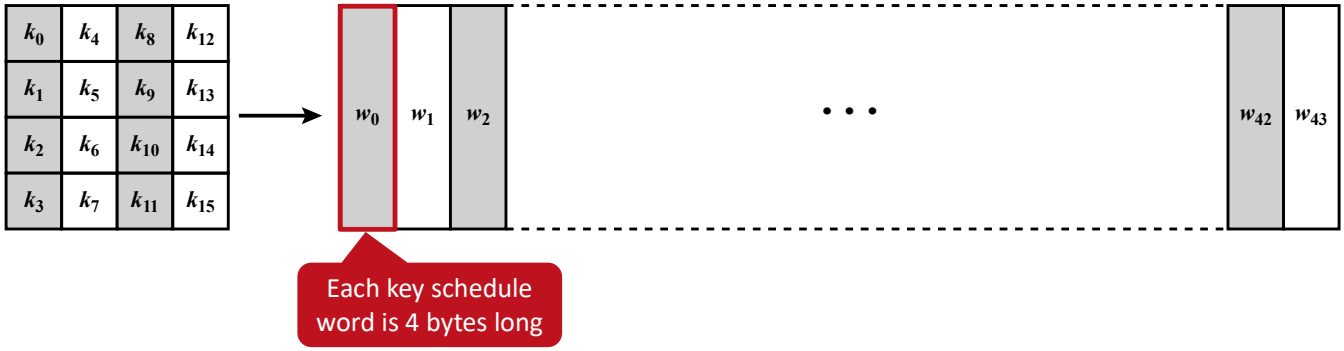
The input is copied into the State array, which is modified at each stage and at the final stage copied into the output matrix




bytes are ordered by column, each column consists of a 4-byte **word**

## The key is also represented by a matrix

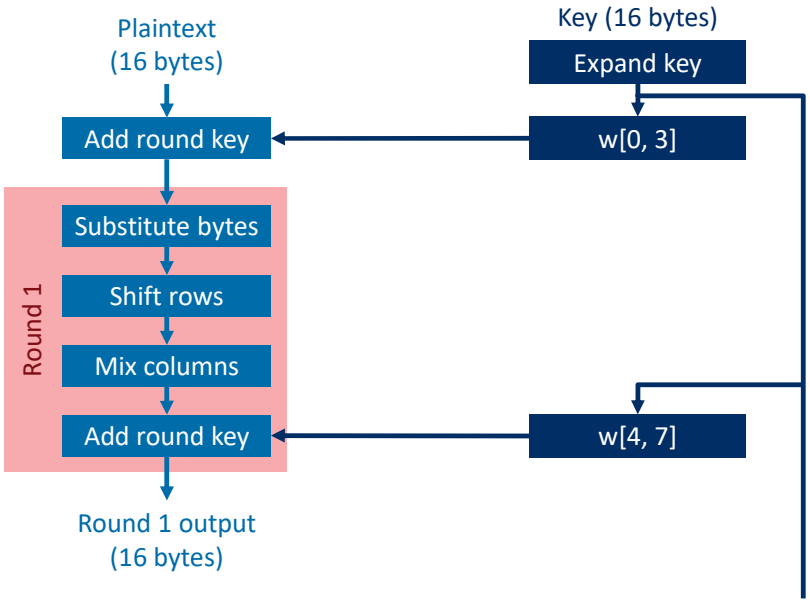
- The key is expanded into an array of key schedule words
- The key is expanded into 44, 52 or 60 words for a 128, 192 and 256-bit key respectively
- Four distinct words are combined into a 128-bit key each round






The key that is provided as input is expanded into an array of forty-four 32-bit words, w[*i*]. Four distinct words (128 bits) serve as a round key for each round.

## Each round consists of 4 steps (except the last)




University of Antwerp
Faculty of Science

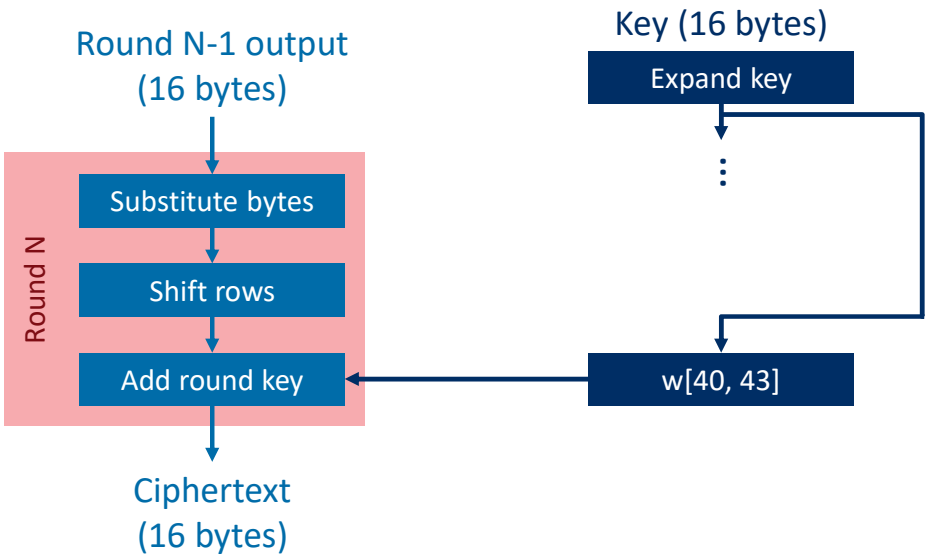
89

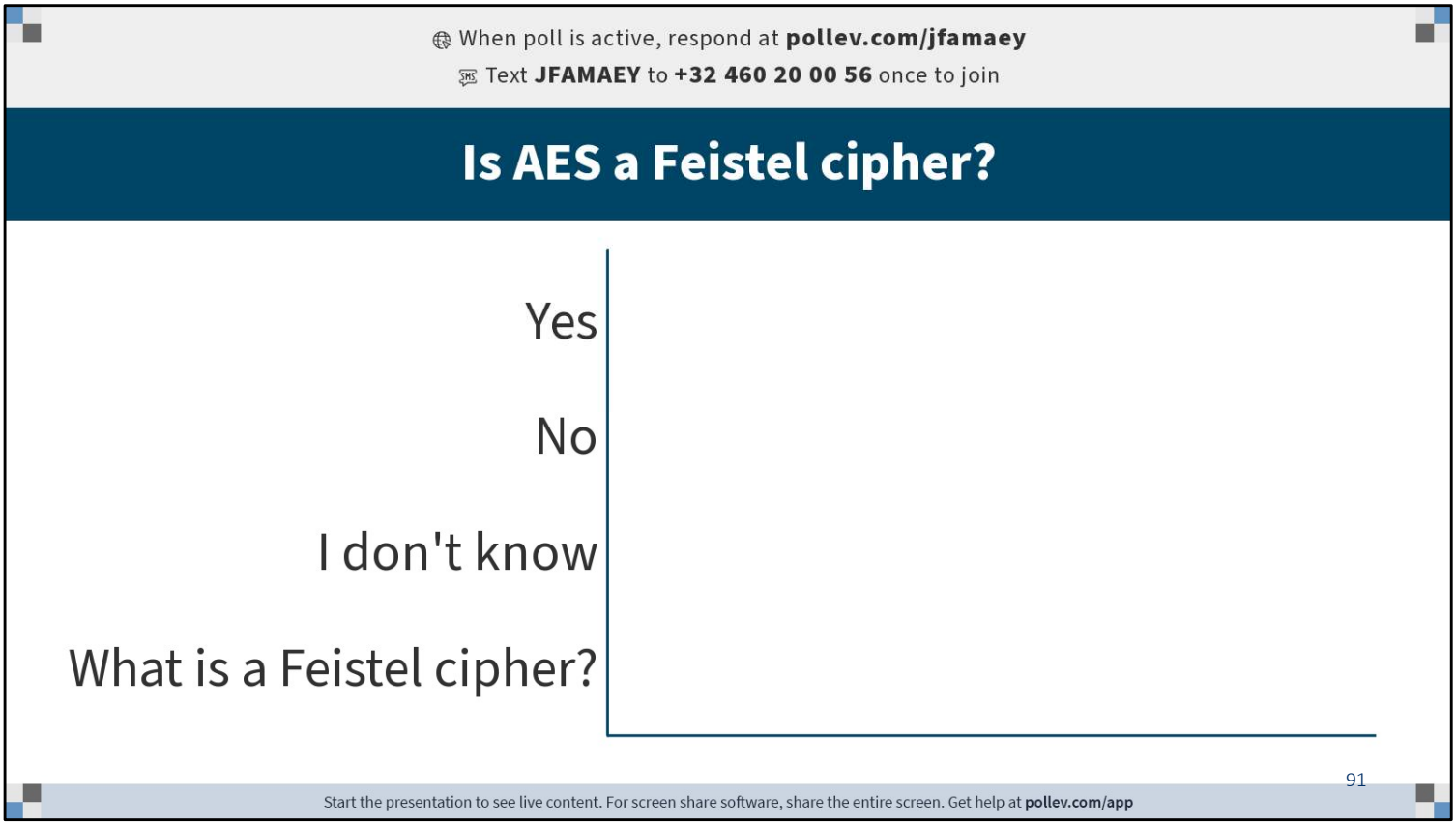
Four different stages are used, one of permutation and three of substitution:

- **Substitute bytes:** Uses an S-box to perform a byte-by-byte substitution of the block.
- **ShiftRows:** A simple permutation.
- **MixColumns:** A substitution that makes use of arithmetic over GF($2^8$).
- **AddRoundKey:** A simple bitwise XOR of the current block with a portion of the expanded key.

The structure is quite simple. For both encryption and decryption, the cipher begins with an AddRoundKey stage, followed by nine rounds that each includes all four stages, followed by a tenth round of three stages. Only the AddRoundKey stage makes use of the key. For this reason, the cipher begins and ends with an AddRoundKey stage. Any other stage, applied at the beginning or end, is reversible without knowledge of the key and so would add no security. The AddRoundKey stage is, in effect, a form of Vernam cipher and by itself would not be formidable. The other three stages together provide confusion, diffusion, and nonlinearity, but by themselves would provide no security because they do not use the key. We can view the cipher as alternating operations of XOR encryption (AddRoundKey) of a block, followed by scrambling of the block (the other three stages), followed by XOR encryption, and so on. This scheme is both efficient and highly secure.

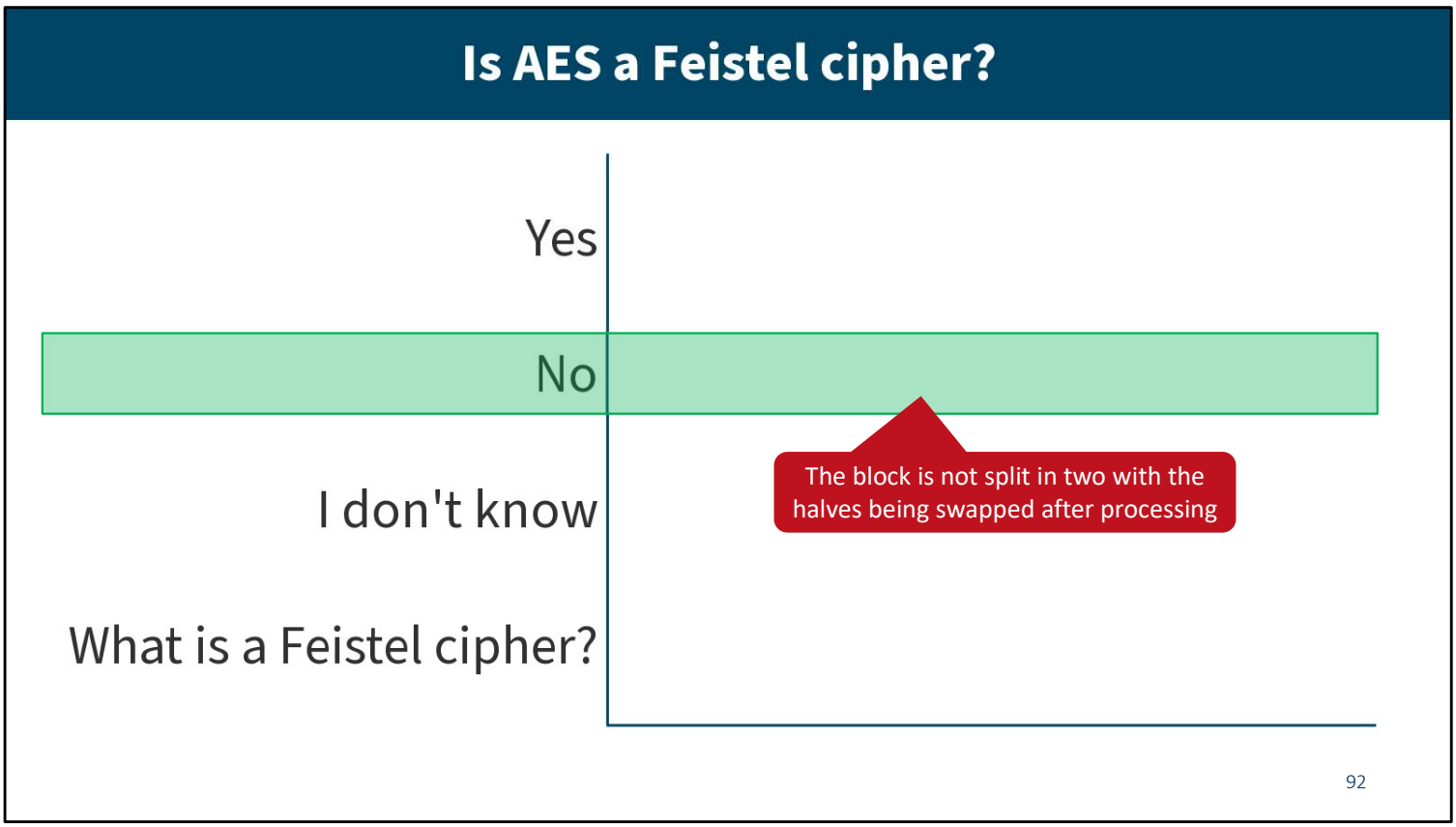
## The last round has only 3 steps (no mix columns)





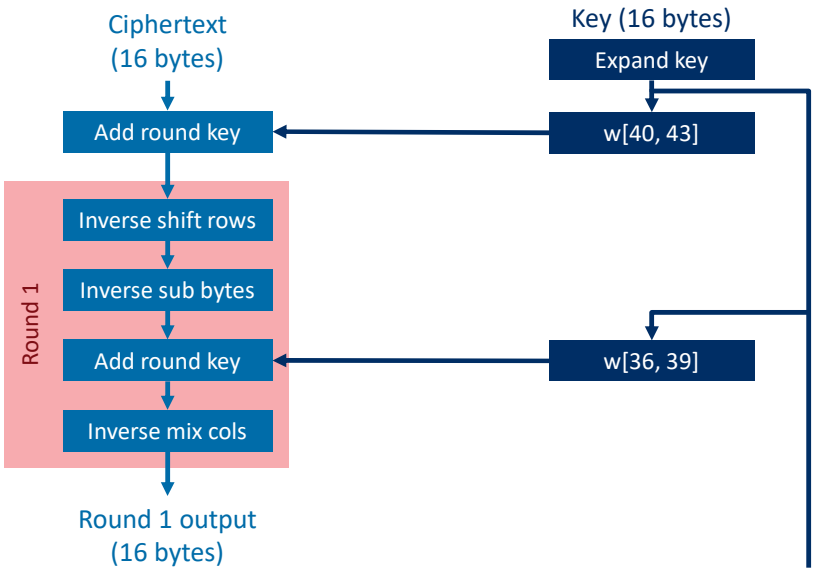

Poll Title: Do not modify the notes in this section to avoid tampering with the Poll Everywhere activity.
More info at polleverywhere.com/support

Is AES a Feistel cipher?
https://www.polleverywhere.com/multiple_choice_polls/LDnGxU6JdQZfjWr?state=opened&flow=Default&onscreen=persist




Poll Title: Is AES a Feistel cipher?

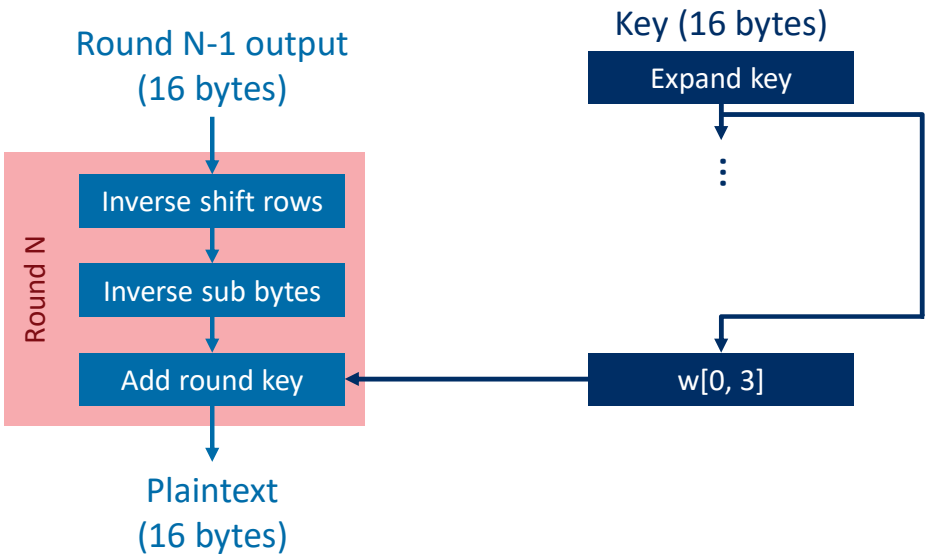
## A general decryption round



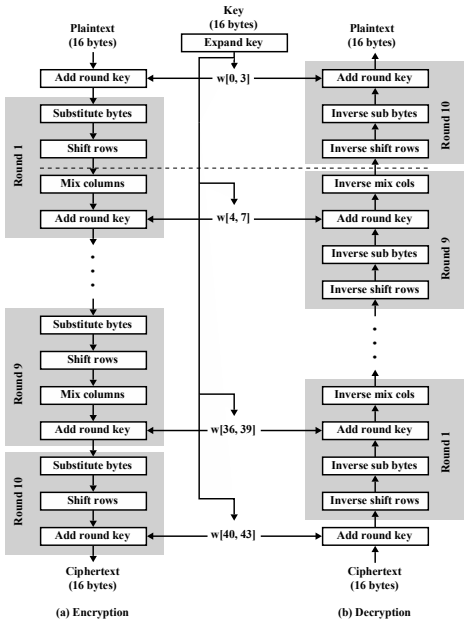



Each stage is easily reversible. For the Substitute Byte, ShiftRows, and MixColumns stages, an inverse function is used in the decryption algorithm. For the AddRoundKey stage, the inverse is achieved by XORing the same round key to the block, using the result that A $\oplus$ B $\oplus$ B = A. As with most block ciphers, the decryption algorithm makes use of the expanded key in reverse order. However, the decryption algorithm is not identical to the encryption algorithm. This is a consequence of the particular structure of AES.

## The last round of decryption



## Encryption and decryption are each other's inverse




University of Antwerp
Faculty of Science

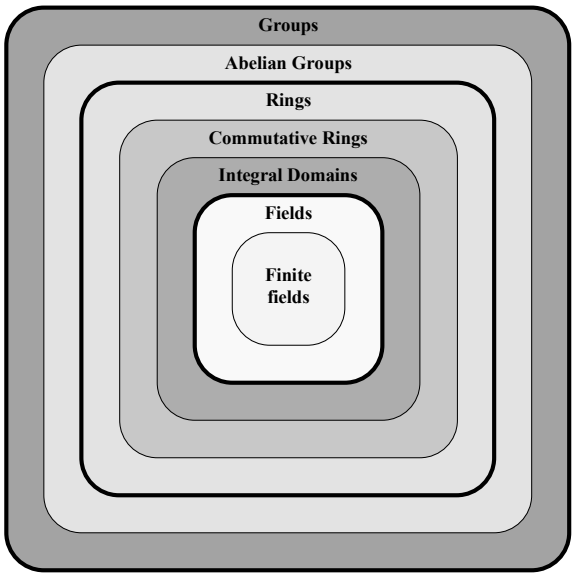
95

Once it is established that all four stages are reversible, it is easy to verify that decryption does recover the plaintext. The figure lays out encryption and decryption going in opposite vertical directions. At each horizontal point (e.g., the dashed line in the figure), **State** is the same for both encryption and decryption.

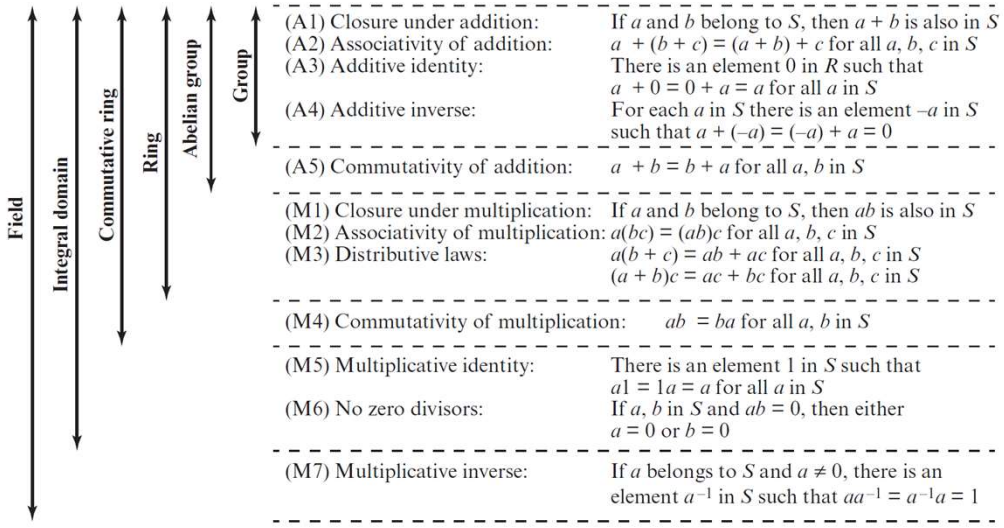
1. General block cipher structure
2. Advanced Encryption Standard (AES)
3. Detour: Finite field arithmetic
4. AES details

University of Antwerp
Faculty of Science

## Groups, rings, and fields



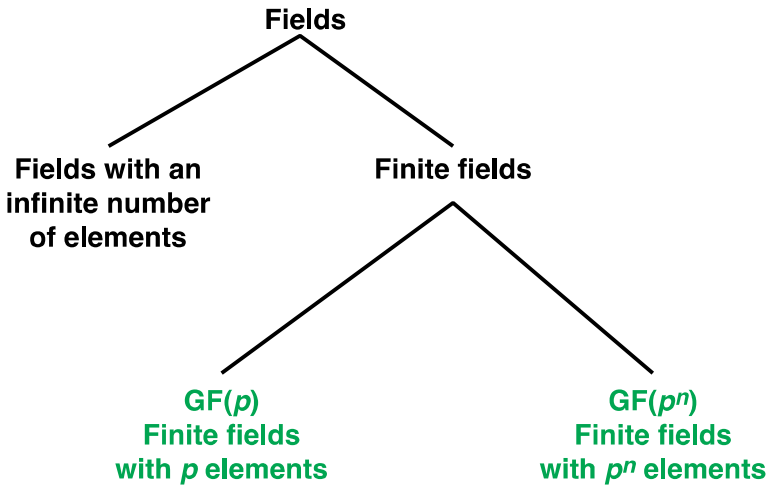
## Properties






- A field F, sometimes denoted by {F, +,* }, is a set of elements with two binary operations, called addition and multiplication, such that for all a, b, c in F the following axioms are obeyed:
  - **(A1–M6)**
  - F is an integral domain; that is, F satisfies axioms A1 through A5 and M1 through M6
  - **(M7) Multiplicative inverse:**
    For each a in F, except 0, there is an element $a^{-1}$ in F such that
    $a\ a^{-1} = (a^{-1})\ a = 1$
- In essence, a field is a set in which we can do addition, subtraction, multiplication, and division without leaving the set. Division is defined with the following rule: $a / b = a\ (b^{-1})$

## Finite fields




University of Antwerp | Faculty of Science

99

- Finite fields play a crucial role in many cryptographic algorithms
- It can be shown that the order of a finite field must be a power of a prime $p^n$, where n is a positive integer
  - The finite field of order $p^n$ is generally written GF($p^n$)
  - GF stands for Galois field, in honour of the mathematician who first studied finite fields
- For a given prime p, we define the finite field of order p, GF(p), as the set $\mathbb{Z}_p = \{0,1,\dots,p-1\}$ of integers together with the arithmetic operations modulo p

# The GF($2^n$) field: $f(x) = \sum_{i=0}^{n-1} a_i x^i$

- Each $a_i$ is a binary value, resulting in $2^n$ polynomials
  - e.g., for n = 3: 0, x, $x^2$, $x^2 + x$, 1, $x + 1$, $x^2 + 1$, and $x^2 + x + 1$
- Arithmetic operation definitions
  - The basic polynomial arithmetic rules are followed, except:
  - Arithmetic on the coefficients is performed modulo 2 (i.e., XOR)
  - If the result of multiplication has a degree larger than $x^{n-1}$ then reduce it modulo some irreducible polynomial* m(x)
- Two possibilities for m(x) when n = 3:
  - $x^3 + x^2 + 1$
  - $x^3 + x + 1$

*A polynomial f(x) over a field F is called **irreducible** if and only if f(x) cannot be expressed as a product of two polynomials, both over F, and both of degree lower than that of f(x).

# Finite fields in AES

- AES requires a set of integers that:
  - Support division (i.e., each nonzero element has a multiplicative inverse)
  - All fit in n bits as to not waste space (i.e., a range from 0 to $2^n$-1)
- The set of integers $\mathbb{Z}_{2^n}$ does not satisfy these requirements:
  - e.g., 2 does not have a multiplicative inverse, i.e., there is no b such that $2b \ mod \ 2^n = 1$
- A finite field of type GF($2^n$) satisfies these requirements
  - It consists of $2^n$ elements, each represented by a polynomial f(x):

$$f(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x + a_0 = \sum_{i=0}^{n-1} a_i x^i$$

  - Where each $a_i$ is a binary value (i.e., 0 or 1), and f(x) thus represents a binary number < $2^n$
- **AES uses GF($2^8$) with m(x) = $x^8 + x^4 + x^3 + x + 1$**

## Addition in GF($2^8$)

Addition is performed as a bit-per-bit XOR

**Example**

$$
\begin{array}{rl}
 & x^7 + x^5 + x^4 + x^3 + x \quad (= 1011\ 1010 = 0xBA) \\
+ & \underline{\qquad\qquad\quad x^3 + x + 1} \quad (= 0000\ 1011 = 0x0B) \\
 & x^7 + x^5 + x^4 + \quad\ 1 \quad (= 1011\ 0001 = 0xB1)
\end{array}
$$

# Multiplication in GF($2^8$)

Multiplication can be performed as polynomial multiplication modulo m(x)

**Example**

$$
\begin{array}{rl}
 & x^7 + x^5 + x^4 + x^3 + x \\
\times & x^3 + x + 1 \\
\hline
 & x^7 + x^5 + x^4 + x^3 + x \\
 & x^8 + x^6 + x^5 + x^4 + x^2 \\
+ & x^{10} + x^8 + x^7 + x^6 + x^4 \\
\hline
 & x^{10} + x^4 + x^3 + x^2 + x
\end{array}
$$

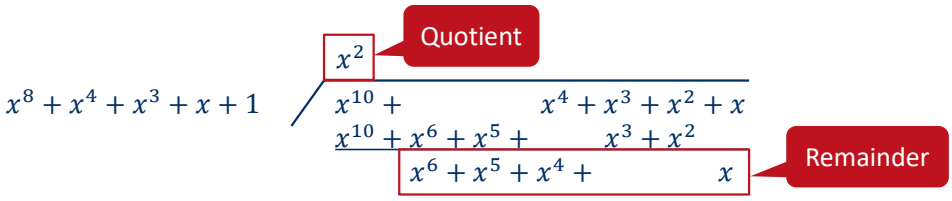
**Problem:** The solution c(x) does not fall within GF($2^8$)

**Solution:** calculate c(x) mod $x^8 + x^4 + x^3 + x + 1$

University of Antwerp
Faculty of Science

# Multiplication in GF($2^8$)

c(x) mod m(x) can be found by calculated the remainder of c(x) / m(x)




**Result**

$a(x) = x^7 + x^5 + x^4 + x^3 + x, \quad b(x) = x^3 + x + 1$

$a(x) \times b(x) = x^6 + x^5 + x^4 + x = 0111\ 0010 = \{72\}$

University of Antwerp
Faculty of Science

# Multiplicative inverse in GF($2^8$)

- The multiplicative inverse of a(x) is $a^{-1}(x)$ if: $a(x) \times a^{-1}(x) \equiv 1 \pmod{m(x)}$
- The value of $a^{-1}(x)$ can be found by using the **Extended Euclidian Algorithm**:

**Initialization**

$r_{-1}(x) = m(x)$ $\quad$ $w_{-1}(x) = 0$

$r_0(x) = a(x)$ $\quad$ $w_0(x) = 1$

**Recursive algorithm**

$r_i(x) = r_{i-2}(x) \bmod r_{i-1}(x)$

$q_i(x) = r_{i-2}(x) / r_{i-1}(x)$

$w_i(x) = w_{i-2}(x) - q_i(x)\, w_{i-1}(x)$

**Stop condition**

if $r_i(x) = 1$ $\quad$ then $w_i(x) = a^{-1}(x)$

## Multiplicative inverse in GF($2^8$)

**Example**

Calculate $a^{-1}(x)$, given $a(x) = 0 \times 22 = 0010\ 0010 = x^5 + x$

**Solution** (using Extended Euclidian Algorithm):

1. $r_1 = x^3 + x + 1, q_1 = x^3, w_1 = x^3$
2. $r_2 = x^2 + 1, q_2 = x^2 + 1, w_2 = x^5 + x^3 + 1$
3. $r_3 = 1, q_3 = x, w_3 = x^6 + x^4 + x^3 + x = 0101\ 1010 = \{5A\} = a^{-1}(x)$

1 General block cipher structure

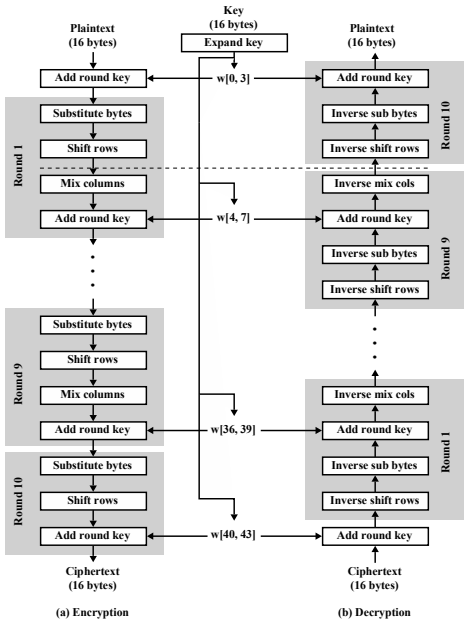
2 Advanced Encryption Standard (AES)

3 Detour: Finite field arithmetic

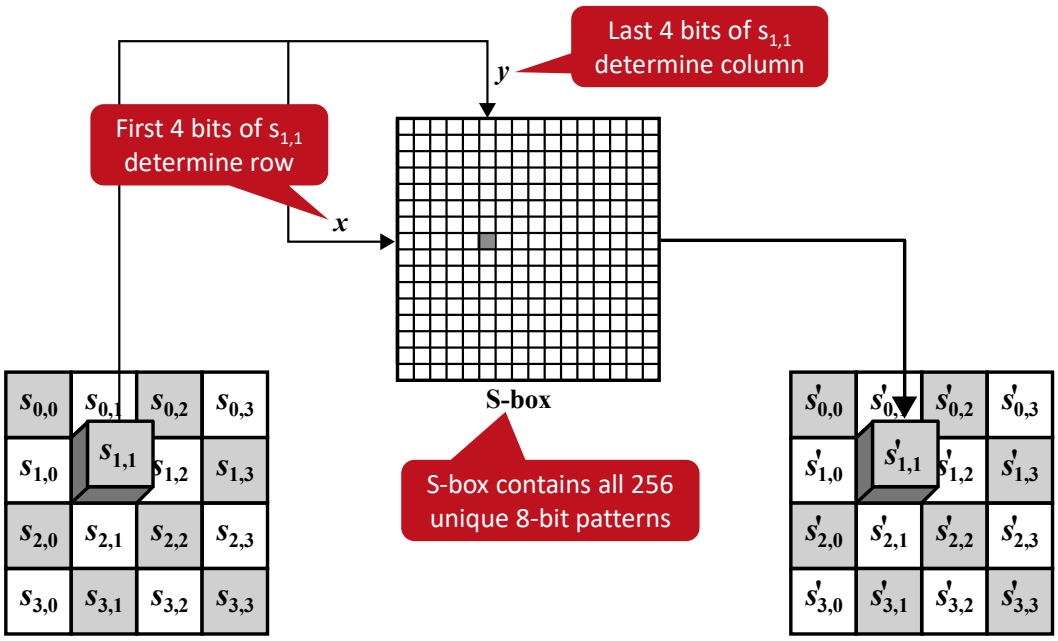
4 AES details

University of Antwerp
Faculty of Science

# Refresher: AES encryption and decryption



## Step 1: SubBytes




University of Antwerp
Faculty of Science

109

The **forward substitute byte transformation**, called SubBytes, is a simple table lookup. AES defines a 16 * 16 matrix of byte values, called an S-box, that contains a permutation of all possible 256 8-bit values. Each individual byte of **State** is mapped into a new byte in the following way: The leftmost 4 bits of the byte are used as a row value and the rightmost 4 bits are used as a column value. These row and column values serve as indexes into the Ss-box to select a unique 8-bit output value. For example, the hexadecimal value {95} references row 9, column 5 of the S-box, which contains the value {2A}. Accordingly, the value {95} is mapped into the value {2A}.

# S-box and inverse S-box

| x \ y | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 63 | 7C | 77 | 7B | F2 | 6B | 6F | C5 | 30 | 01 | 67 | 2B | FE | D7 | AB | 76 |
| 1 | CA | 82 | C9 | 7D | FA | 59 | 47 | F0 | AD | D4 | A2 | AF | 9C | A4 | 72 | C0 |
| 2 | B7 | FD | 93 | 26 | 36 | 3F | F7 | CC | 34 | A5 | E5 | F1 | 71 | D8 | 31 | 15 |
| 3 | 04 | C7 | 23 | C3 | 18 | 96 | 05 | 9A | 07 | 12 | 80 | E2 | EB | 27 | B2 | 75 |
| 4 | 09 | 83 | 2C | 1A | 1B | 6E | 5A | A0 | 52 | 3B | D6 | B3 | 29 | E3 | 2F | 84 |
| 5 | 53 | D1 | 00 | ED | 20 | FC | B1 | 5B | 6A | CB | BE | 39 | 4A | 4C | 58 | CF |
| 6 | D0 | EF | AA | FB | 43 | 4D | 33 | 85 | 45 | F9 | 02 | 7F | 50 | 3C | 9F | A8 |
| 7 | 51 | A3 | 40 | 8F | 92 | 9D | 38 | F5 | BC | B6 | DA | 21 | 10 | FF | F3 | D2 |
| 8 | CD | 0C | 13 | EC | 5F | 97 | 44 | 17 | C4 | A7 | 7E | 3D | 64 | 5D | 19 | 73 |
| 9 | 60 | 81 | 4F | DC | 22 | 2A | 90 | 88 | 46 | EE | B8 | 14 | DE | 5E | 0B | DB |
| A | E0 | 32 | 3A | 0A | 49 | 06 | 24 | 5C | C2 | D3 | AC | 62 | 91 | 95 | E4 | 79 |
| B | E7 | C8 | 37 | 6D | 8D | D5 | 4E | A9 | 6C | 56 | F4 | EA | 65 | 7A | AE | 08 |
| C | BA | 78 | 25 | 2E | 1C | A6 | B4 | C6 | E8 | DD | 74 | 1F | 4B | BD | 8B | 8A |
| D | 70 | 3E | B5 | 66 | 48 | 03 | F6 | 0E | 61 | 35 | 57 | B9 | 86 | C1 | 1D | 9E |
| E | E1 | F8 | 98 | 11 | 69 | D9 | 8E | 94 | 9B | 1E | 87 | E9 | CE | 55 | 28 | DF |
| F | 8C | A1 | 89 | 0D | BF | E6 | 42 | 68 | 41 | 99 | 2D | 0F | B0 | 54 | BB | 16 |

S-box

S-box(29) = A5
IS-box(A5) = 29

Inverse S-box (IS-box)

| x \ y | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 52 | 09 | 6A | D5 | 30 | 36 | A5 | 38 | BF | 40 | A3 | 9E | 81 | F3 | D7 | FB |
| 1 | 7C | E3 | 39 | 82 | 9B | 2F | FF | 87 | 34 | 8E | 43 | 44 | C4 | DE | E9 | CB |
| 2 | 54 | 7B | 94 | 32 | A6 | C2 | 23 | 3D | EE | 4C | 95 | 0B | 42 | FA | C3 | 4E |
| 3 | 08 | 2E | A1 | 66 | 28 | D9 | 24 | B2 | 76 | 5B | A2 | 49 | 6D | 8B | D1 | 25 |
| 4 | 72 | F8 | F6 | 64 | 86 | 68 | 98 | 16 | D4 | A4 | 5C | CC | 5D | 65 | B6 | 92 |
| 5 | 6C | 70 | 48 | 50 | FD | ED | B9 | DA | 5E | 15 | 46 | 57 | A7 | 8D | 9D | 84 |
| 6 | 90 | D8 | AB | 00 | 8C | BC | D3 | 0A | F7 | E4 | 58 | 05 | B8 | B3 | 45 | 06 |
| 7 | D0 | 2C | 1E | 8F | CA | 3F | 0F | 02 | C1 | AF | BD | 03 | 01 | 13 | 8A | 6B |
| 8 | 3A | 91 | 11 | 41 | 4F | 67 | DC | EA | 97 | F2 | CF | CE | F0 | B4 | E6 | 73 |
| 9 | 96 | AC | 74 | 22 | E7 | AD | 35 | 85 | E2 | F9 | 37 | E8 | 1C | 75 | DF | 6E |
| A | 47 | F1 | 1A | 71 | 1D | 29 | C5 | 89 | 6F | B7 | 62 | 0E | AA | 18 | BE | 1B |
| B | FC | 56 | 3E | 4B | C6 | D2 | 79 | 20 | 9A | DB | C0 | FE | 78 | CD | 5A | F4 |
| C | 1F | DD | A8 | 33 | 88 | 07 | C7 | 31 | B1 | 12 | 10 | 59 | 27 | 80 | EC | 5F |
| D | 60 | 51 | 7F | A9 | 19 | B5 | 4A | 0D | 2D | E5 | 7A | 9F | 93 | C9 | 9C | EF |
| E | A0 | E0 | 3B | 4D | AE | 2A | F5 | B0 | C8 | EB | BB | 3C | 83 | 53 | 99 | 61 |
| F | 17 | 2B | 04 | 7E | BA | 77 | D6 | 26 | E1 | 69 | 14 | 63 | 55 | 21 | 0C | 7D |

## An example of substitute bytes transformation

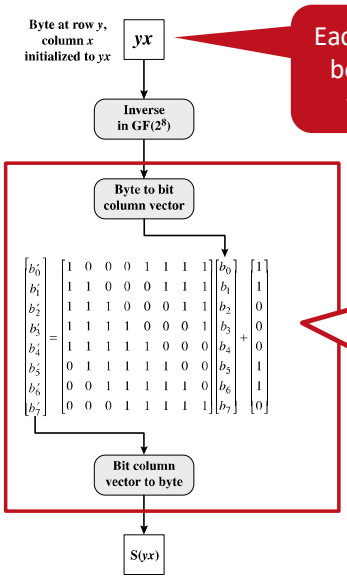
Both x (row) and y (column) of the 8-bit input can be transformed into a two-digit hexadecimal number (4 bits), where each digit determines the row or column, which in turn are each also represented by a single digit hexadecimal number

| | | | |
|---|---|---|---|
| EA | 04 | 65 | 85 |
| 83 | 45 | 5D | 96 |
| 5C | 33 | 98 | B0 |
| F0 | 2D | AD | C5 |

→

| | | | |
|---|---|---|---|
| 87 | F2 | 4D | 97 |
| EC | 6E | 4C | 90 |
| 4A | C3 | 46 | E7 |
| 8C | D8 | 95 | A6 |

## S-box is not arbitrary and can be calculated!




Each of the 256 values of the S-box is initialized to yx, with y the row and x the column

Consider the byte consisting of 8 labelled bits: $(b_7b_6b_5b_4b_3b_2b_1b_0)$

$$b_i' = b_i \oplus b_{(i+4)\ mod\ 8} \oplus b_{(i+5)\ mod\ 8} \oplus b_{(i+6)\ mod\ 8} \oplus b_{(i+7)\ mod\ 8} \oplus c_i$$

with $c = (c_7c_6c_5c_4c_3c_2c_1c_0) = (01100011)$

University of Antwerp | Faculty of Science

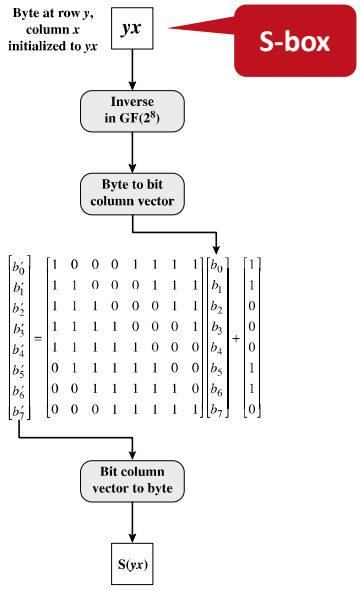
112

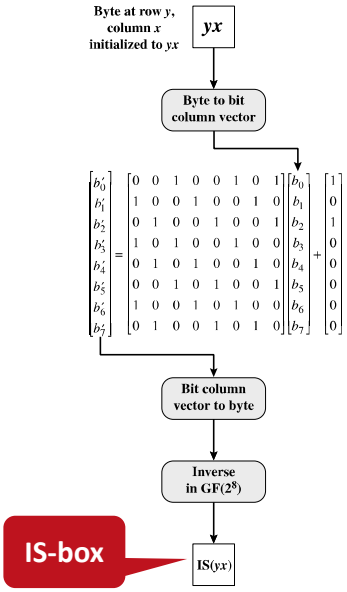
1. Initialize the S-box with the byte values in ascending sequence row by row. The first row contains {00}, {01}, {02},... , {0F}; the second row contains {10}, {11}, etc.; and so on. Thus, the value of the byte at row *y*, column *x* is {*yx*}.
2. Map each byte in the S-box to its multiplicative inverse in the finite field GF($2^8$); the value {00} is mapped to itself.
3. Consider that each byte in the S-box consists of 8 bits labeled ($b_7$, $b_6$, $b_5$, $b_4$, $b_3$, $b_2$, $b_1$, $b_0$). Apply the following transformation to each bit of each byte in the S-box:

   $b_i' = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i$

   where $c_i$ is the $i^{th}$ bit of byte *c* with the value {63}; that is, $(c_7c_6c_5c_4c_3c_2c_1c_0)$ = (01100011). The prime (′) indicates that the variable is to be updated by the value on the right. The AES standard depicts this transformation in matrix form as follows.

## IS-box can be similarly calculated



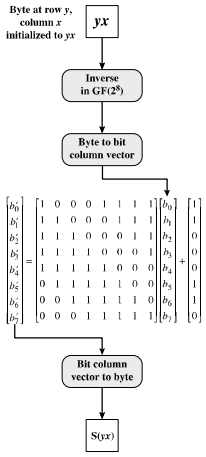

University of Antwerp
Faculty of Science

113

The **inverse substitute byte transformation**, called InvSubBytes, makes use of the inverse S-box. Note, for example, that the input {2A} produces the output {95}, and the input {95} to the S-box produces {2A}. The inverse S-box is constructed by applying the inverse of the transformation followed by taking the multiplicative inverse in GF($2^8$).

# Exercise: Calculate S-box element

**Exercise:** Calculate the S-box element at row 9 and column 5, or yx = {95} = (1001 0101)




Consider the byte consisting of 8 labelled bits:

$(b_7b_6b_5b_4b_3b_2b_1b_0)$

$$b_i' = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i$$

with $c = (c_7c_6c_5c_4c_3c_2c_1c_0) = (01100011)$

$$m(x) = x^8 + x^4 + x^3 + x + 1$$

**Extended Euclidian Algorithm**

*Initialization*

$r_{-1}(x) = m(x)$ $\quad w_{-1}(x) = 0$

$r_0(x) = a(x)$ $\quad w_0(x) = 1$

*Recursive algorithm*

$r_i(x) = r_{i-2}(x) \bmod r_{i-1}(x)$

$q_i(x) = r_{i-2}(x) / r_{i-1}(x)$

$w_i(x) = w_{i-2}(x) - q_i(x)\, w_{i-1}(x)$

*Stop condition*

if $r_i(x) = 1$ then $w_i(x) = a^{-1}(x)$

# Solution: Calculate S-box element solution (1/2)

**Solution**

Step 1: calculate multiplicative inverse of $\{95\} = 1001\ 0101 = x^7 + x^4 + x^2 + 1$

$r_{-1}(x) = m(x) = x^8 + x^4 + x^3 + x + 1, r_0(x) = a(x) = x^7 + x^4 + x^2 + 1$
$w_{-1}(x) = 0, w_0(x) = 1$

$r_1(x) = x^5 + x^4 + 1, q_1(x) = x, w_1(x) = x$

$r_2(x) = x, q_2(x) = x^2 + x + 1, w_2(x) = x^3 + x^2 + x + 1$

$r_3(x) = 1, q_3(x) = x^4 + x^3$ $w_3(x) = x^7 + x^3 + x$

$\{95\}^{-1} = (1000\ 1010) = \{8A\}$

# Exercise: Calculate S-box element solution (2/2)

**Solution**

Step 2: matrix transformation for b = {8A} = (10001010)

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \boxed{\{2A\}}$$

## Rationale for S-box generation algorithm

**Designed to be resistant to known cryptanalytic attacks:**

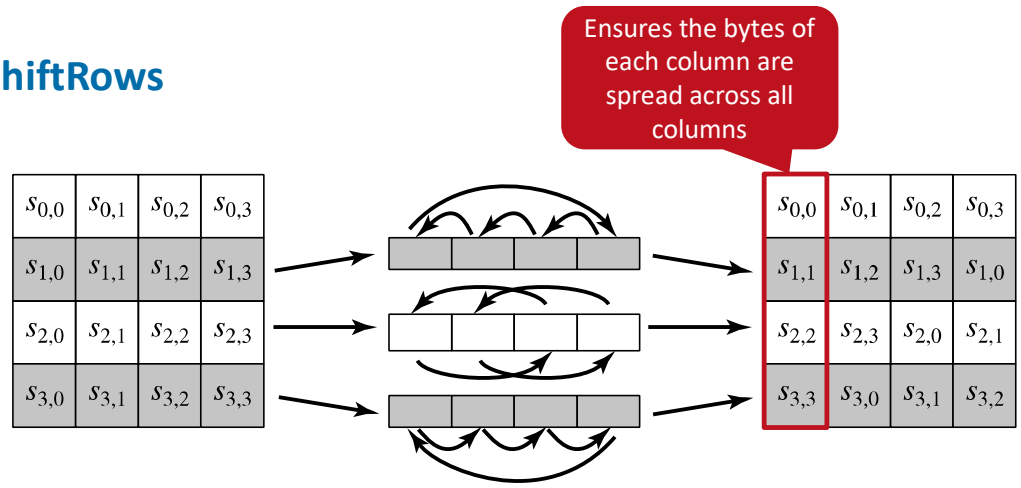
- **Low correlation** between input and output bits
- Output is **not a linear** mathematical function of input
- S-box has **no fixed points**: S-box(a) ≠ a
- S-box has no opposite fixed points: S-box(a) ≠ $\bar{a}$ (with $\bar{a}$ bitwise complement of a)
- S-box is **invertible**: IS-box(S-box(a)) = a
- S-box does **not self-inverse**: S-box(a) ≠ IS-box(a)

$\bar{a}$ = bitwise complement of a



The S-box is designed to be resistant to known cryptanalytic attacks. Specifically, the Rijndael developers sought a design that has a low correlation between input bits and output bits and the property that the output is not a linear mathematical function of the input. The nonlinearity is due to the use of the multiplicative inverse. In addition, the constant c was chosen so that the S-box has no fixed points [S-box($a$) = $a$] and no "opposite fixed points" [S-box($a$) = $a'$], where $a'$ is the bitwise complement of $a$. Of course, the S-box must be invertible, that is, IS-box[S-box($a$)] = $a$. However, the S-box does not self-inverse in the sense that it is not true that S-box($a$) = IS-box($a$). For example, S-box({95}) = {2A}, but IS-box({95}) = {AD}.

## Step 2: ShiftRows




**Example**

| | | | |
|---|---|---|---|
| 87 | F2 | 4D | 97 |
| EC | 6E | 4C | 90 |
| 4A | C3 | 46 | E7 |
| 8C | D8 | 95 | A6 |

→

| | | | |
|---|---|---|---|
| 87 | F2 | 4D | 97 |
| 6E | 4C | 90 | EC |
| 46 | E7 | 4A | C3 |
| A6 | 8C | D8 | 95 |

University of Antwerp | Faculty of Science

118

The first row of **State** is not altered. For the second row, a 1-byte circular left shift is performed. For the third row, a 2-byte circular left shift is performed. For the fourth row, a 3-byte circular left shift is per- formed. The following is an example of ShiftRows.

The **inverse shift row transformation**, called InvShiftRows, performs the circular shifts in the opposite direction for each of the last three rows, with a 1-byte circular right shift for the second row, and so on.

The shift row transformation is more substantial than it may first appear. This is because the **State**, as well as the cipher input and output, is treated as an array of four 4-byte columns. Thus, on encryption, the first 4 bytes of the plaintext are copied to the first column of **State**, and so on. Furthermore, as will be seen, the round key is applied to **State** column by column. Thus, a row shift moves an individual byte from one column to another, which is a linear distance of a multiple of 4 bytes. Also note that the transformation ensures that the 4 bytes of one column are spread out to four different columns.

## Step 3: MixColumns

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

Small hexadecimal values for fast calculation

16-byte input state

Sum of products performed in GF($2^8$)

The MixColumns transformation on a column can be expressed as:

$$s'_{0,j} = (2 \cdot s_{0,j}) \oplus (3 \cdot s_{1,j}) \oplus s_{2,j} \oplus s_{3,j}$$
$$s'_{1,j} = s_{0,j} \oplus (2 \cdot s_{1,j}) \oplus (3 \cdot s_{2,j} \oplus s_{3,j}$$
$$s'_{2,j} = s_{0,j} \oplus s_{1,j} \oplus (2 \cdot s_{2,j}) \oplus (3 \cdot s_{3,j})$$
$$s'_{3,j} = (3 \cdot s_{0,j}) \oplus s_{1,j} \oplus s_{2,j} \oplus (2 \cdot s_{3,j})$$

University of Antwerp | Faculty of Science

119

The **forward mix column transformation**, called MixColumns, operates on each column individually. Each byte of a column is mapped into a new value that is a function of all four bytes in that column. Each element in the product matrix is the sum of products of elements of one row and one column. In this case, the individual additions and multiplications are performed in GF($2^8$).

The coefficients of the matrix are based on a linear code with maximal distance between code words, which ensures a good mixing among the bytes of each column. The mix column transformation combined with the shift row transformation ensures that after a few rounds all output bits depend on all input bits. In addition, the choice of coefficients in MixColumns, which are all {01}, {02}, or {03}, was influenced by implementation considerations. Multiplication by these coefficients involves at most a shift and an XOR. The coefficients in InvMixColumns are more formidable to implement. However, encryption was deemed more important than decryption for two reasons:

1. For the CFB and OFB cipher modes, only encryption is used.
2. As with any block cipher, AES can be used to construct a message authentication code, and for this, only encryption is used.

# Exercise: Calculating the MixColumns output state

**Exercise:** Calculate the output state element $s'_{0,0}$ of the MixColumns transformation given the input state below.

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

Input state:

| | | | |
|---|---|---|---|
| 87 | F2 | 4D | 97 |
| 6E | 4C | 90 | EC |
| 46 | E7 | 4A | C3 |
| A6 | 8C | D8 | 95 |

$$s'_{0,j} = (2 \cdot s_{0,j}) \oplus (3 \cdot s_{1,j}) \oplus s_{2,j} \oplus s_{3,j}$$
$$s'_{1,j} = s_{0,j} \oplus (2 \cdot s_{1,j}) \oplus (3 \cdot s_{2,j} \oplus s_{3,j}$$
$$s'_{2,j} = s_{0,j} \oplus s_{1,j} \oplus (2 \cdot s_{2,j}) \oplus (3 \cdot s_{3,j})$$
$$s'_{3,j} = (3 \cdot s_{0,j}) \oplus s_{1,j} \oplus s_{2,j} \oplus (2 \cdot s_{3,j})$$

# Solution: Calculating the MixColumns output state

Input state:

| | | | |
|---|---|---|---|
| 87 | F2 | 4D | 97 |
| 6E | 4C | 90 | EC |
| 46 | E7 | 4A | C3 |
| A6 | 8C | D8 | 95 |

Full solution:

| | | | |
|---|---|---|---|
| 47 | 40 | A3 | 4C |
| 37 | D4 | 70 | 9F |
| 94 | E4 | 3A | 42 |
| ED | A5 | A6 | BC |

**Solution**

$$s'_{0,0} = x \times (x^7 + x^2 + x + 1) + (x + 1) \times (x^6 + x^5 + x^3 + x^2 + x) + (x^6 + x^2 + x) + (x^7 + x^5 + x^2 + x)$$

$$s'_{0,0} = (x^8 + x^3 + x^2 + x) + (x^7 + x^6 + x^4 + x^3 + x^2) + (x^6 + x^5 + x^3 + x^2 + x) + (x^6 + x^2 + x) + (x^7 + x^5 + x^2 + x)$$

$$s'_{0,0} = x^8 + x^6 + x^4 + x^3 + x^2 \bmod m(x)$$

$$s'_{0,0} = x^6 + x^2 + x + 1 = 0100\ 0111 = \{47\}$$

## Inverse MixColumns step

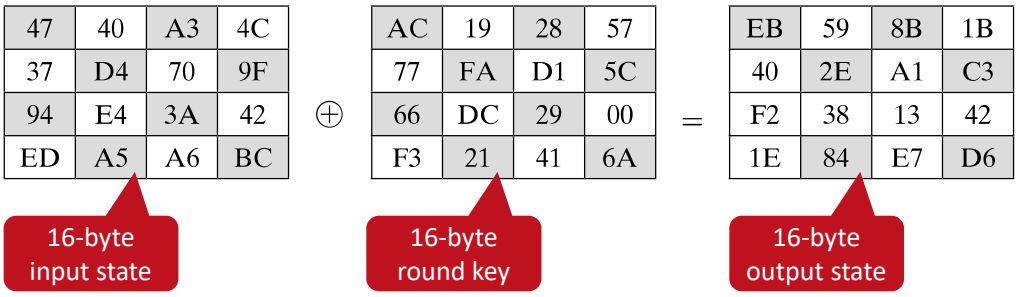
$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

Larger hexadecimal values, means decryption is computationally harder than encryption

## Step 4: AddRoundKey

Bitwise XOR of 128-bit input state with 128-bit round key

Example




University of Antwerp
Faculty of Science

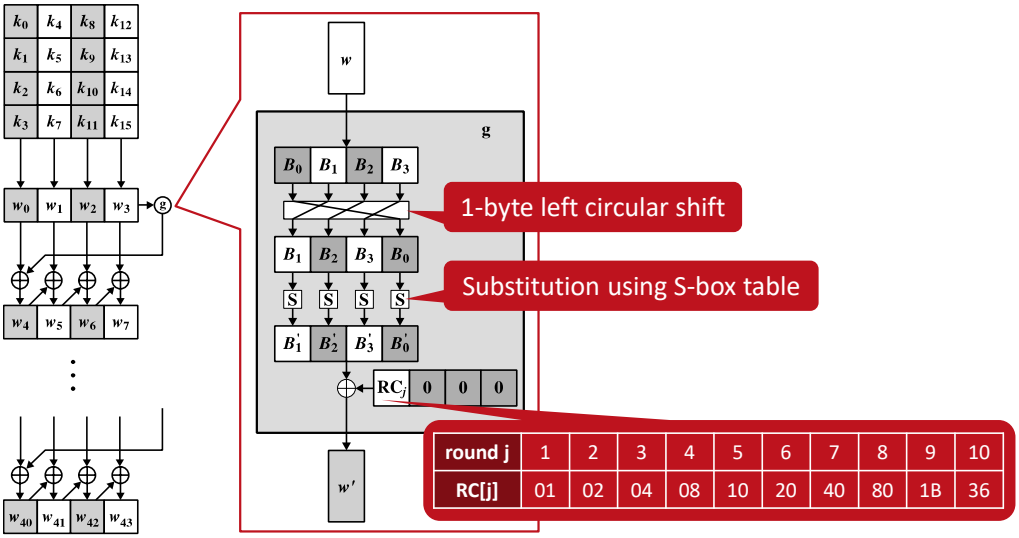
123

In the **forward add round key transformation**, called AddRoundKey, the 128 bits of **State** are bitwise XORed with the 128 bits of the round key. The operation is viewed as a column-wise operation between the 4 bytes of a **State** column and one word of the round key; it can also be viewed as a byte-level operation.

The add round key transformation is as simple as possible and affects every bit of **State**. The complexity of the round key expansion, plus the complexity of the other stages of AES, ensure security.

## AES key expansion

Transform key into linear array of 44, 52 or 60 words




University of Antwerp
Faculty of Science

124

The AES key expansion algorithm takes as input a four-word (16-byte) key and produces a linear array of 44 words (176 bytes). This is sufficient to provide a four-word round key for the initial AddRoundKey stage and each of the 10 rounds of the cipher. The AES key expansion algorithm takes as input a four-word (16-byte) key and produces a linear array of 44 words (176 bytes). This is sufficient to provide a four-word round key for the initial AddRoundKey stage and each of the 10 rounds of the cipher.

1. RotWord performs a one-byte circular left shift on a word. This means that an input word [B0, B1, B2, B3] is transformed into [B1, B2, B3, B0].
2. SubWord performs a byte substitution on each byte of its input word, using the S-box.
3. The result of steps 1 and 2 is XORed with a round constant, Rcon[j].

The round constant is a word in which the three rightmost bytes are always 0. Thus, the effect of an XOR of a word with Rcon is to only perform an XOR on the leftmost byte of the word. The round constant is different for each round and is defined as Rcon[j] = (RC[j], 0, 0, 0), with RC[1] = 1, RC[j] = 2 * RC[j - 1] and with multiplication defined over the field GF($2^8$).

## Rationale for key generation algorithm

- Knowledge of a part of the cipher key or round keys does not enable calculation of many other round-key bits
- The transformation is invertible
- Computationally efficient on many processors
- The round constants RC[j] eliminate symmetry between round keys
- Each cipher key bit affects many round key bits (diffusion)
- Nonlinear to prohibit determination of round key differences based on cipher key differences
- Simple description



The Rijndael developers designed the expansion key algorithm to be resistant to known cryptanalytic attacks. The inclusion of a round-dependent round constant eliminates the symmetry, or similarity, between the ways in which round keys are generated in different rounds. The specific criteria that were used are:

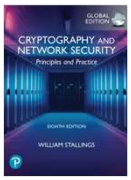
- Knowledge of a part of the cipher key or round key does not enable calculation of many other round-key bits.
- An invertible transformation [i.e., knowledge of any *Nk* consecutive words of the expanded key enables regeneration of the entire expanded key (*Nk* = key size in words)].
- Speed on a wide range of processors.
- Usage of round constants to eliminate symmetries.
- Diffusion of cipher key differences into the round keys; that is, each key bit affects many round key bits.
- Enough nonlinearity to prohibit the full determination of round key differences from cipher key differences only.
- Simplicity of description.

The authors do not quantify the first point on the preceding list, but the idea is that if you know less than *Nk* consecutive words of either the cipher key or one of the round keys, then it is difficult to reconstruct the remaining unknown bits. The fewer bits one knows, the more difficult it is to do the reconstruction or to determine other bits in the key expansion.

## Summary of symmetric ciphers

| Cipher | Type | Characteristics |
|---|---|---|
| Caesar | Substitution | Extremely vulnerable to **brute force attacks** |
| Monoalphabetic | Substitution | Vulnerable to statistical **frequency attacks** |
| Vigenère | Substitution | Limited vulnerability to **frequency attacks** |
| One-time pad | Substitution | **Unconditionally secure**, but practically infeasible |
| RC4 | Substitution | Not secure due to **bias in PRG** |
| Salsa20 | Substitution | **Computationally secure** with key size of 256 bits |
| DES | Hybrid | Vulnerable to **brute force** due to 56-bit key size |
| Triple DES | Hybrid | **Computationally secure** with key size of 168 bits |
| AES | Hybrid | **Computationally secure** |

# Further reading on block ciphers




- Chapter 4 (Section 4.1, 4.2)
- Chapter 5 (Sections 5.1 – 5.7) [optional]
- Chapter 6 (Section 6.1—6.4)
- Chapter 7 (Section 7.1)


- Block ciphers (videos 1, 2 and 3)

# Block operation modes

## NIST modes of operation for block ciphers

- When the plaintext size is larger than the block size B, the text is split into fixed-length blocks, all encrypted with the same key
  - Encryption of multiple blocks with the same key reduces cryptographic security
- NIST defined block cipher operation modes to solve this issue
  - 1981: NIST defined ECB, CBC, OFB and CFB for DES in FIPS 81 ("DES Modes of Operation")
  - 2001: Support for CTR mode and AES in SP800-38A ("Recom. for Block Cipher Modes of Operation")
  - 2010: Support for XTS-AES in SP800-38E ("The XTS-AES Mode for Confidentiality on Storage Devices")
- Modes can in principle be applied to any type of block cipher

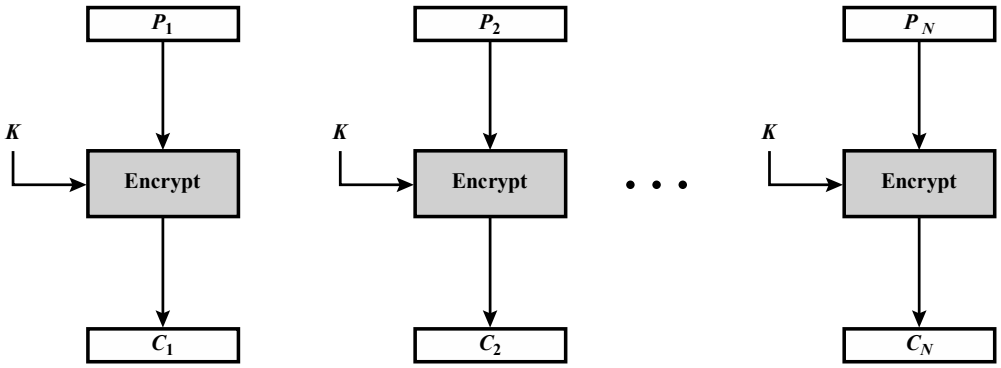
| Mode | Typical Application |
|---|---|
| Electronic Codebook (ECB) | • Secure transmission of one value (e.g., encryption key) |
| Cipher Block Chaining (CBC) | • General-purpose block-oriented transmission<br>• Authentication |
| Cipher Feedback (CFB) | • General-purpose stream-oriented transmission<br>• Authentication |
| Output Feedback (OFB) | • Stream-oriented transmission over noisy channel (e.g., satellite communication) |
| Counter (CTR) | • General-purpose block-oriented transmission<br>• Useful for high-speed requirements |
| XEX-based Tweaked-codebook mode with ciphertext Stealing (XTS-AES) | • Developed specifically for sector-based storage devices<br>• Protects against adversaries that have access to the stored data |



A block cipher takes a fixed-length block of text of length *b* bits and a key as input and produces a *b*-bit block of ciphertext. If the amount of plaintext to be encrypted is greater than *b* bits, then the block cipher can still be used by breaking the plain- text up into *b*-bit blocks. When multiple blocks of plaintext are encrypted using the same key, a number of security issues arise. To apply a block cipher in a variety of applications, five *modes of operation* have been defined by NIST (SP 800-38A). In essence, a mode of operation is a technique for enhancing the effect of a cryptographic algorithm or adapting the algorithm for an application, such as applying a block cipher to a sequence of data blocks or a data stream. The five modes are intended to cover a wide variety of applications of encryption for which a block cipher could be used. These modes are intended for use with any symmetric block cipher, including triple DES and AES.

## Electronic codebook (ECB)

Simplest form, where each block of plaintext is encoded independently using the same key.





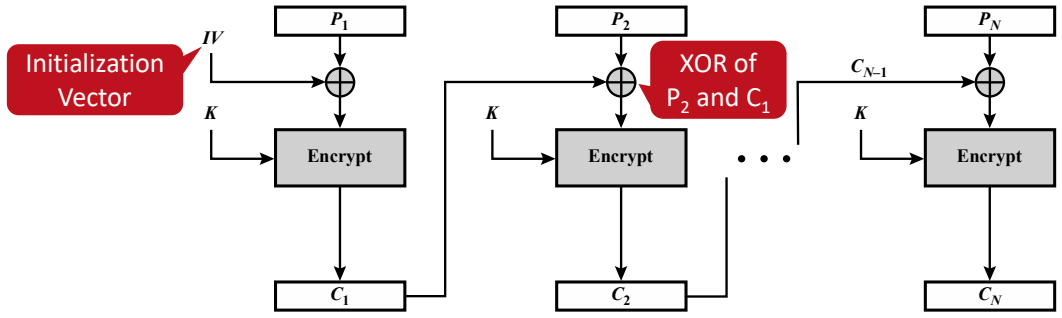

The simplest mode is the **electronic codebook (ECB)** mode, in which plaintext is handled one block at a time and each block of plaintext is encrypted using the same key. The term *codebook* is used because, for a given key, there is a unique ciphertext for every *b*-bit block of plaintext. Therefore, we can imagine a gigantic codebook in which there is an entry for every possible *b*-bit plaintext pattern showing its corresponding ciphertext. For a message longer than *b* bits, the procedure is simply to break the message into *b*-bit blocks, padding the last block if necessary. Decryption is performed one block at a time, always using the same key.

The ECB mode should be used only to secure messages shorter than a single block of underlying cipher (i.e., 64 bits for 3DES and 128 bits for AES), such as to encrypt a secret key. Because in most of the cases messages are longer than the encryption block mode, this mode has a minimum practical value.
The most significant characteristic of ECB is that if the same *b*-bit block of plaintext appears more than once in the message, it always produces the same ciphertext.

For lengthy messages, the ECB mode may not be secure. If the message is highly structured, it may be possible for a cryptanalyst to exploit these regularities. For example, if it is known that the message always starts out with certain predefined fields, then the cryptanalyst may have a number of known plaintext–ciphertext pairs to work with. If the message has repetitive elements with a period of repetition a multiple of *b* bits, then these elements can be identified by the analyst. This may help in the analysis or may provide an opportunity for substituting or rearranging blocks.

## Cipher Block Chaining (CBC)

Makes sure encrypting the same plaintext block with the same key does not result in the same ciphertext block (in contrast to ECB)




IV must be known to sender and receiver, but kept hidden from third parties (e.g., using ECB encryption)

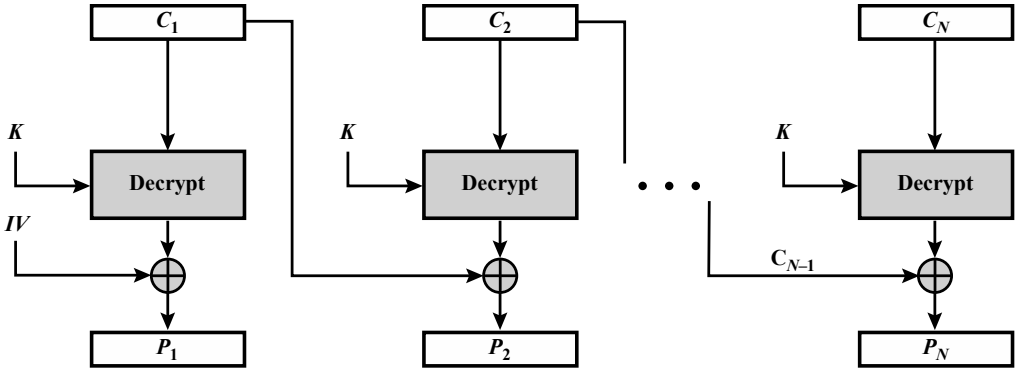


To overcome the security deficiencies of ECB, we would like a technique in which the same plaintext block, if repeated, produces different ciphertext blocks. A simple way to satisfy this requirement is the **cipher block chaining (CBC)** mode. In this scheme, the input to the encryption algorithm is the XOR of the current plaintext block and the preceding ciphertext block; the same key is used for each block. In effect, we have chained together the processing of the sequence of plaintext blocks. The input to the encryption function for each plaintext block bears no fixed relationship to the plaintext block. Therefore, repeating patterns of *b* bits are not exposed. As with the ECB mode, the CBC mode requires that the last block be padded to a full *b* bits if it is a partial block.

To produce the first block of ciphertext, an initialization vector (IV) is XORed with the first block of plaintext. The IV must be known to both the sender and receiver but be unpredictable by a third party. In particular, for any given plaintext, it must not be possible to predict the IV that will be associated to the plaintext in advance of the generation of the IV. For maximum security, the IV should be protected against unauthorized changes. This could be done by sending the IV using ECB encryption.

So long as it is unpredictable, the specific choice of IV is unimportant. SP 800-38A recommends two possible methods: The first method is to apply the encryption function, under the same key that is used for the encryption of the plaintext, to a **nonce**. The nonce must be a data block that is unique to each execution of the encryption operation. For example, the nonce may be a counter, a timestamp, or a message number. The second method is to generate a random data block using a random number generator.

## Cipher Block Chaining (CBC) decryption




University of Antwerp
Faculty of Science



For decryption, each cipher block is passed through the decryption algorithm. The result is XORed with the preceding ciphertext block to produce the plaintext block. On decryption, the IV is XORed with the output of the decryption algorithm to recover the first block of plaintext. The IV is a data block that is the same size as the cipher block.

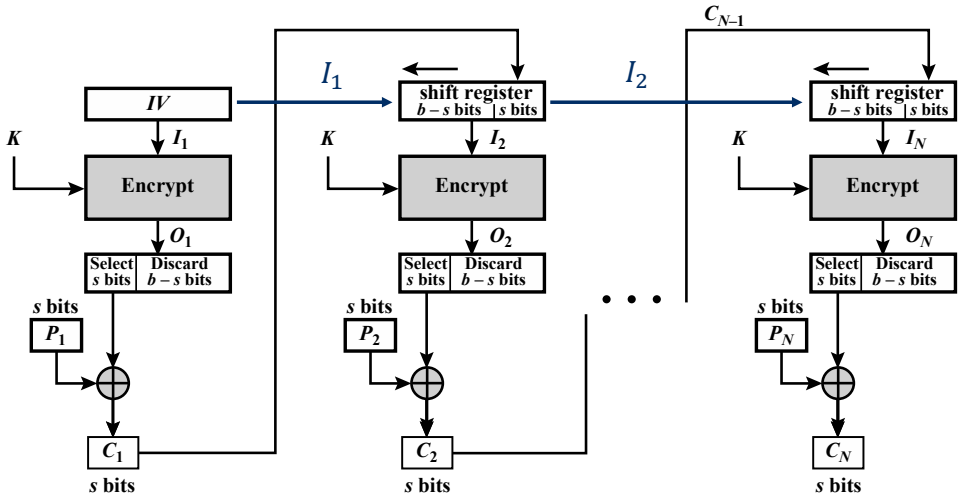
## Cipher Block Chaining (CBC) proof of equivalence

**Proof of equivalence** (i.e., decryption is the inverse of encryption)

$$P_j = D(K, C_j) \oplus C_{j-1}$$
$$\Leftrightarrow P_j \oplus C_{j-1} = D(K, C_j) \oplus C_{j-1} \oplus C_{j-1}$$
$$\Leftrightarrow E(K, P_j \oplus C_{j-1}) = E(K, D(K, C_j) \oplus C_{j-1} \oplus C_{j-1})$$
$$\Leftrightarrow E(K, P_j \oplus C_{j-1}) = E\left(K, D(K, C_j)\right)$$
$$\Leftrightarrow E(K, P_j \oplus C_{j-1}) = C_j$$

## Cipher Feedback (CFB)

- A stream cipher mode that operates on small blocks of s bits (e.g., 8)
- Starting from an IV of b bits (the original block size)




$$I_j = LSB_{b-s}(I_{j-1}) \,||\, C_{j-1}$$

$$C_j = MSB_s\left(E(K, I_j)\right) \oplus P_j$$

$LSB_x$ = x least significant bits;
$MSB_x$ = x most significant bits



For AES, DES, or any block cipher, encryption is performed on a block of *b* bits. In the case of DES, *b* = 64 and in the case of AES, *b* = 128. However, it is possible to convert a block cipher into a stream cipher, using one of the three modes to be discussed in this and the next two sections: **cipher feedback (CFB)** mode, **output feedback (OFB)** mode, and **counter (CTR)** mode. A stream cipher eliminates the need to pad a message to be an integral number of blocks. It also can operate in real time. Thus, if a character stream is being transmitted, each character can be encrypted and transmitted immediately using a character-oriented stream cipher. One desirable property of a stream cipher is that the ciphertext be of the same length as the plaintext. Thus, if 8-bit characters are being transmitted, each character should be encrypted to produce a ciphertext output of 8 bits. If more than 8 bits are produced, transmission capacity is wasted.

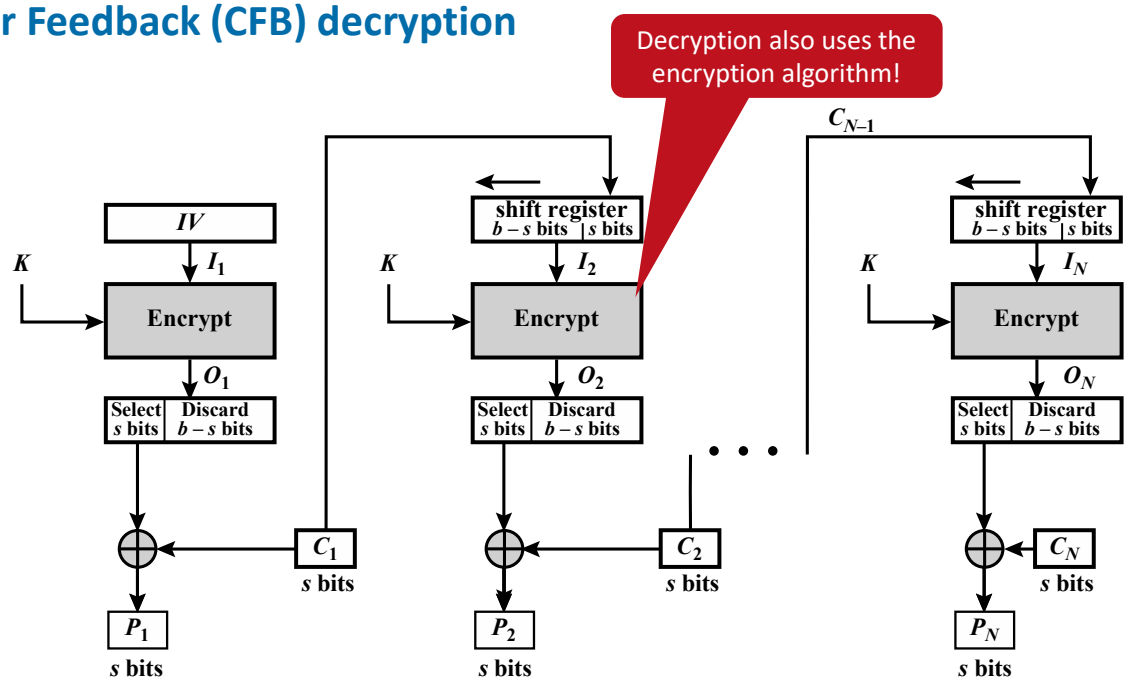
In CFB, it is assumed that the unit of transmission is *s* bits; a common value is *s* = 8. As with CBC, the units of plaintext are chained together, so that the ciphertext of any plaintext unit is a function of all the preceding plaintext. In this case, rather than blocks of *b* bits, the plaintext is divided into *segments* of *s* bits. The input to the encryption function is a *b*-bit shift register that is initially set to some initialization vector (IV). The leftmost (most significant) *s* bits of the output of the encryption function are XORed with the first segment of plaintext *P*1 to produce the first unit of ciphertext *C*1, which is then transmitted. In addition, the contents of the shift register are shifted left by *s* bits, and *C*1 is placed in the rightmost (least significant) *s* bits of the shift register. This process continues until all plaintext units have been encrypted.

Although CFB can be viewed as a stream cipher, it does not conform to the typical

construction of a stream cipher. In a typical stream cipher, the cipher takes as input some initial value and a key and generates a stream of bits, which is then XORed with the plaintext bits. In the case of CFB, the stream of bits that is XORed with the plaintext also depends on the plaintext.

In CFB encryption, like CBC encryption, the input block to each forward cipher function (except the first) depends on the result of the previous forward cipher function; therefore, multiple forward cipher operations cannot be performed in parallel.

## Cipher Feedback (CFB) decryption






For decryption, the same scheme is used, except that the received ciphertext unit is XORed with the output of the encryption function to produce the plaintext unit. In CFB decryption, the required forward cipher operations can be performed in parallel if the input blocks are first constructed (in series) from the IV and the ciphertext.

Note that it is the *encryption* function that is used, not the decryption function. This is easily explained. Let MSB$s$($X$) be defined as the most significant $s$ bits of $X$. Then

$$C1 = P1 \oplus \text{MSB}s[\text{E}(K, \text{IV})]$$

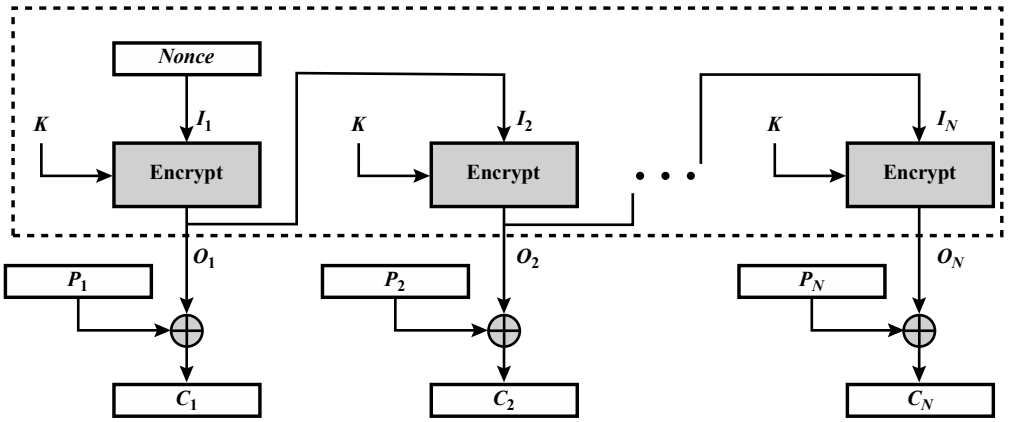
Therefore, by rearranging terms:

$$P1 = C1 \oplus \text{MSB}s[\text{E}(K, \text{IV})]$$

The same reasoning holds for subsequent steps in the process.

## Output feedback (OFB)

Similar to CFB, but instead operates on full blocks of size b




Nonce is an IV that must be unique for each execution, since the encryption of a block does not depend on the plaintext P, but only on the key K and the nonce itself.

University of Antwerp | Faculty of Science

136

The **output feedback (OFB)** mode is similar in structure to that of CFB. For OFB, the output of the encryption function is fed back to become the input for encrypting the next block of plaintext. In CFB, the output of the XOR unit is fed back to become input for encrypting the next block. The other difference is that the OFB mode operates on full blocks of plaintext and ciphertext, whereas CFB operates on an *s*-bit subset.

As with CBC and CFB, the OFB mode requires an initialization vector. In the case of OFB, the IV must be a nonce; that is, the IV must be unique to each execution of the encryption operation. The reason for this is that the sequence of encryption output blocks, $O_i$, depends only on the key and the IV and does not depend on the plaintext. Therefore, for a given key and IV, the stream of output bits used to XOR with the stream of plaintext bits is fixed. If two different messages had an identical block of plaintext in the identical position, then an attacker would be able to determine that portion of the $O_i$ stream.

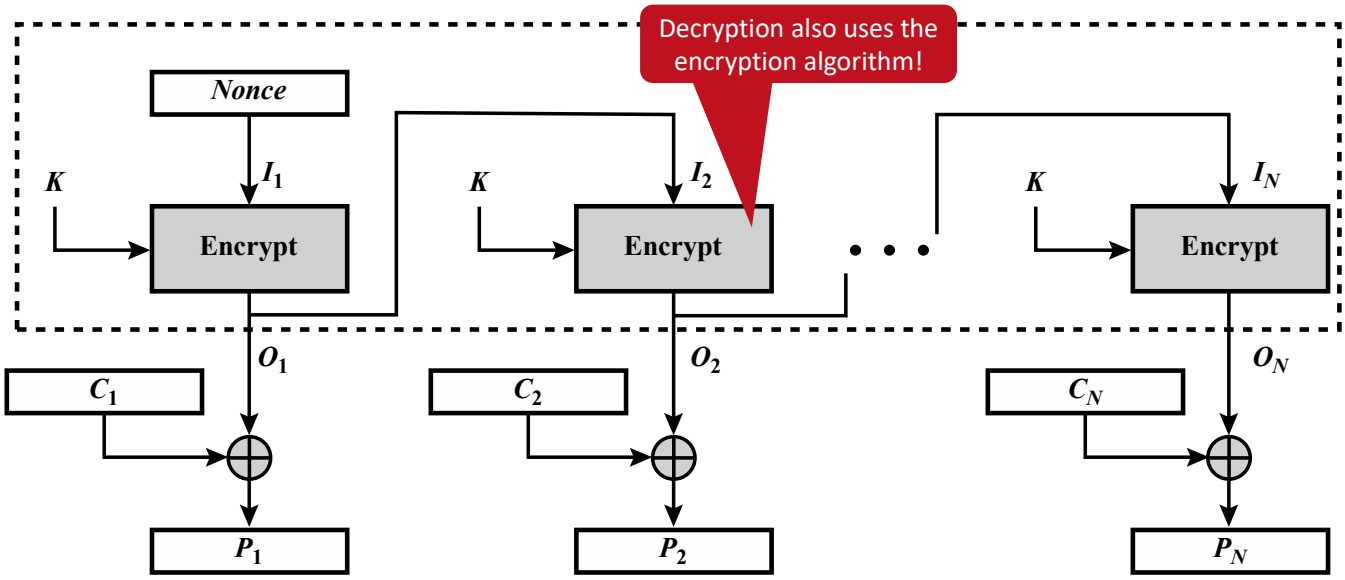
One advantage of the OFB method is that bit errors in transmission do not propagate. For example, if a bit error occurs in *C*1, only the recovered value of *P*1 is affected; subsequent plaintext units are not corrupted. With CFB, *C*1 also serves as input to the shift register and therefore causes additional corruption downstream.

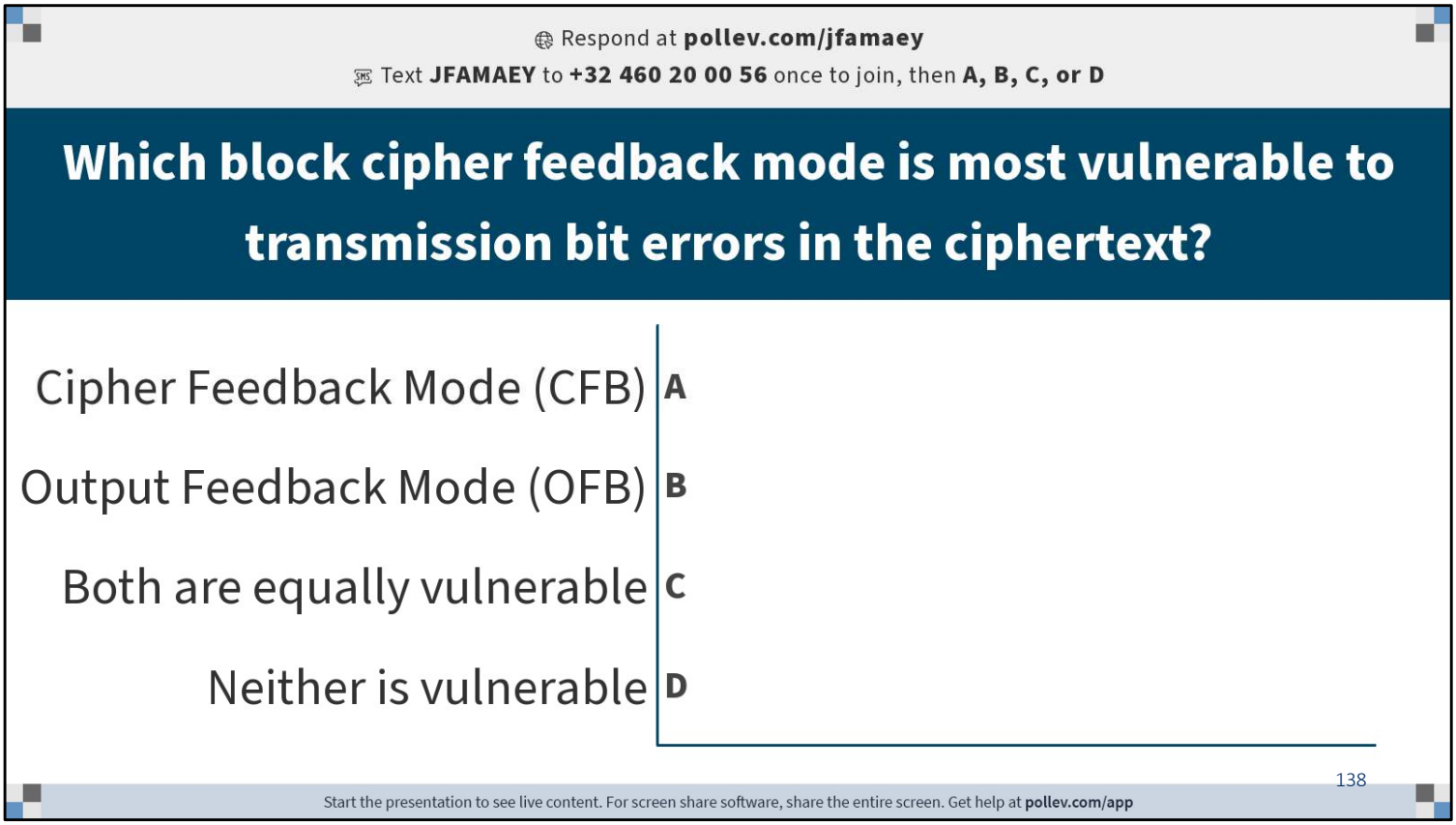
The disadvantage of OFB is that it is more vulnerable to a message stream modification attack than is CFB. Consider that complementing a bit in the cipher-text complements the corresponding bit in the recovered plaintext. Thus, controlled changes to the recovered plaintext can be made. This may make it possible for an opponent, by making

the necessary changes to the checksum portion of the message as well as to the data portion, to alter the ciphertext in such a way that it is not detected by an error-correcting code.

OFB has the structure of a typical stream cipher, because the cipher generates a stream of bits as a function of an initial value and a key, and that stream of bits is XORed with the plaintext bits. The generated stream that is XORed with the plaintext is itself independent of the plaintext. One distinction from the stream ciphers is that OFB encrypts plaintext a full block at a time, where typically a block is 64 or 128 bits. Many stream ciphers encrypt one byte at a time.

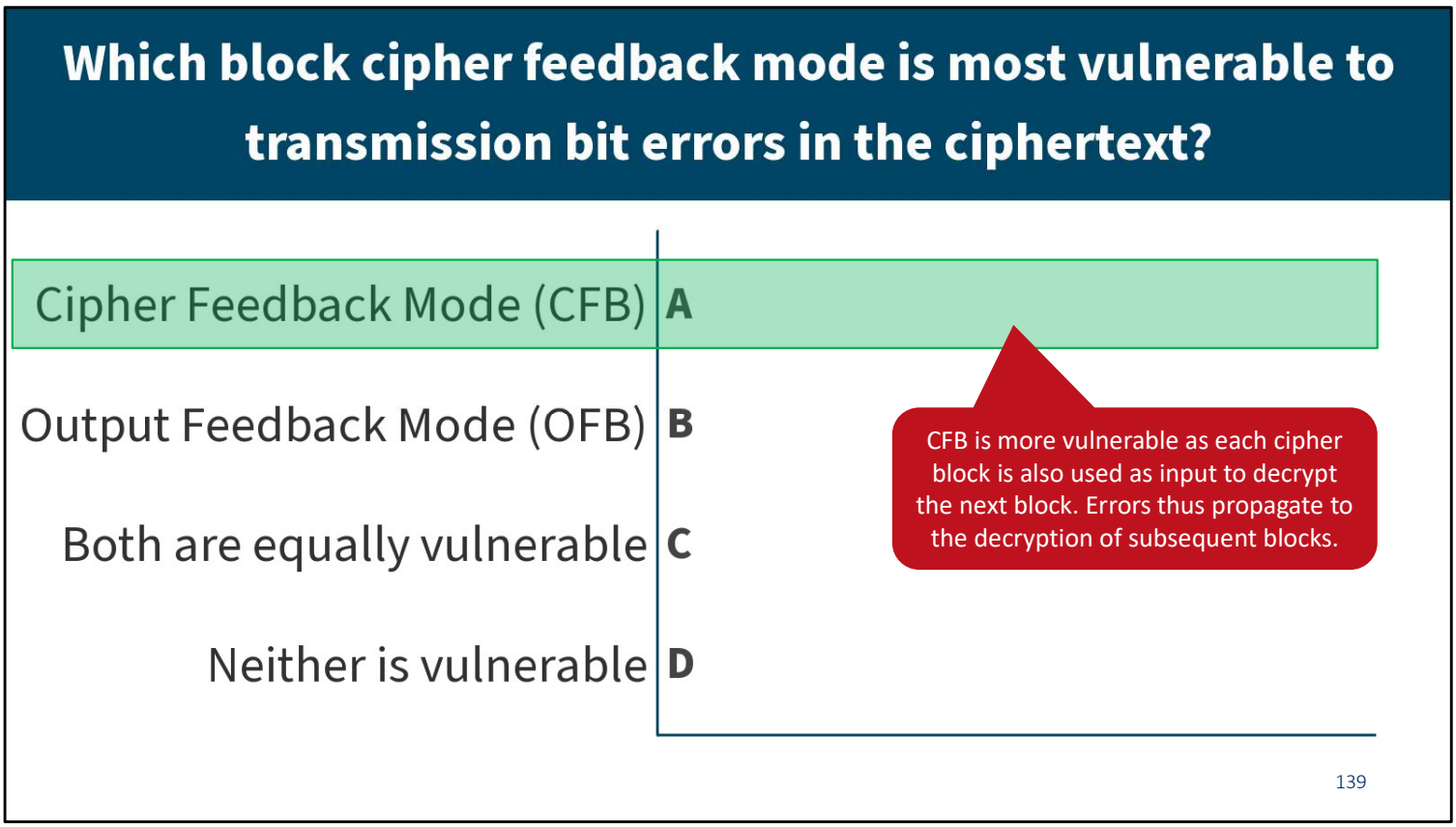
## Output feedback (OFB) decryption






Poll Title: Do not modify the notes in this section to avoid tampering with the Poll Everywhere activity.
More info at polleverywhere.com/support

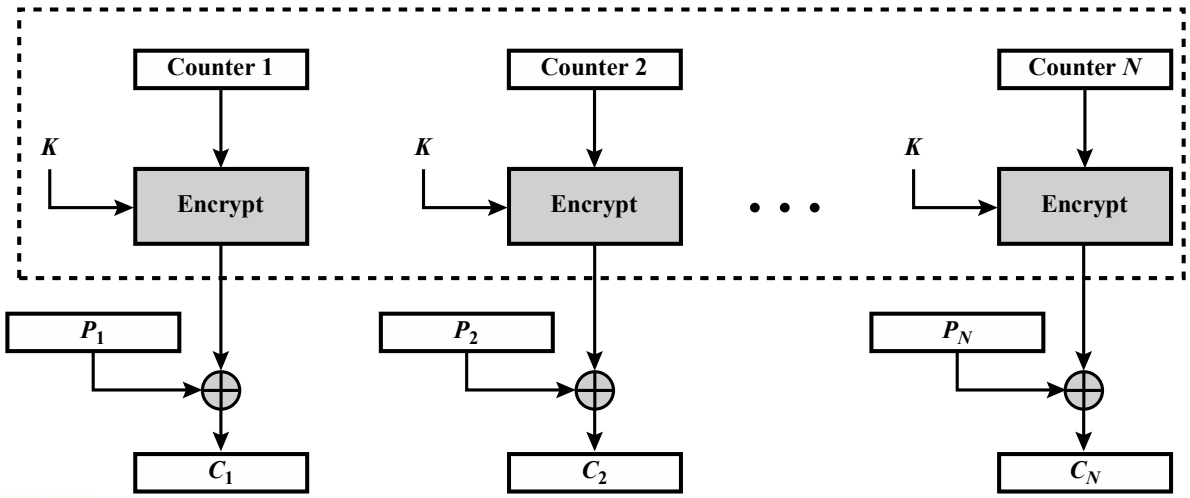
Which block cipher feedback mode is most vulnerable to transmission bit errors in the ciphertext?
https://www.polleverywhere.com/multiple_choice_polls/0xcaH2QO0CZxDTU?state=opened&flow=Default&onscreen=persist




Poll Title: Which block cipher feedback mode is most vulnerable to transmission bit errors in the ciphertext?

## Counter (CTR)

Uses a counter equal to the plaintext block size, which must be different for each encrypted plaintext block.




University of Antwerp | Faculty of Science

140

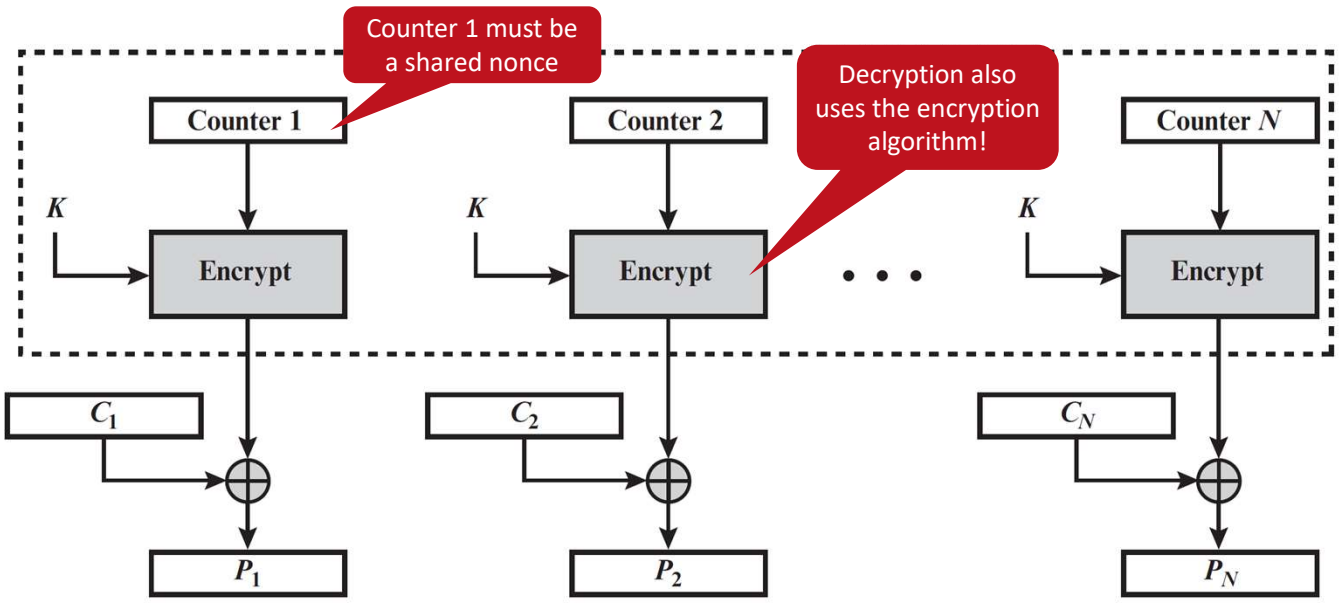
Although interest in the **counter (CTR)** mode has increased recently with applications to ATM (asynchronous transfer mode) network security and IPsec (IP security), this mode was proposed in 1979.

A counter equal to the plaintext block size is used. The only requirement stated in SP 800-38A is that the counter value must be different for each plaintext block that is encrypted. Typically, the counter is initialized to some value and then incremented by 1 for each subsequent block (modulo $2^b$, where $b$ is the block size). For encryption, the counter is encrypted and then XORed with the plaintext block to produce the ciphertext block; there is no chaining. For decryption, the same sequence of counter values is used, with each encrypted counter XORed with a ciphertext block to recover the corresponding plaintext block. Thus, the initial counter value must be made available for decryption.

As with the OFB mode, the initial counter value must be a nonce; that is, $T_1$ must be different for all of the messages encrypted using the same key. Further, all *Ti* values across all messages must be unique. If, contrary to this requirement, a counter value is used multiple times, then the confidentiality of all of the plaintext blocks corresponding to that counter value may be compromised. In particular, if any plaintext block that is encrypted using a given counter value is known, then the output of the encryption function can be determined easily from the associated ciphertext block. This output allows any other plaintext blocks that are encrypted using the same counter value to be easily recovered from their associated ciphertext blocks. One way to ensure the uniqueness of counter values is to continue to increment the counter value by 1 across

messages. That is, the first counter value of each message is one more than the last counter value of the preceding message.

## Counter (CTR) decryption



## Advantages of Counter mode

**Hard- and software efficiency**
As there is no chaining, encryption of blocks can be done in parallel

**Pre-processing**
Encryption only depends on the counter, and can therefore be done in advance

**Random access**
As there is no chaining, the $i^{th}$ block can be accessed in random access fashion

**Proven security**
If a unique counter is used for each block, CTR is at least as secure as other modes

**Simplicity**
CTR (like CFB and OFB) only requires the implementation of the encryption algorithm



- **Hardware efficiency:** Unlike the three chaining modes, encryption (or decryption) in CTR mode can be done in parallel on multiple blocks of plaintext or ciphertext. For the chaining modes, the algorithm must complete the computation on one block before beginning on the next block. This limits the maximum throughput of the algorithm to the reciprocal of the time for one execution of block encryption or decryption. In CTR mode, the throughput is only limited by the amount of parallelism that is achieved.
- **Software efficiency:** Similarly, because of the opportunities for parallel execution in CTR mode, processors that support parallel features, such as aggressive pipelining, multiple instruction dispatch per clock cycle, a large number of registers, and SIMD instructions, can be effectively utilized.
- **Preprocessing:** The execution of the underlying encryption algorithm does not depend on input of the plaintext or ciphertext. Therefore, if sufficient memory is available and security is maintained, preprocessing can be used to prepare the output of the encryption boxes that feed into the XOR functions. When the plaintext or ciphertext input is presented, then the only computation is a series of XORs. Such a strategy greatly enhances throughput.
- **Random access:** The *i*th block of plaintext or ciphertext can be processed in random-access fashion. With the chaining modes, block *Ci* cannot be computed until the *i* - 1 prior blocks are computed. There may be applications in which a ciphertext is stored and it is desired to decrypt just one block; for such applications, the random access feature is attractive.
- **Provable security:** It can be shown that CTR is at least as secure as the other modes.
- **Simplicity:** Unlike ECB and CBC modes, CTR mode requires only the implementation

of the encryption algorithm and not the decryption algorithm. This matters most when the decryption algorithm differs substantially from the encryption algorithm, as it does for AES. In addition, the decryption key scheduling need not be implemented.

## XEX-based Tweaked-codebook mode with ciphertext Stealing

- For encryption of data stored on sector-based devices
- First standardized by IEEE in 2007 (Std. 1619-2007) and later by NIST in 2010
- Based on the concept of **tweakable block ciphers**, that uses as input a (non-secret) tweak T in addition to the secret symmetric key K
- Requirements for "data at rest" differ from those of transmitted data
  1. The ciphertext is freely available for an attacker
  2. The encrypted data must be the same size as the plaintext data
  3. Data are accessed in fixed sized blocks, independently from each other
  4. Encryption is performed in 16-byte blocks, independent from other blocks
  5. There are no other metadata used, except the location of the data blocks
  6. The same plaintext is encrypted to different ciphertexts at different locations, but the same ciphertext in the same location
  7. A standard conformant device can be constructed for decryption of data encrypted by another standard conformant device



The XTS-AES mode is based on the concept of a **tweakable block cipher**, which functions in much the same manner as a salt used with passwords.

The requirements for encrypting stored data, also referred to as "data at rest" differ somewhat from those for transmitted data. The P1619 standard was designed to have the following characteristics:

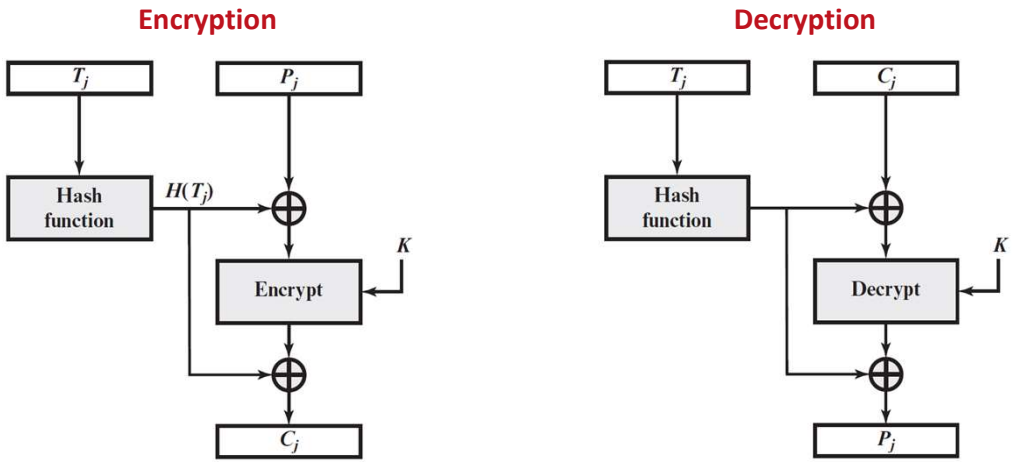
1. The ciphertext is freely available for an attacker. Among the circumstances that lead to this situation:
    1. A group of users has authorized access to a database. Some of the records in the database are encrypted so that only specific users can successfully read/write them. Other users can retrieve an encrypted record but are unable to read it without the key.
    2. An unauthorized user manages to gain access to encrypted records.
    3. A data disk or laptop is stolen, giving the adversary access to the encrypted data.
2. The data layout is not changed on the storage medium and in transit. The encrypted data must be the same size as the plaintext data.
3. Data are accessed in fixed sized blocks, independently from each other. That is, an authorized user may access one or more blocks in any order.
4. Encryption is performed in 16-byte blocks, independently from other blocks (except the last two plaintext blocks of a sector, if its size is not a multiple of 16 bytes).
5. There are no other metadata used, except the location of the data blocks within the whole data set.
6. The same plaintext is encrypted to different ciphertexts at different locations, but

always to the same ciphertext when written to the same location again.
7. A standard conformant device can be constructed for decryption of data encrypted by another standard conformant device.

The P1619 group considered some of the existing modes of operation for use with stored data. For CTR mode, an adversary with write access to the encrypted media can flip any bit of the plaintext simply by flipping the corresponding ciphertext bit.

Next, consider requirement 6 and the use of CBC. To enforce the requirement that the same plaintext encrypts to different ciphertext in different locations, the IV could be derived from the sector number. Each sector contains multiple blocks. An adversary with read/write access to the encrypted disk can copy a ciphertext sector from one position to another, and an application reading the sector off the new location will still get the same plaintext sector (except perhaps the first 128 bits). For example, this means that an adversary that is allowed to read a sector from the second position but not the first can find the content of the sector in the first position by manipulating the ciphertext. Another weakness is that an adversary can flip any bit of the plaintext by flipping the corresponding ciphertext bit of the previous block, with the side-effect of “randomizing” the previous block.

## General structure of tweakable block ciphers




A block cipher mode of operation can be created by using a different tweak value on each block.



A tweakable block cipher is one that has three inputs: a plaintext $P$, a symmetric key $K$, and a tweak $T$; and produces a ciphertext output $C$. We can write this as $C = E(K, T, P)$. The tweak need not be kept secret. Whereas the purpose of the key is to provide security, the purpose of the tweak is to provide variability. That is, the use of different tweaks with the same plaintext and same key produces different outputs. The basic structure of several tweakable clock ciphers that have been implemented is shown in the figure. Encryption can be expressed as:

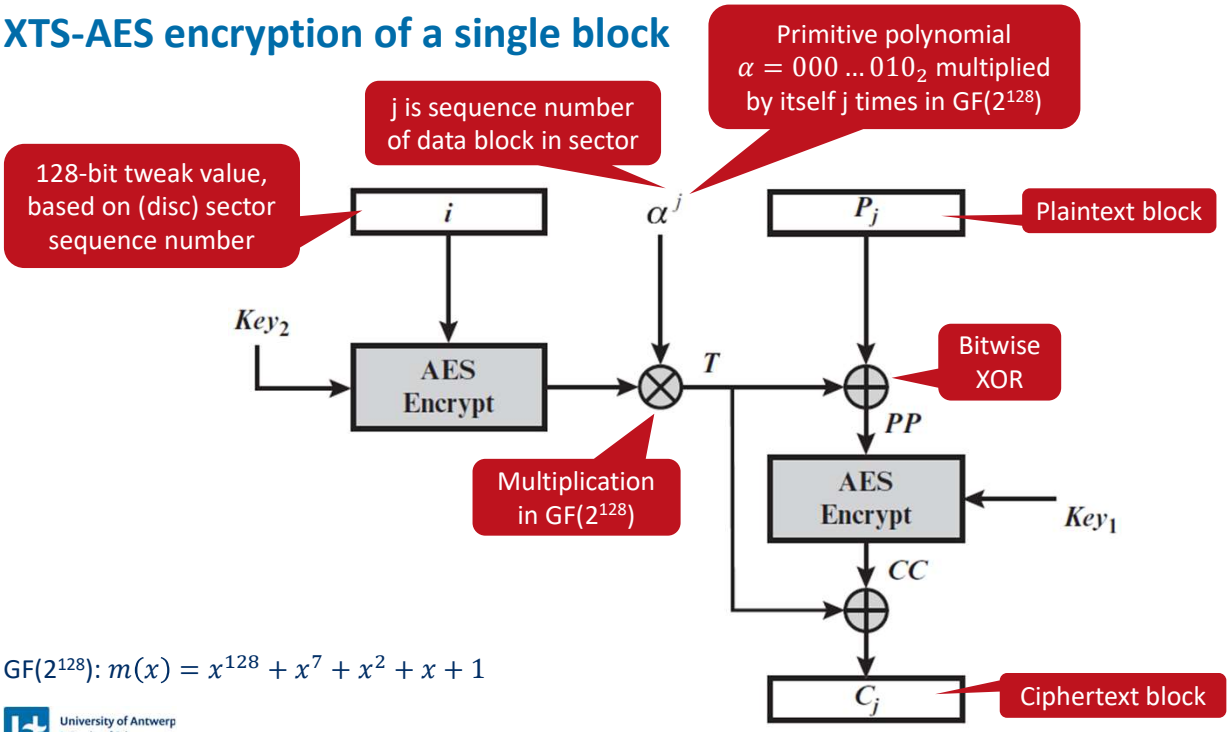
$$C = H(T) \oplus E(K, H(T) \oplus P)$$

where H is a hash function. For decryption, the same structure is used with the plaintext as input and decryption as the function instead of encryption. To see that this works, we can write

$$H(T) \oplus C = E(K, H(T) \oplus P)$$
$$D[K, H(T) \oplus C] = H(T) \oplus P$$
$$H(T) \oplus D(K, H(T) \oplus C) = P$$

It is now easy to construct a block cipher mode of operation by using a different tweak value on each block. In essence, the ECB mode is used but for each block the tweak is changed. This overcomes the principal security weakness of ECB, which is that two encryptions of the same block yield the same ciphertext.

## XTS-AES encryption of a single block




GF($2^{128}$): $m(x) = x^{128} + x^7 + x^2 + x + 1$

University of Antwerp | Faculty of Science

145

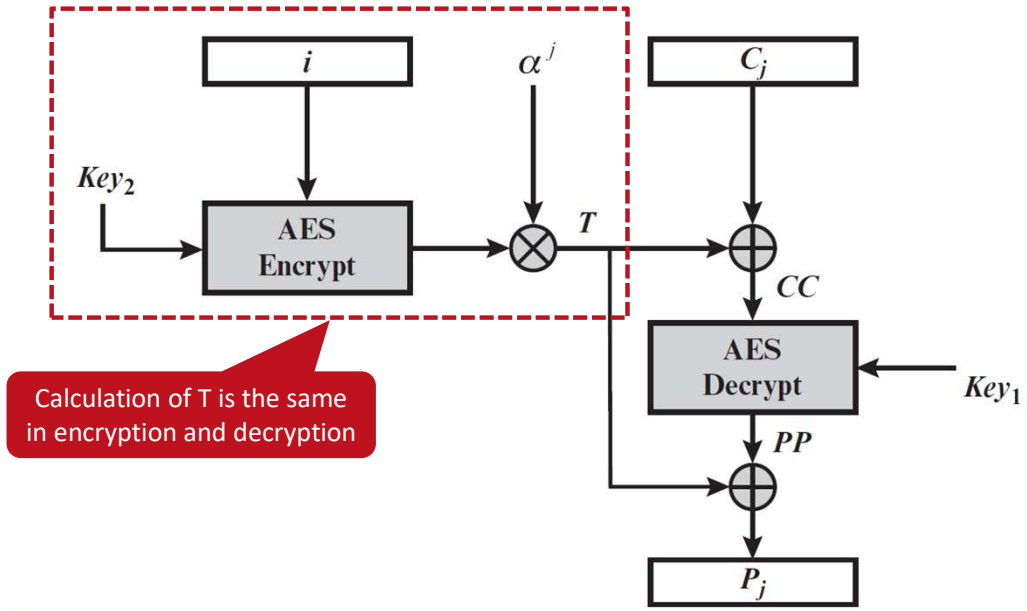
The operation involves two instances of the AES algorithm with two keys. The following parameters are associated with the algorithm.

- ***Key*** The 256 or 512 bit XTS-AES key; this is parsed as a concatenation of two fields of equal size called $Key_1$ and $Key_2$, such that $Key = Key_1 \,||\, Key_2$.
- $\boldsymbol{P_j}$ The *j*th block of plaintext. All blocks except possibly the final block have a length of 128 bits. A plaintext data unit, typically a disk sector, consists of a sequence of plaintext blocks $P_1, P_2, \ldots, P_m$.
- $\boldsymbol{C_j}$ The *j*th block of ciphertext. All blocks except possibly the final block have a length of 128 bits.
- ***j*** The sequential number of the 128-bit block inside the data unit.
- ***i*** The value of the 128-bit tweak. Each data unit (sector) is assigned a tweak value that is a nonnegative integer. The tweak values are assigned consecutively, starting from an arbitrary nonnegative integer.
- **α** A primitive element of GF($2^{128}$) that corresponds to polynomial *x* (i.e., 0000 ... $010_2$).
- $\boldsymbol{\alpha}_j$ a multiplied by itself *j* times, in GF($2^{128}$).
- $\oplus$ Bitwise XOR.
- $\otimes$ Modular multiplication of two polynomials with binary coefficients modulo $x^{128} + x^7 + x^2 + x + 1$. Thus, this is multiplication in GF($2^{128}$).

In essence, the parameter *j* functions much like the counter in CTR mode. It assures that if the same plaintext block appears at two different positions within a data unit, it will encrypt to two different ciphertext blocks. The parameter *i* functions much like a nonce at the data unit level. It assures that, if the same plaintext block appears at the same

position in two different data units, it will encrypt to two different ciphertext blocks. More generally, it assures that the same plaintext data unit will encrypt to two different ciphertext data units for two different data unit positions.

## XTS-AES decryption of a single block




University of Antwerp
Faculty of Science



To see that decryption recovers the plaintext, let us expand the last line of both encryption and decryption. For encryption, we have

$$C = CC \oplus T = \mathrm{E}(K1, \mathrm{PP}) \oplus T = \mathrm{E}(K1, P \oplus T) \oplus T$$

and for decryption, we have

$$P = PP \oplus T = \mathrm{D}(K1, CC) \oplus T = \mathrm{D}(K1, C \oplus T) \oplus T$$

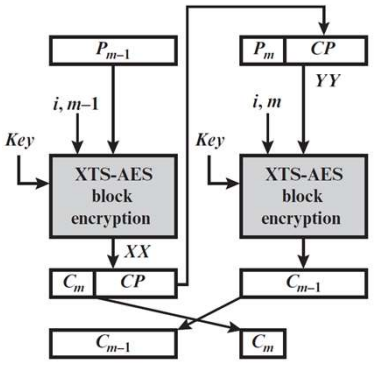
Now, we substitute for C:

$$\begin{aligned} P &= \mathrm{D}(K1, C \oplus T) \oplus T \\ &= \mathrm{D}(K1, [E(K1, P \oplus T) \oplus T] \oplus T) \oplus T \\ &= \mathrm{D}(K1, E(K1, P \oplus T)) \oplus T \\ &= (P \oplus T) \oplus T = P \end{aligned}$$

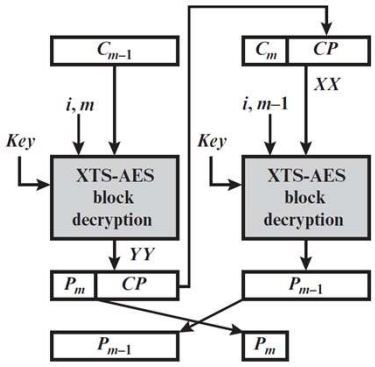
## XTS-AES operations on a full sector

- Each sector consists of a set of 128-bit blocks labelled $P_0$, $P_1$, ..., $P_m$
- All blocks, except the last 2, are encrypted independently (cf., previous slides)
- If the last block contains less than 128 bits, **ciphertext-stealing** is applied

**Encryption of $P_{m-1}$ and $P_m$**




**Decryption of $C_{m-1}$ and $C_m$**




University of Antwerp | Faculty of Science

147

The plaintext of a sector or data unit is organized into blocks of 128 bits. Blocks are labelled $P_0$, $P_1$, ..., $P_m$. The last block may be null or may contain from 1 to 127 bits. In other words, the input to the XTS-AES algorithm consists of $m$ 128-bit blocks and possibly a final partial block.

For encryption and decryption, each block is treated independently and encrypted/decrypted as shown in the previous slides. The only exception occurs when the last block has less than 128 bits. In that case, the last two blocks are encrypted/decrypted using a **ciphertext-stealing** technique instead of padding. The figure shows the scheme. $P_{m-1}$ is the last full plaintext block, and $P_m$ is the final plaintext block, which contains $s$ bits with $1 \leq s \leq 127$. $C_{m-1}$ is the last full ciphertext block, and $C_m$ is the final ciphertext block, which contains $s$ bits. This technique is commonly called ciphertext stealing because the processing of the last block "steals" a temporary ciphertext of the penultimate block to complete the cipher block.

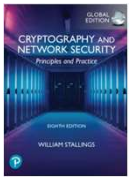
As can be seen, XTS-AES mode, like CTR mode, is suitable for parallel operation. Because there is no chaining, multiple blocks can be encrypted or decrypted simultaneously. Unlike CTR mode, XTS-AES mode includes a nonce (the parameter $i$) as well as a counter (parameter $j$).

# Summary of block cipher operational modes

- Block cipher operation modes:
  - Ensure that the same plaintext block does not result in same ciphertext block
  - CFB mode allows block ciphers to be used as stream ciphers
  - CTR mode is computationally efficient and easy to implement
  - XTS-AES mode is optimized for the peculiarities of encrypting data-at-rest

| Mode | Typical Application |
|---|---|
| **Electronic Codebook (ECB)** | • Secure transmission of one values (e.g., encryption key) |
| **Cipher Block Chaining (CBC)** | • General-purpose block-oriented transmission<br>• Authentication |
| **Cipher Feedback (CFB)** | • General-purpose stream-oriented transmission<br>• Authentication |
| **Output Feedback (OFB)** | • Stream-oriented transmission over noisy channel (e.g., satellite communication) |
| **Counter (CTR)** | • General-purpose block-oriented transmission<br>• Useful for high-speed requirements |
| **XEX-based Tweaked-codebook mode with ciphertext Stealing (XTS-AES)** | • Developed specifically for sector-based storage devices<br>• Protects against adversaries that have access to the stored data |

# Further reading on block cipher operational modes




- Chapter 7 (Section 7.2—7.7)

- How to Use Block Ciphers 1
  - Modes of operation: one-time key
- How to Use Block Ciphers 2
  - CBC
  - CTR

# End of Part 2

University of Antwerp
Faculty of Science