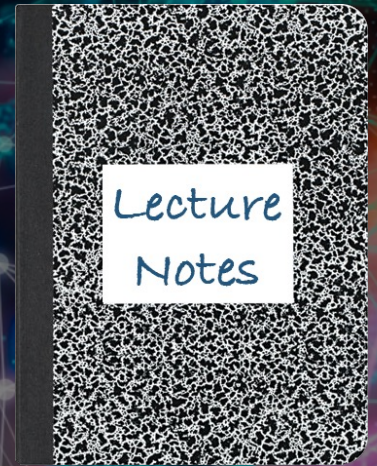


CS 419: Computer Security

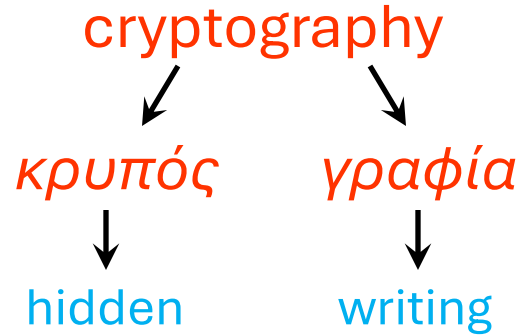
# Week 2: Cryptography

## Symmetric Cryptography

Paul Krzyzanowski

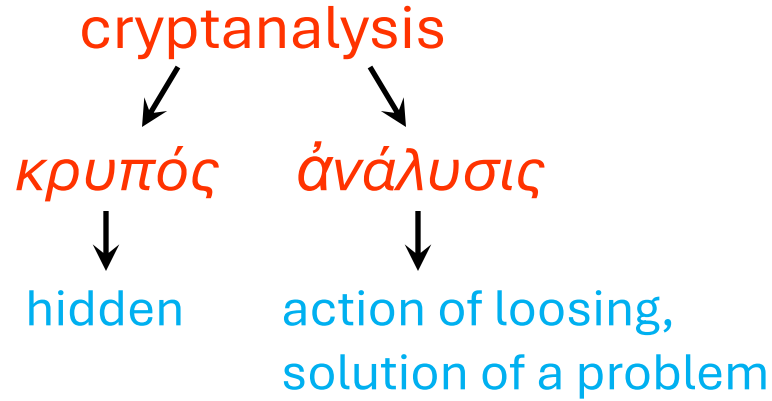


© 2025 Paul Krzyzanowski. No part of this content, may be reproduced or reposted in whole or in part in any manner without the permission of the copyright owner.



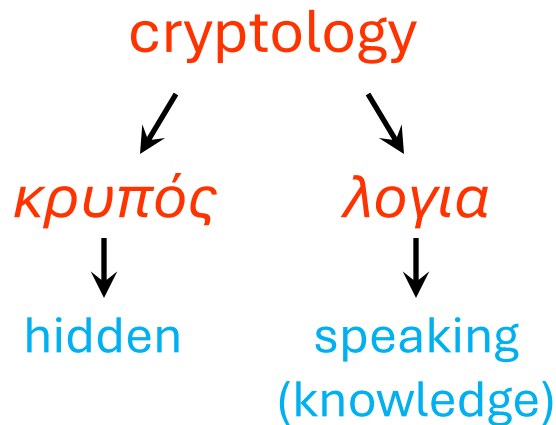
A secret manner of writing, ... Generally, the art of writing or solving ciphers.

— *Oxford English Dictionary*



The analysis and decryption of encrypted text or information without prior knowledge of the keys.

— *Oxford English Dictionary*



**1967** D. Kahn, *Codebreakers* p. xvi, Cryptology is the science that embraces cryptography and cryptanalysis, but the term ‘cryptology’ sometimes loosely designates the entire dual field of both rendering signals secure and extracting information from them.

— *Oxford English Dictionary*

# Cryptography $\neq$ Security

Cryptography may be a component of a secure system

Just adding cryptography may not make a system secure

# Cryptography: what is it good for?

- Confidentiality
  - Others cannot read contents of the message
- Authentication
  - Determine origin of message
- Integrity
  - Verify that message has not been modified
- Nonrepudiation
  - Sender should not be able to falsely deny that a message was sent

# Terms

**Plaintext** (cleartext) message  $P$

**Encryption**  $E(P)$

Produces **Ciphertext**,  $C = E(P)$

**Decryption**,  $P = D(C)$

**Cipher** = cryptographic algorithm

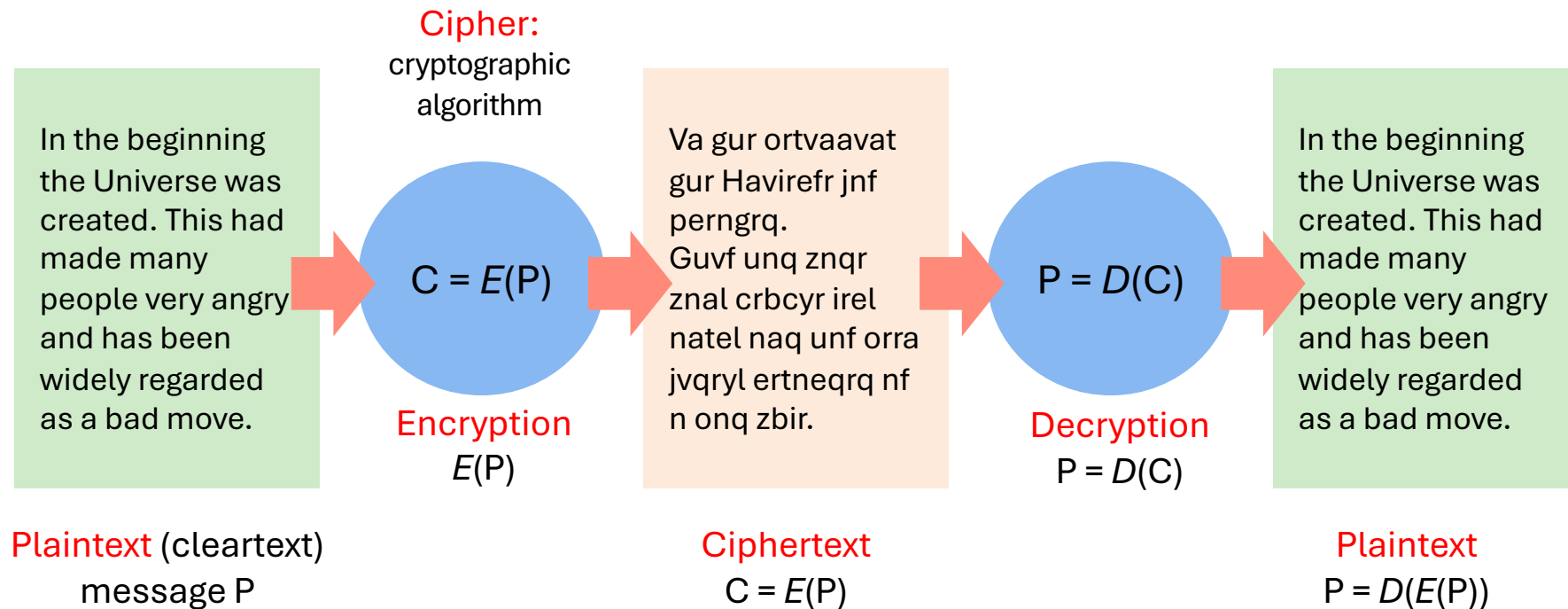


**Cryptosystem**

The diagram consists of a vertical line with two horizontal arrows pointing left. The top arrow points to the word 'Encryption' and the bottom arrow points to the word 'Decryption'. The vertical line is positioned to the right of the text 'Encryption' and 'Decryption', and to the left of the text 'Cryptosystem'.

Encryption & decryption algorithms  
for a specific cipher

# Terms



**Cryptosystem:** Encryption & decryption algorithms for a specific cipher



# Obfuscation and Secret Algorithms

- Obfuscation
  - The algorithm takes plaintext as input and creates ciphertext
  - All security rests in the algorithm
  - Vulnerable to leaking and reverse engineering
  - Useless once exposed
- Secret (proprietary) algorithms
  - No large-scale peer review to validate its effectiveness (who will test it?)

Many proprietary algorithms have been reverse-engineered and found to be weak

- A5/1, A5/2 – used in GSM encryption
- RC3, RC4 – used in SSL, WEP (Wired Equivalent Privacy) on Wi-Fi networks
- Crypto AG – a Swiss company that added backdoors under direction of the CIA and German BND
- DECT Standard Cipher (DSC) – used in cordless phones
- Content Scrambling System (CSS) – DVD encryption
- HDCP (High-Bandwidth Digital Content Protection) – HDTV interface
- Advanced Access Content System – Blu-ray encryption
- Firewire protocol
- Enigma cipher machine – German WWII cipher
- Every NATO and Warsaw Pact algorithm during Cold War

# Schneier's Law

*"Anyone, from the most clueless amateur to the best cryptographer, can create an algorithm that he himself can't break. It's not even hard."*

*— Bruce Schneier*

See [https://en.wikipedia.org/wiki/Category:Broken\\_cryptography\\_algorithms](https://en.wikipedia.org/wiki/Category:Broken_cryptography_algorithms)

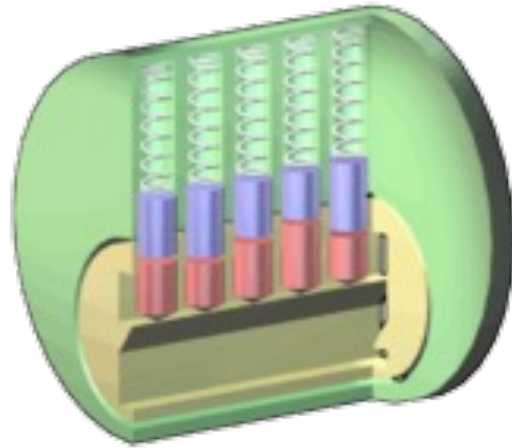
# Shared algorithms & secret keys

# The key



BTW, the above is a *bump key*. See [http://en.wikipedia.org/wiki/Lock\\_bumping](http://en.wikipedia.org/wiki/Lock_bumping)

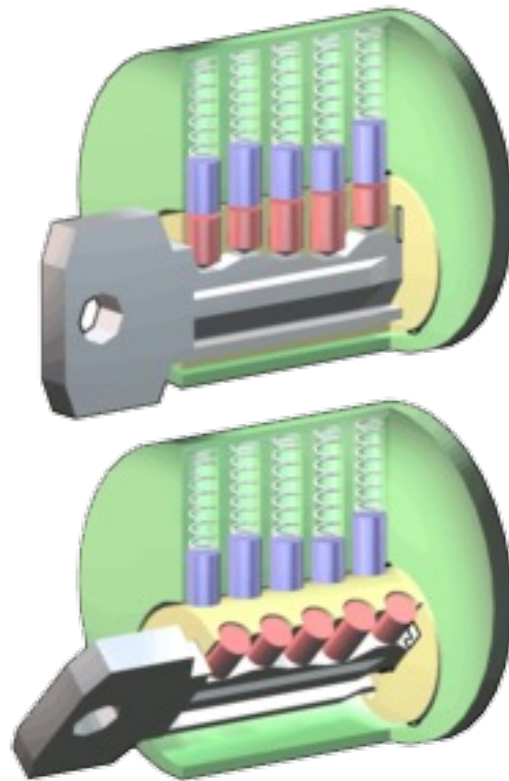
# The lock



Source: [en.wikipedia.org/wiki/Pin\\_tumbler\\_lock](https://en.wikipedia.org/wiki/Pin_tumbler_lock)

# The key & lock

- We understand how the mechanism works:
  - Strengths
  - Weaknesses
- Based on this understanding, we can assess how much to trust the key & lock



Source: [en.wikipedia.org/wiki/Pin\\_tumbler\\_lock](https://en.wikipedia.org/wiki/Pin_tumbler_lock)

# Kerckhoffs's Principle (1883)

A cryptosystem should be secure even if everything about the system, **except the key**, is public knowledge

Security should rest entirely on the secrecy of the key

# Symmetric Ciphers

One shared secret key,  $K$ , for encryption & decryption

$$C = E_K(P)$$

$$P = D_K(C)$$



# Classic Cryptosystems: Substitution Ciphers

# Atbash (אתבש) – Ancient Hebrew cipher

MY CAT HAS FLEAS

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
Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A

NBXZGSZHUOVZH

Origin of the name:

א – alef (1<sup>st</sup> letter),  
ת – tav (last letter),  
ב – bet (2<sup>nd</sup>),  
ש – shin (2<sup>nd</sup> to last)

c. 600 BCE

No information (key) needs to be conveyed!

א	ב	ג	ד	ה	ו	ז	ח	ט	י	כ	ל	מ	נ	ס	ע	פ	צ	ק	ר	ש	ת
ת	ש	ר	ק	צ	פ	ע	ס	נ	מ	ל	כ	י	ט	ח	ז	ו	ה	ד	ג	ב	א

# India – Mlecchita vikalpa: Kautiliya

*“The art of understanding writing in cypher, and the writing of words in a peculiar way”*

## Kautiliya

- Documented in the Kama Sutra
- Phonetic substitution scheme used in India 400 BCE – 200 CE
- Short & long vowels are exchanged with consonants

a	ā	i	ī	u	ū	ṛ	ṛī	ḷ	ḷī	e	ai	o	au	ṁ	ḥ	ñ	ś	ṣ	s	i	r	l	u
kh	g	gh	ṇ	ch	j	jh	ñ	ṭh	ḍ	ḍh	ṇ	th	d	dh	n	ph	b	bh	m	y	r	l	v

# Cæsar cipher

## Earliest documented military use of cryptography

- Julius Caesar c. 60 BCE
  - Documented by the Roman historian Suetonius in *De Vita Caesarum*
- **Shift cipher**: a simple form of a **substitution cipher**
- Each letter is replaced by one  $n$  positions away modulo alphabet size
  - $n$  = shift value = key
  - Caesar used  $n = 3$

# Cæsar cipher

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
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

# Cæsar cipher

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
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T

—————→ *shift alphabet by n (6)*

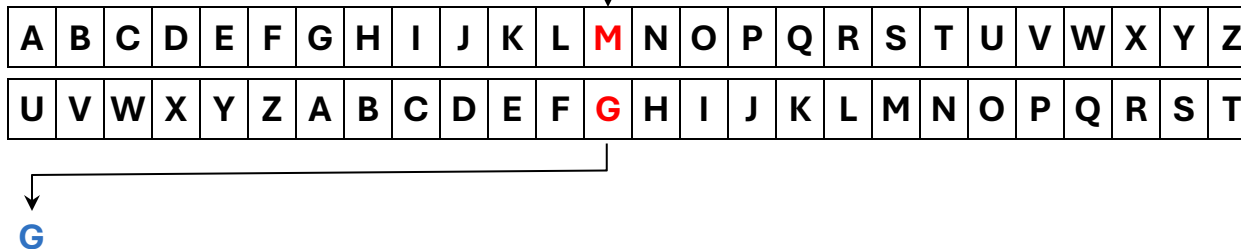
# Cæsar cipher

MY CAT HAS FLEAS

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
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T

# Cæsar cipher

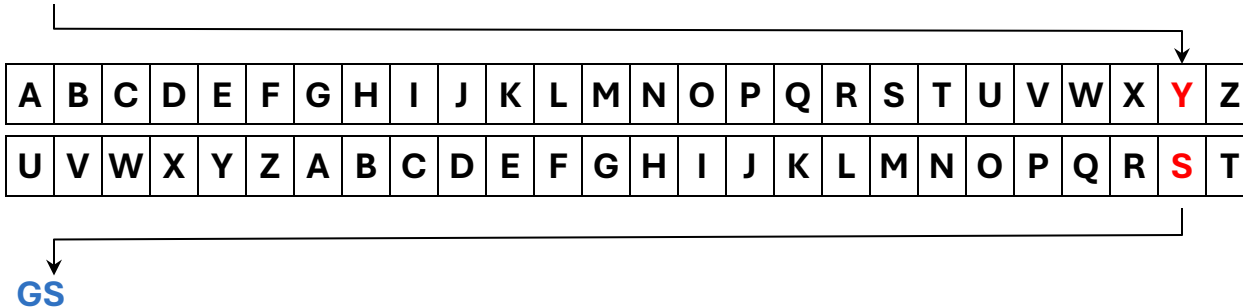
MY CAT HAS FLEAS





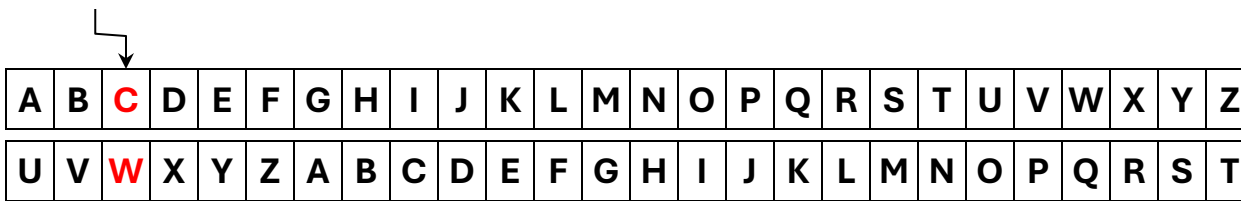
# Cæsar cipher

MY CAT HAS FLEAS



# Cæsar cipher

MY CAT HAS FLEAS



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
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T

GSW

# Cæsar cipher

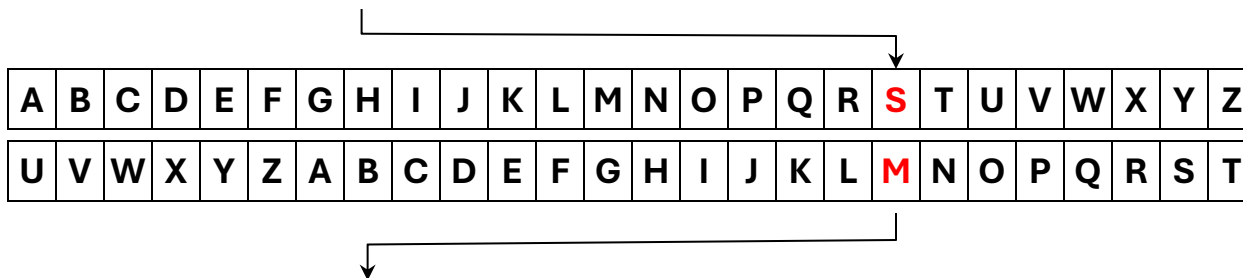
MY CAT HAS FLEAS

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
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T

GSWU

# Cæsar cipher

MY CAT HAS FLEAS



GSWUNBMUFZYUM

- One piece of information is needed for decryption: *key = shift value*
- Trivially easy to crack  
(25 possibilities for a 26-character alphabet)

# Monoalphabetic substitution cipher

MY CAT HAS FLEAS

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
M	P	S	R	L	Q	E	A	J	T	N	C	I	F	Z	W	O	Y	B	X	G	K	U	D	V	H

IVSMXAMBQCLMB

**Monoalphabetic** = constant mapping between plaintext and ciphertext

General case: arbitrary mapping

(versus a Cæsar cipher, where the letters are in an alphabetic sequence but shifted)

*Both sides must have the same substitution alphabet*

# Monoalphabetic substitution cipher: frequency analysis

Easy to decode: vulnerable to frequency analysis

Moby Dick (1.2M chars)		Shakespeare (55.8M chars)	
e	12.300%	e	11.797%
o	7.282%	o	8.299%
d	4.015%	d	3.943%
b	1.773%	b	1.634%
x	0.108%	x	0.140%

## Common digrams

TH (3.56%), HE (3.07%),  
IN (2.43%), ER (2.05%),  
AN, RE, ...

## Common trigrams

THE, ING, AND, HER, ERE, ...

# Shannon Entropy

Shannon Entropy is a measure of the uncertainty or randomness in a set of data

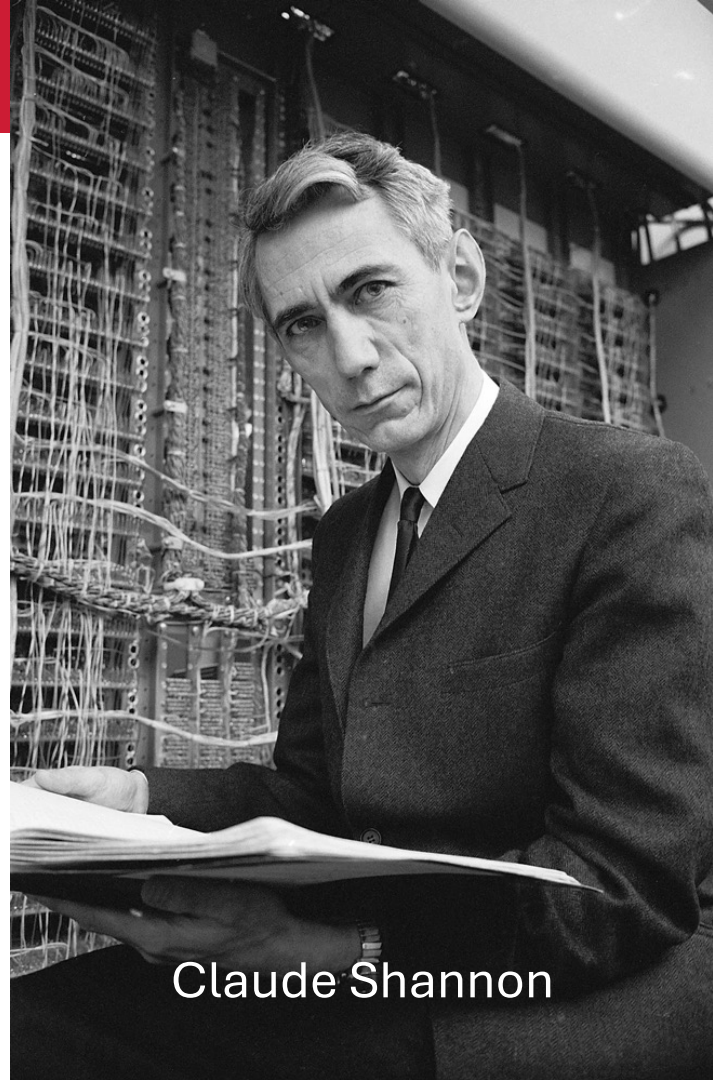
$$H(X) = -\sum P(x) \log_2 P(x)$$

where  $P(x)$  is the probability of occurrence of a particular outcome  $x$  (e.g., the frequency of a character appearing in the message)

It measures the amount of information in bits

**High entropy** = more randomness

⇒ which is what we want for ciphertext



Claude Shannon

# Entropy

If all letters were equally probable in a message, the entropy would be  $\log_2 26 = 4.70$  bits of information per byte

- Moby Dick text: entropy = 4.175
- Shakespeare texts: entropy = 4.19

Monoalphabetic substitution ciphers don't increase entropy

- The frequency of characters remains the same – they're just different characters

*Increase entropy by enabling the same character to be encrypted differently in different parts of the message*

If each letter's probability is the same:

$$H = - \sum_{i=1}^{26} \frac{1}{26} \log_2 \left( \frac{1}{26} \right)$$

$$H = -26 \times \frac{1}{26} \log_2 \left( \frac{1}{26} \right)$$

$$H = -\log_2 \left( \frac{1}{26} \right)$$

$$H = \log_2(26)$$

$$H = 4.7004$$



# Polyalphabetic substitution ciphers

- Designed to thwart frequency analysis techniques
  - Different ciphertext symbols can represent the same plaintext symbol
  - 1 → *many* relationship between letter and substitution
- Leon Battista Alberti: 1466
  - Two disks
  - Line up predetermined letter on inner disk with outer disk
  - Plaintext on inner → ciphertext on outer
  - After  $n$  symbols, the disk is rotated to a new alignment

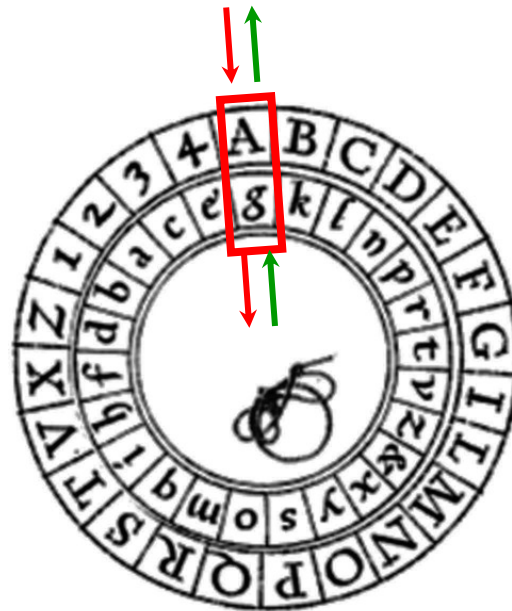


Image source: [https://en.wikipedia.org/wiki/Alberti\\_cipher\\_disk](https://en.wikipedia.org/wiki/Alberti_cipher_disk)

encrypt: A → g  
decrypt: g → A



<http://dabblesandbabbles.com/printable-secret-decoder-wheel/>

# Vigenère polyalphabetic cipher

Blaise de Vigenère, court of Henry III of France, 1518

No need for a disk: use table and key word to encipher a message

	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
A	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
B	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	A
C	C	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
D	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
E	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	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	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

# Vigenère polyalphabetic cipher

- Repeat keyword over text: (e.g., key=FACE)

Keystream:      **FA CEF ACE FACEF ...**

Plaintext:        **MY CAT HAS FLEAS**

- Encrypt:** find intersection:

**row** = keystream letter

**column** = plaintext (message) letter

- Decrypt:** find column

– Row = keystream letter, search for ciphertext

– Column heading = plaintext letter

The message is encrypted with as many substitution ciphers as there are unique letters in the keyword

	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
A	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
B	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	A
C	C	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
D	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
E	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	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	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

# Vigenère polyalphabetic cipher

*plaintext letter* ↓

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

↑ *ciphertext letter*

*keytext letter* →



# Vigenère polyalphabetic cipher

FA CEF ACE FACEF  
MY CAT HAS FLEAS

---

R

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
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	A
C	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
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
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	D
F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G

# Vigenère polyalphabetic cipher

FA CEF ACE FACEF  
MY CAT HAS FLEAS

---

R<sup>Y</sup>

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
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	A
C	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
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
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	D
F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G

# Vigenère polyalphabetic cipher

FA CEF ACE FACEF  
MY CAT HAS FLEAS

---

RY **E**

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
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	A
C	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
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
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	D
F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G



# Vigenère polyalphabetic cipher

FA CEF ACE FACEF

MY CAT HAS FLEAS

RY EE

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
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	A
C	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
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
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	D
F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G

# Vigenère polyalphabetic cipher

FA CEF ACE FACEF

MY CAT HAS FLEAS

RY EEY

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
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	A
C	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
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
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	D
F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G

# Vigenère polyalphabetic cipher

FA CEF ACE FACEF

MY CAT HAS FLEAS

RY EEY HCW KLGE~~X~~

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
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	A
C	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
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
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	D
F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G

# Vigenère polyalphabetic cipher

*"The rebels reposed their major trust, however, in the Vigenère, sometimes using it in the form of a brass cipher disc. In theory, it was an excellent choice, for so far as the South knew the cipher was unbreakable. In practice, it proved a dismal failure. For one thing, transmission errors that added or subtracted a letter ... unmeshed the key from the cipher and caused no end of difficulty. Once Major Cunningham of General Kirby-Smith's staff tried for twelve hours to decipher a garbled message; he finally gave up in disgust and galloped around the Union flank to the sender to find out what it said."*

<http://rz1.razorpoint.com/index.html>

# Cryptanalysis of the Vigenère cipher

It was hard to break with long keys and small amounts of ciphertext

## Cryptanalysis of the Vigenère cipher

### 1. Determine key length

- Count **coincidences** – identical sets of characters  $n$  characters apart
- Key length is likely to be the separation with the maximum # of coincidences
  - Compare likelihood of letter repetition in ciphertext to that of randomly distributed text

### 2. Determine values of each character of the key

- You know the length of the key – that's the # of Caesar ciphers you have
- Do a frequency analysis of each position of the key

# Vernam Cipher (1917): One-time pad

We can achieve maximum entropy by using a truly random key that is as long as the message

- Large non-repeating set of *random* key letters originally written on a pad or punched paper tape for a teleprinter
- Each key letter on the pad encrypts exactly one plaintext character
  - Encryption is exclusive-or of the characters or, for manual operations, the addition of characters modulo *alphabet size* (e.g., 26)
- The sender destroys pad pages that have been used

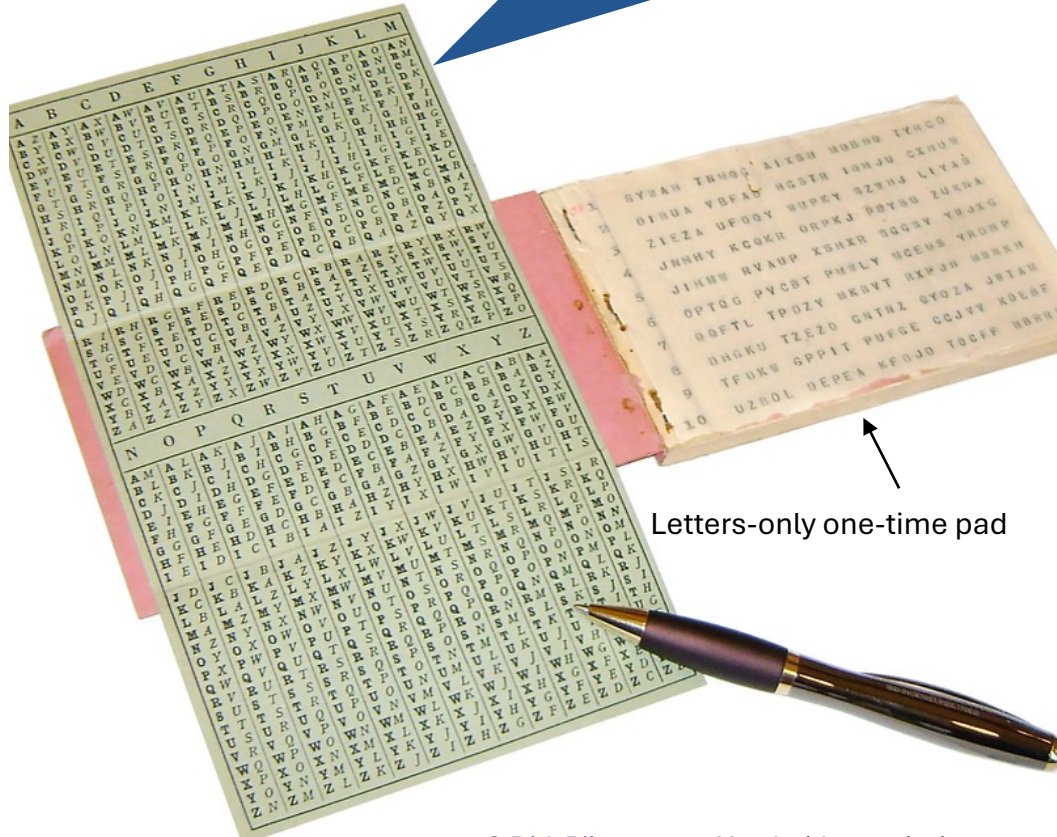
CIHJT	UUHML	FRUGC	ZIBGD	BQPNI	PDNJG	LPLLP	YJYXM
DCXAC	JSJUK	BIOYT	MTQPX	DLIRC	BEXYK	VKIMB	TYIPE
UOLYQ	OKOXH	PIJKY	DRDBC	GEFZG	UACKD	RARCD	HBYRI
DZJYO	YKAIE	LIUYW	DFOHU	IOHZV	SRNDD	KPSSO	JMPQT
MHQHL	OHQQD	SMHNP	HHOHQ	GXRPI	XBXIP	LLZAA	VCMDG
AWSSZ	YMFNI	ATMON	IXPBY	FOZLE	CVYSJ	XZGPU	CTFQY
HOVHU	OCJGU	QMTQV	OIGOR	BFHIZ	TYFDB	VBRMN	XNLZC

**The one-time pad is the only provably secure encryption scheme**

<https://en.wikipedia.org/wiki/File:OneTimePadExcerpt.agr.png>

# Some one-time pads

Reciprocal table  
Used to look up plaintext+ciphertext values



Letters-only one-time pad

A Russian One-time pad, captured by MI5  
Photo from [ramnum.com](http://ramnum.com). Used with permission

© [Dirk Rijmenants](http://Dirk Rijmenants). Used with permission

# One-time pad

If pad contains

KWXOPWMAELGHW...

and we want to encrypt

MY CAT HAS FLEAS

Ciphertext =

WUZOIDSJWKHO

$$M + K \bmod 26 = W$$

$$Y + W \bmod 26 = U$$

$$C + X \bmod 26 = Z$$

$$A + O \bmod 26 = O$$

$$T + P \bmod 26 = I$$

$$H + W \bmod 26 = D$$

$$A + M \bmod 26 = M$$

$$S + A \bmod 26 = S$$

$$F + E \bmod 26 = J$$

$$L + L \bmod 26 = W$$

$$E + G \bmod 26 = K$$

$$A + H \bmod 26 = H$$

$$S + W \bmod 26 = O$$



# One-time pad

The same ciphertext can decrypt to *anything* depending on the key!

You can come up with a key to decrypt the ciphertext to any message

Same ciphertext:

WUZOIDMSJWKHO

With a pad containing:

DNVLUXEACWVSQ...

Produces:

THE DOG IS HAPPY

W	-	D	mod	26	=	T
U	-	N	mod	26	=	H
Z	-	V	mod	26	=	E
O	-	L	mod	26	=	D
I	-	U	mod	26	=	O
D	-	X	mod	26	=	G
M	-	E	mod	26	=	I
S	-	A	mod	26	=	S
J	-	C	mod	26	=	H
W	-	W	mod	26	=	A
K	-	V	mod	26	=	P
H	-	S	mod	26	=	P
O	-	Q	mod	26	=	Y

## Digression: exclusive-or

# Boolean logic refresher: AND

## AND ( $\wedge$ ): clears bits

AND clears bits

- AND 1 – keep the bit
- AND 0 – clear the bit

### Truth table

$$1 \wedge 1 = 1$$

$$1 \wedge 0 = 0$$

$$0 \wedge 1 = 0$$

$$0 \wedge 0 = 0$$

If you clear a bit, you will never know if it used to be a 0 or a 1

**Note:** I'm using Boolean algebra notation here. The BCPL and B languages used & and | for AND and OR, respectively – for both logical (conditional) and bitwise operations. C, the successor to B, introduced the use of ^ for a bitwise XOR operation and && and || for logical AND and OR operations. C++, C#, D, Java, Perl, Ruby, PHP and Python followed the convention of using ^ as an exclusive-or operator.

# Boolean logic refresher: OR

OR ( $\vee$ ): sets bits

OR sets bits

- OR 1 – set the bit
- OR 0 – keep the bit

Truth table

$$1 \vee 1 = 1$$

$$1 \vee 0 = 1$$

$$0 \vee 1 = 1$$

$$0 \vee 0 = 0$$

If you set a bit, you will never know if it used to be a 0 or a 1

# Boolean logic refresher: Exclusive-OR (XOR)

XOR ( $\oplus$ ): flips bits

XOR flips bits

- XOR 1 – flip the bit
- XOR 0 – keep the bit as it is

Truth table

$$1 \oplus 1 = 0$$

$$1 \oplus 0 = 1$$

$$0 \oplus 1 = 1$$

$$0 \oplus 0 = 0$$

If you flip a bit, you can restore it by XORing it with 1 again

# XOR in cryptography

We use XOR operations a lot in cryptography

They allow us to flip certain bits to encrypt and later unflip to decrypt

	0	1	1	1	0	0	0	0
$\oplus$	1	0	1	0	1	1	0	0
<hr/>								
	1	1	0	1	1	1	0	0
$\oplus$	1	0	1	0	1	1	0	0
<hr/>								
	0	1	1	1	0	0	0	0

0	1	1	0	1	0	1	1
1	0	1	1	1	0	0	1
<hr/>							
1	1	0	1	0	0	1	0
1	0	1	1	1	0	0	1
<hr/>							
0	1	1	0	1	0	1	1

plaintext = 0x70 0x6b

$\oplus$  key = 0xac 0xb9

ciphertext = 0xdc 0xd2

$\oplus$  key = 0xac 0xb9

result = 0x70 0x6b

*plaintext recovered!*

End of digression

# One-time pads in computers

One-time pads easily work with arbitrary binary data

- Random key sequence as long as the message
- Exclusive-or key sequence with message
- Receiver has the same key sequence



# One-time pad – C code

```
void onetimepad(void)
{
    FILE *if = fopen("intext", "rb");
    FILE *kf = fopen("keytext", "rb");
    FILE *of = fopen("outtext", "wb");
    int c, k;

    while ((c = getc(if)) != EOF) {
        k = getc(kf);
        putc((c^k), of);
    }
    fclose(if); fclose(kf); fclose(of);
}
```

# One-time pad – Python code

```
def onetimepad():  
    with open("intext", "rb") as if_file, \  
        open("keytext", "rb") as kf_file, \  
        open("outtext", "wb") as of_file:  
  
        while True:  
            c = if_file.read(1)  
            if not c: # EOF condition  
                break  
            k = kf_file.read(1)  
            if not k: # Key exhausted (shouldn't happen)  
                break  
  
            result = bytes([c[0] ^ k[0]])  
            of_file.write(result)
```

Since read(1) returns bytes, we access the actual byte value with [0] and wrap the XOR result back in a bytes() object

# One-time pads provide **perfect secrecy**

## Perfect secrecy

- Ciphertext conveys no information about the content of plaintext
- *Achieved only if the key is random and as long as the plaintext*

Problems with one-time pads:

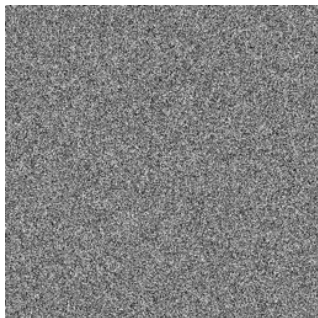
- **The key needs to be as long as the message!**
- Key storage and distribution can be problematic
- Keys must be generated **randomly**
  - Cannot use a pseudo-random number generator
- Cannot reuse key sequence
- Sender and receiver *must* remain synchronized (e.g., cannot lose any part of the message)

# Reusing a key with a one-time pad

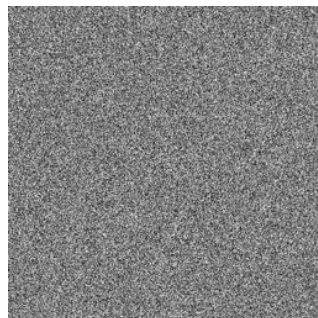
We're using a one-time pad to encrypt two image files using a random key as long as the message.  
But the same key was used for the two images.



img1



key1



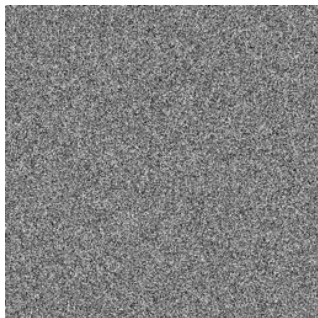
img1.enc



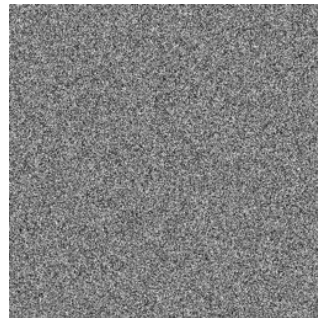
By reusing the same key,  
information can be leaked



img2



key1



img2.enc

# Random numbers

*“Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin”*

– John von Neumann, 1951

- Pseudo-random generators
  - Linear feedback shift registers
  - Multiplicative lagged Fibonacci generators
  - Linear congruential generator
- Obtain randomness from:
  - Time between keystrokes
  - Various network/kernel events
  - Cosmic rays
  - Electrical noise, thermal noise
  - Other encrypted messages

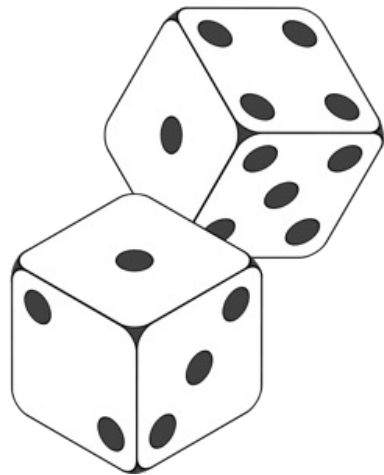


Image from Wikimedia Commons

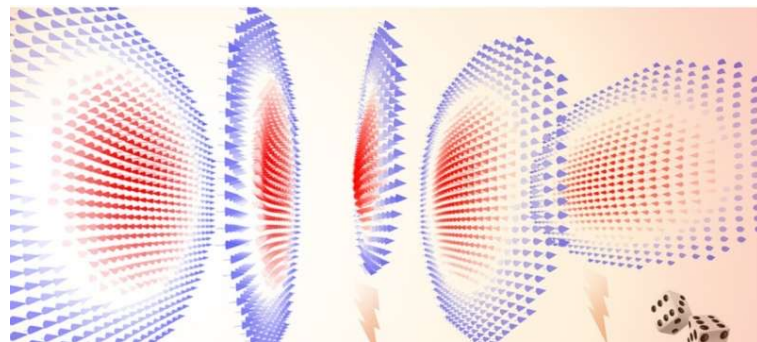
## Researchers use tiny magnetic swirls to generate true random numbers



by Brown University  
February 7, 2022

Whether for use in cybersecurity, gaming or scientific simulation, the world needs true random numbers, but generating them is harder than one might think. But a group of Brown University physicists has developed a technique that can potentially generate millions of random digits per second by harnessing the behavior of skyrmions—tiny magnetic anomalies that arise in certain two-dimensional materials.

Their research, published in Nature Communications, reveals previously unexplored dynamics of single skyrmions, the researchers say. Discovered around a half-decade ago, skyrmions have sparked interest in physics as a path toward next-generation computing devices that take advantage of the magnetic properties of particles—a field known as spintronics.



Magnetic swirls called skyrmions fluctuate randomly in size, a behavior that can be harnessed to generate true random numbers. Credit: Xiao lab / Brown University

<https://phys.org/news/2022-02-tiny-magnetic-swirls-true-random.html>

# Ongoing research for random number generators

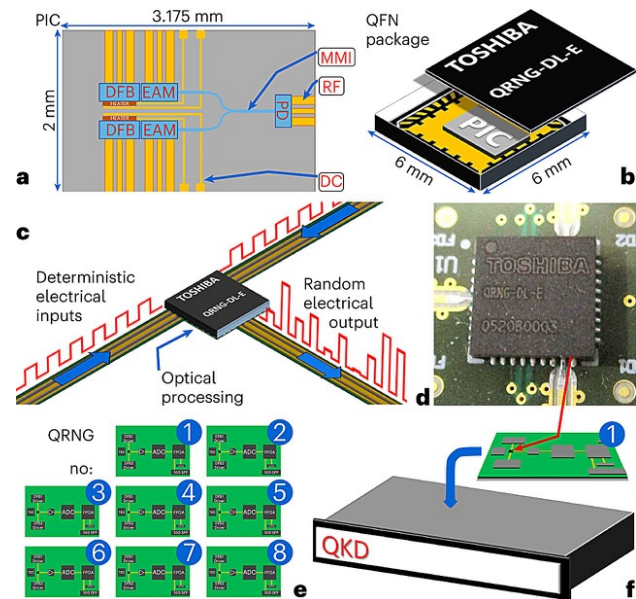
## New quantum random number generator achieves 2 Gbit/s speed

by Ingrid Fadelli

June 11, 2024

The reliable generation of random numbers has become a central component of information and communications technology. In fact, random number generators, algorithms or devices that can produce random sequences of numbers, are now helping to secure communications between different devices, produce statistical samples, and for various other applications.

Researchers at Toshiba Europe Ltd. recently developed a new quantum random number generator (QRNG) based on a photonic integrated circuit that can be directly integrated in electronic devices. This QRNG, introduced in a paper published in *Nature Electronics*, can securely and robustly generate random numbers at a remarkable speed of  $2 \text{ Gbit s}^{-1}$ .



<https://techxplore.com/news/2024-06-quantum-random-generator-gbits.html>

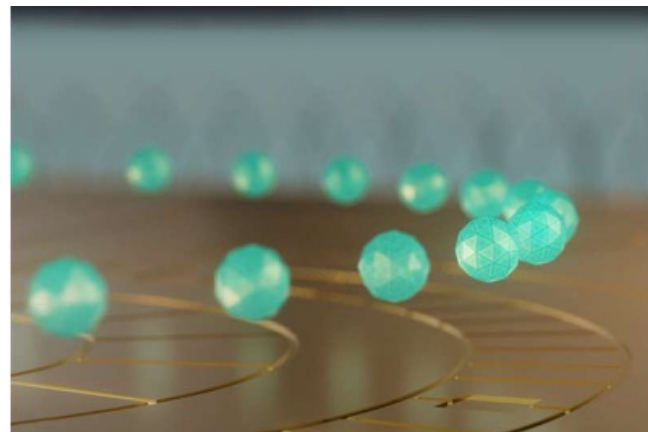
## New quantum random number generator achieves 2 Gbit/s speed



by University of Texas at Austin  
March 26, 2025

In a new paper in Nature, a team of researchers from JPMorganChase, Quantinuum, Argonne National Laboratory, Oak Ridge National Laboratory and The University of Texas at Austin describe a milestone in the field of quantum computing, with potential applications in cryptography, fairness and privacy.

Using a 56-qubit quantum computer, they have for the first time experimentally demonstrated certified randomness, a way of generating random numbers from a quantum computer and then using a classical supercomputer to prove they are truly random and freshly generated. This could pave the way toward the use of quantum computers for a practical task unattainable through classical methods.



Using a 56-qubit quantum computer, researchers have for the first time experimentally demonstrated a way of generating random numbers from a quantum computer and then using a classical supercomputer to prove they are truly random and freshly generated. Credit: Quantinuum

<https://www.pikpng.com/transpng/hxToTow>



# Ongoing research for random number generators

## NIST and Partners Use Quantum Mechanics to Make a Factory for Random Numbers

NIST

June 11, 2025

- NIST and its partners at the University of Colorado Boulder built the first random number generator that uses quantum entanglement to produce verifiable random numbers.
- Broadcast as a free public service, the Colorado University Randomness Beacon (CURBy) can be used anywhere an independent, public source of random numbers would be useful, such as selecting jury candidates or assigning resources through a public lottery.
- CURBy uses a NIST-developed protocol called Twine that allows data from the beacon to be traced, verified and protected from manipulation.

The process starts by generating a pair of entangled photons inside a special nonlinear crystal. The photons travel via optical fiber to separate labs at opposite ends of the hall. Once the photons reach the labs, their polarizations are measured. The outcomes of these measurements are truly random. This process is repeated 250,000 times per second.



Instrumentation for the quantum random number generator in the NIST Boulder laboratories (Credit: NIST).

<https://www.nist.gov/news-events/news/2025/06/nist-and-partners-use-quantum-mechanics-make-factory-random-numbers>

# Lavarand – Lava Lamp Randomness

- The fluid dynamics of a lava lamp create chaotic, unpredictable movements with high entropy
- An initial lava lamp based random number generator was deployed by SGI: 1997-2001
- Cloudflare uses a wall of ~100 lava lamps with a camera pointed at the lamps.
- Each image becomes a random seed for generating encryption keys



See <https://www.cloudflare.com/learning/ssl/lava-lamp-encryption/> for a description of lava lamp based random number generation and Cloudflare's need for good sources of randomness.

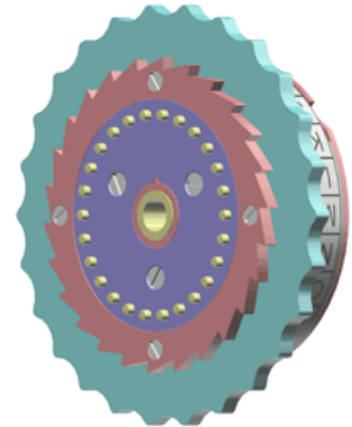
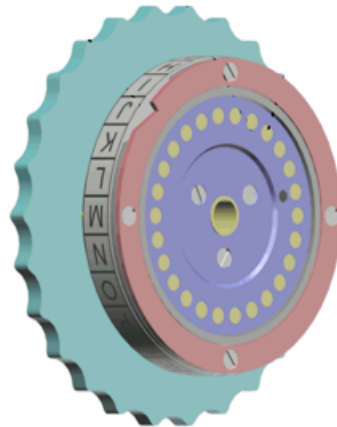
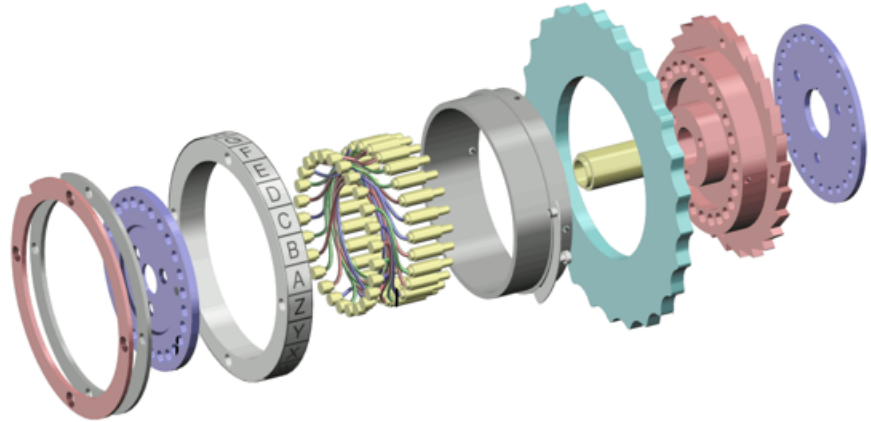
# Rotor machines

1920s: mechanical devices used for automating encryption

## Rotor machine:

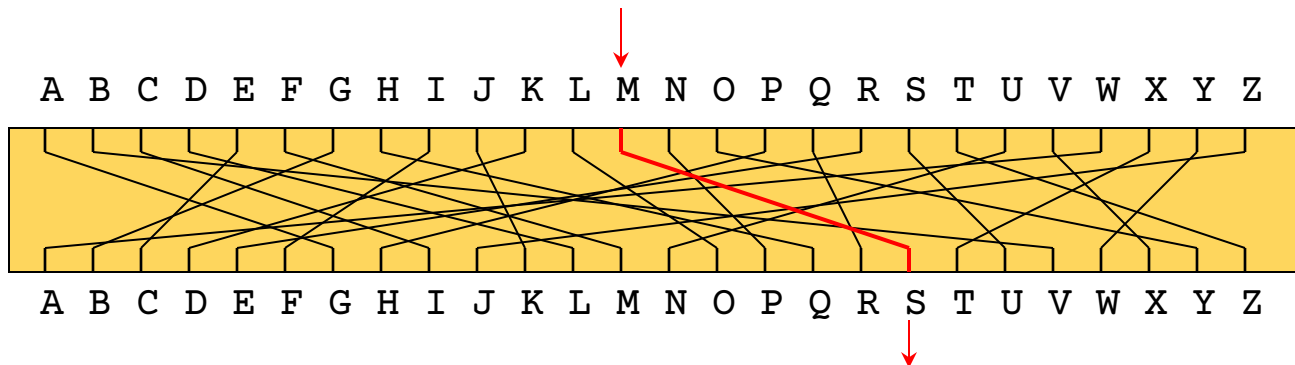
- Set of independently rotating cylinders (rotors) through which electrical pulses flow
- Each rotor has input & output pin for each letter of the alphabet
  - Each rotor implements a substitution cipher
- Output of each rotor is fed into the next rotor
- Together they implement a version of the Vigenère cipher (with a long period)

# The structure of a rotor in a rotor machine



# Rotor machines

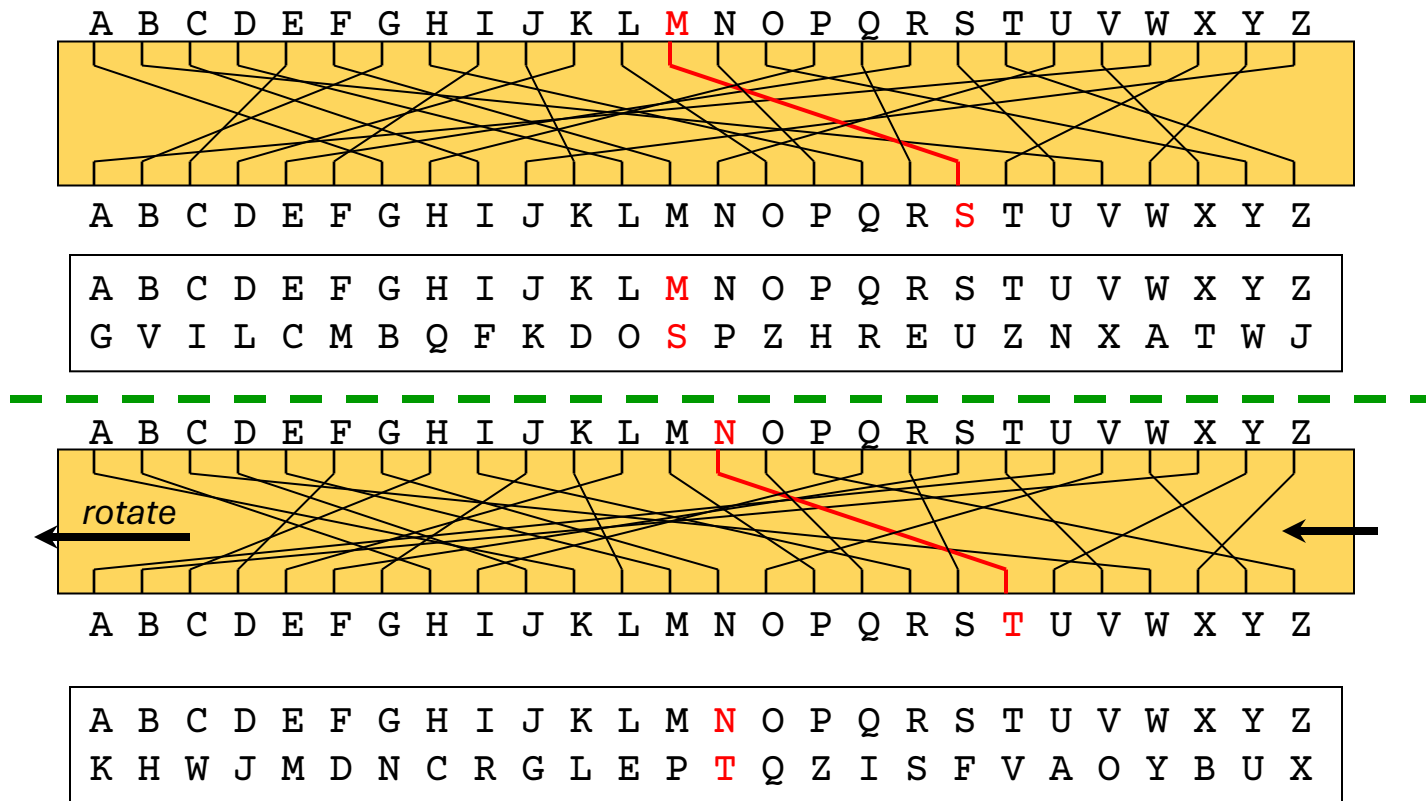
## Simplest rotor machine: single cylinder



After a character is entered, the cylinder rotates one position

- Internal connections shifted by one
- Polyalphabetic substitution cipher with a period of 26

# Single cylinder rotor machine



# Multi-cylinder rotor machines

- Single cylinder rotor machine
  - Substitution cipher with a period = length of the alphabet (e.g., 26)
- Multi-cylinder rotor machine
  - Feed output of one cylinder as input to the next one
  - First rotor advances after the character is entered
  - The second rotor advances after a full period of the first
  - Polyalphabetic substitution cipher

Period = (length of alphabet)<sup>number of rotors</sup>

  - 3 26-char cylinders  $\Rightarrow 26^3 = 17,576$  substitution alphabets
  - 5 26-char cylinders  $\Rightarrow 26^5 = 11,881,367$  substitution alphabets

# Enigma

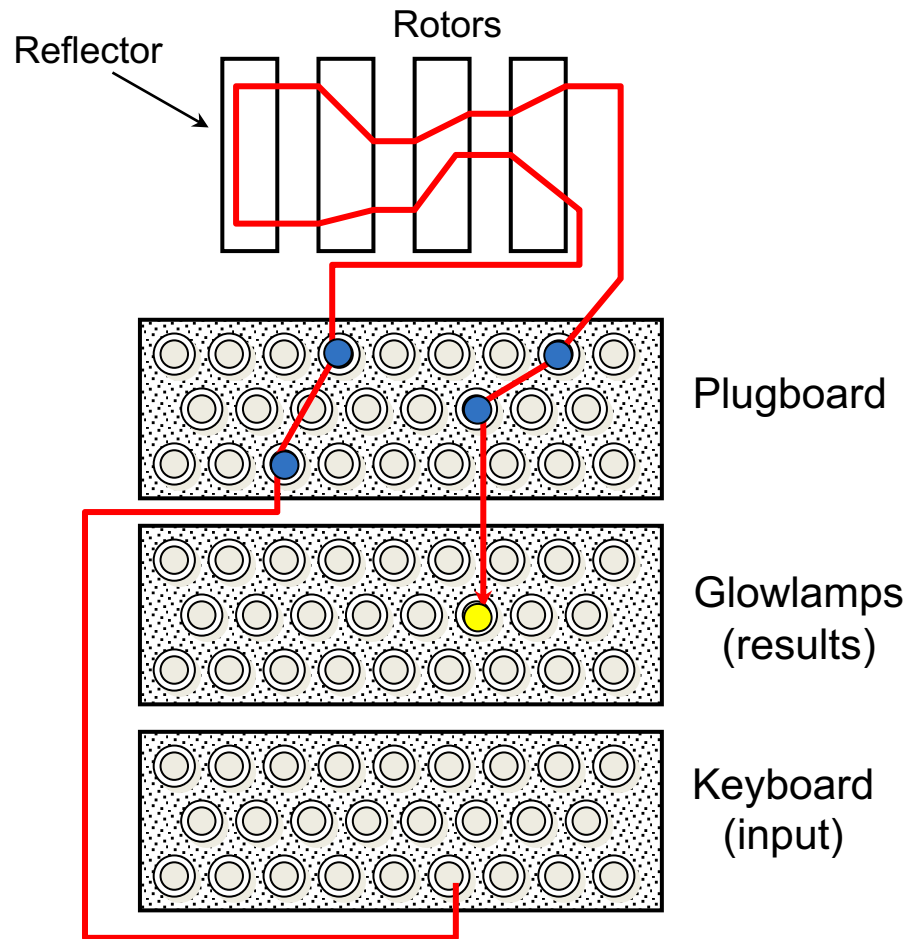
- Enigma machine used in Germany during WWII
- Three rotor system
  - $26^3 = 17,576$  possible rotor positions
- Input data permuted via patch panel before sending to rotor engine
- Data from last rotor reflected back through rotors  
⇒ **makes encryption symmetric**
- All parties agree on the initial settings of the rotors
  - Setting was  $f(\text{date})$  in a book of codes
- Broken by Marian Rejewski of the Polish Cipher Bureau
  - Alan Turing at Bletchley Park designed the British bombe device to identify the rotor order



See [https://en.wikipedia.org/wiki/Cryptanalysis\\_of\\_the\\_Enigma](https://en.wikipedia.org/wiki/Cryptanalysis_of_the_Enigma)



# Enigma



# Enigma successor: Soviet Union's Fialka

The Soviet Union needed a more secure cipher than Enigma

- Rotor machine with 10 rotors and 30 contacts/rotor
  - Fialka:  $30^{10} = 5.9 \times 10^{14}$  possible rotor positions
  - Enigma:  $26^3 = 17,576$  possible rotor positions
- Punched cards configure initial settings
- More complex stepping mechanism than Enigma
- Various versions used by Warsaw Pact countries into the 1990s



# Soviet Union Fialka – 10-rotor system



# Classic Cryptosystems: Transposition Ciphers

# Transposition ciphers

- Permute letters in plaintext according to specific rules
- Knowledge of rules will allow messages to be decrypted
- First documented use by Spartans in the 5<sup>th</sup> century BCE
  - **Scytale** (*rhymes with Italy*) = staff cipher

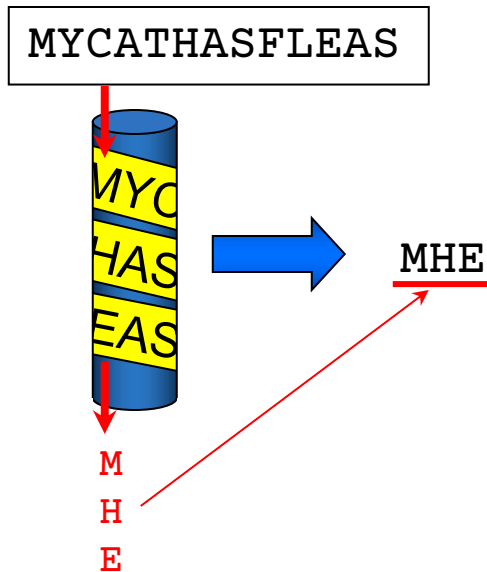
## Transposition ciphers:

- Scramble the letters of the plaintext
- Split common letter sequences to increase the entropy of digraphs and trigraphs

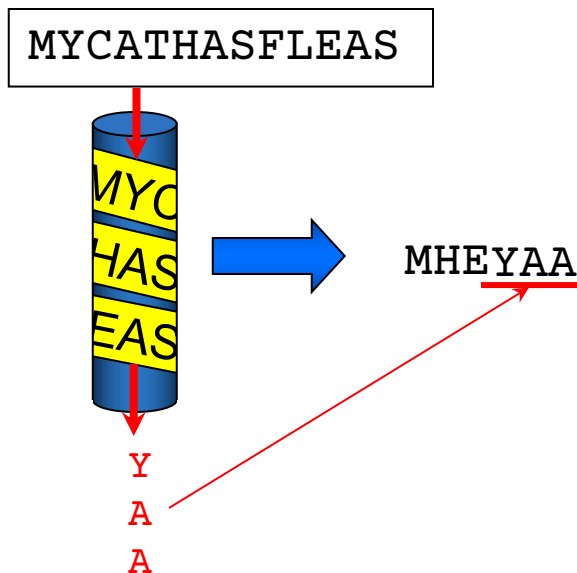


# Transposition ciphers: **scytale**

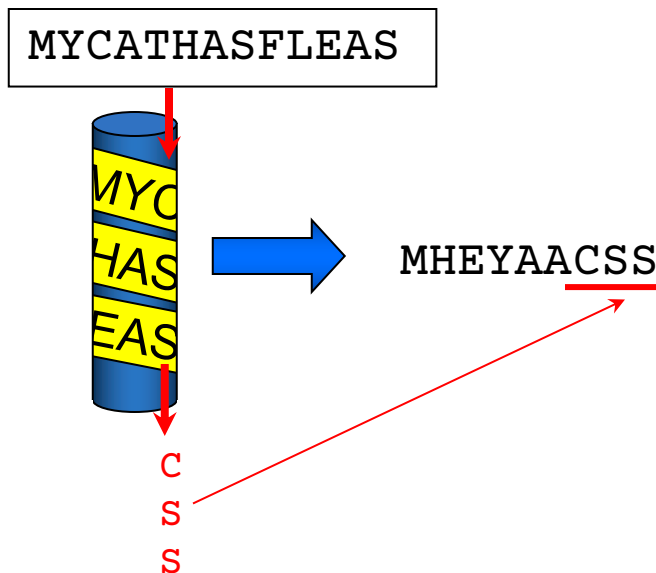
Secret = diameter of scytale



# Transposition ciphers: **scytale**

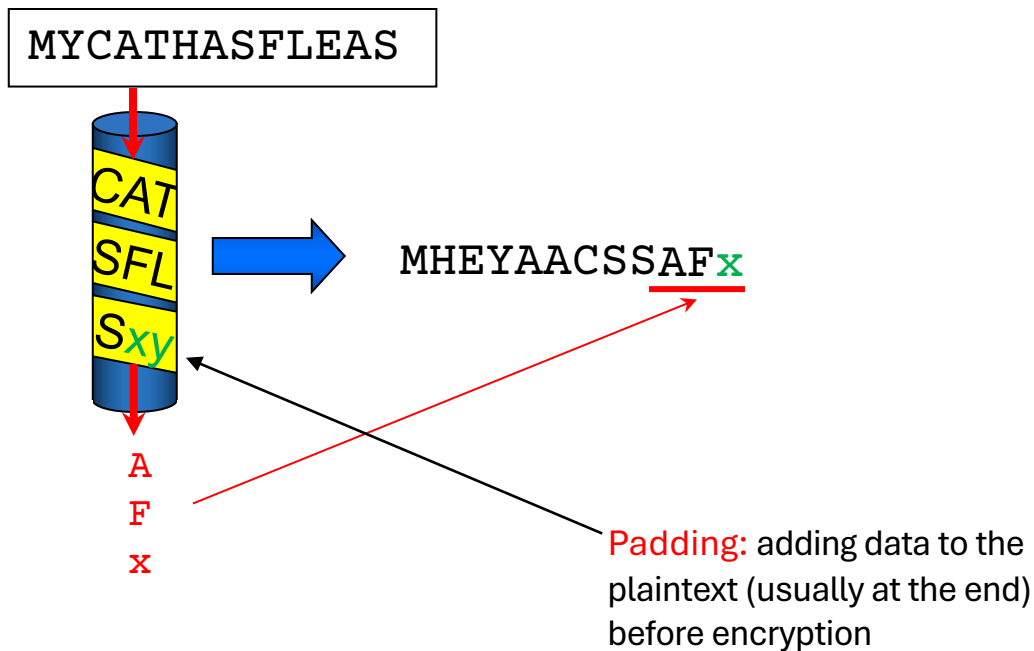


# Transposition ciphers: **scytale**

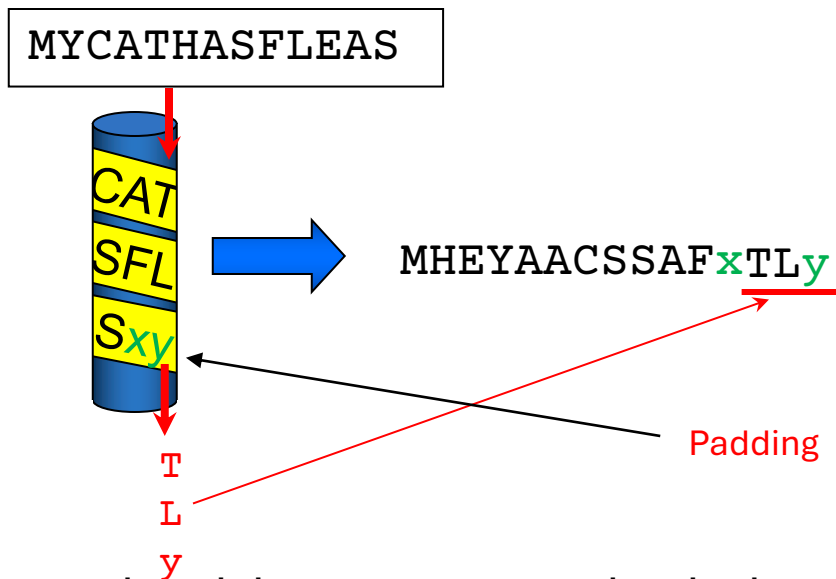




## Transposition ciphers: **scytale**



# Transposition ciphers: **scytale**

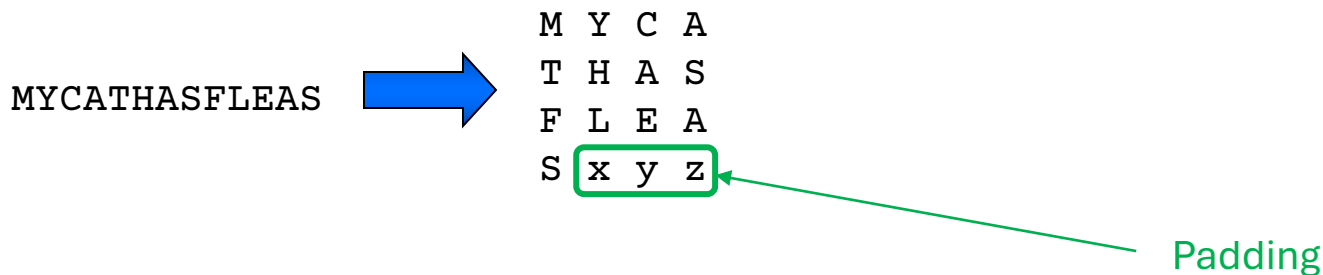


**Padding:** adding extra data to the plaintext to ensure that its length is a multiple of a specific block size

# Scytale as a set of columns

## Table version of scytale

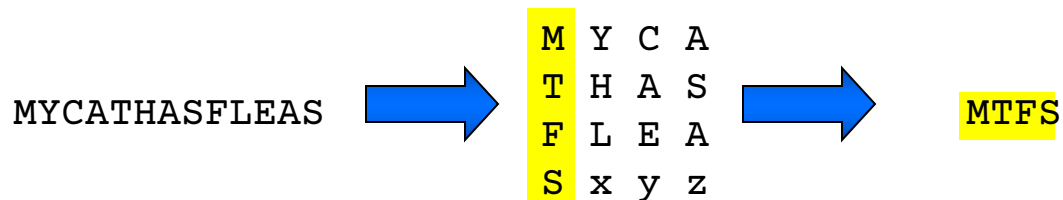
- Enter data horizontally, read it vertically
- Secrecy is the width of the table



# Scytale as a set of columns

## Table version of scytale

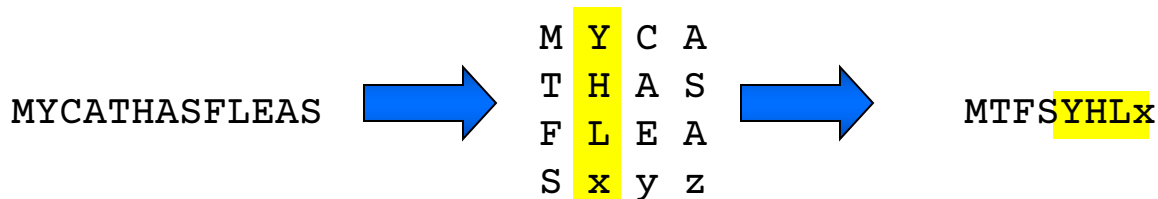
- Enter data horizontally, read it vertically
- Secrecy is the width of the table



# Scytale as a set of columns

## Table version of scytale

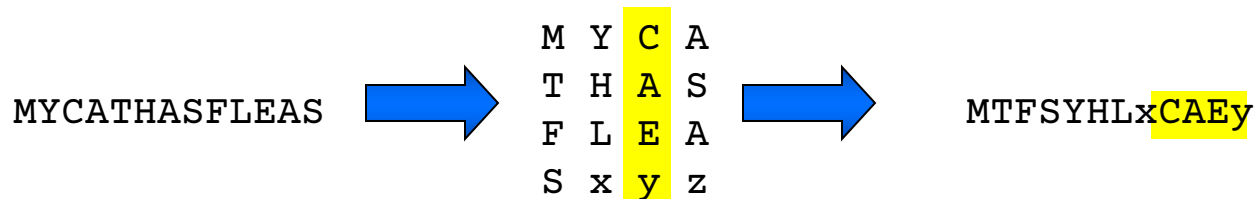
- Enter data horizontally, read it vertically
- Secrecy is the width of the table



# Scytale as a set of columns

## Table version of scytale

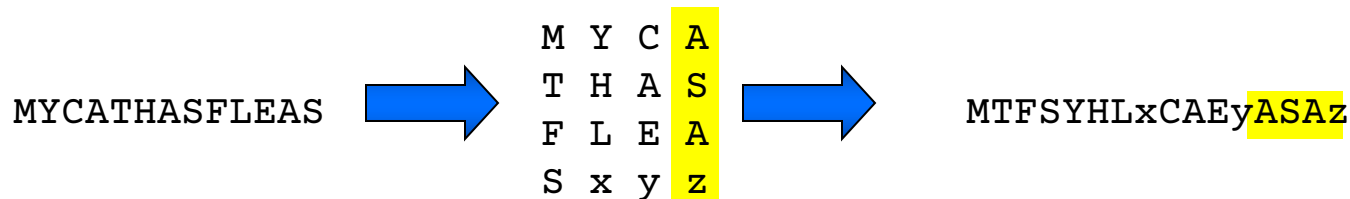
- Enter data horizontally, read it vertically
- Secrecy is the width of the table



# Scytale as a set of columns

## Table version of scytale

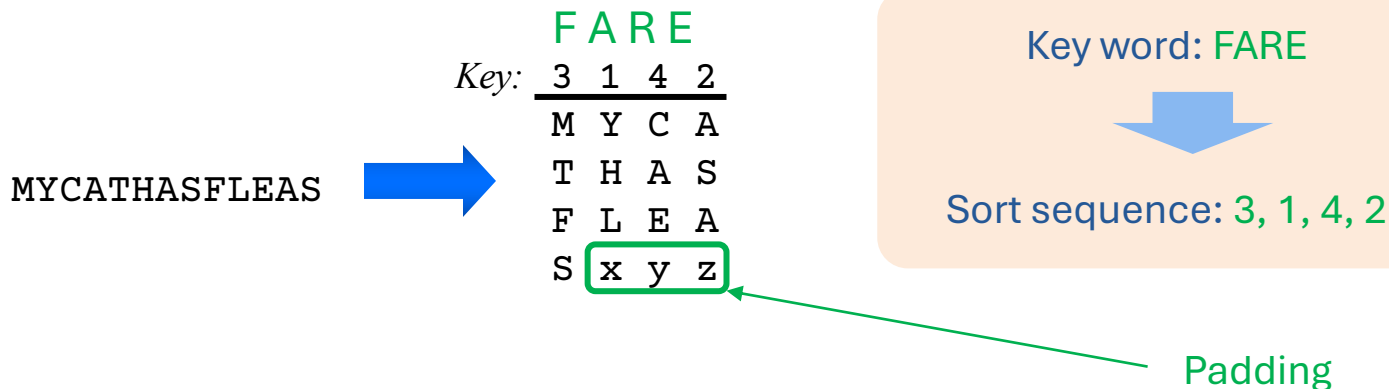
- Enter data horizontally, read it vertically
- Secrecy is the width of the table



# Columnar transposition cipher

## Add a key

- Created in the mid 1600s – used by the French, Japanese, & Russians into the 20<sup>th</sup> century
- Key word defines the width of the table and the sequence of reading the columns
- Read down columns, sorting by key letters

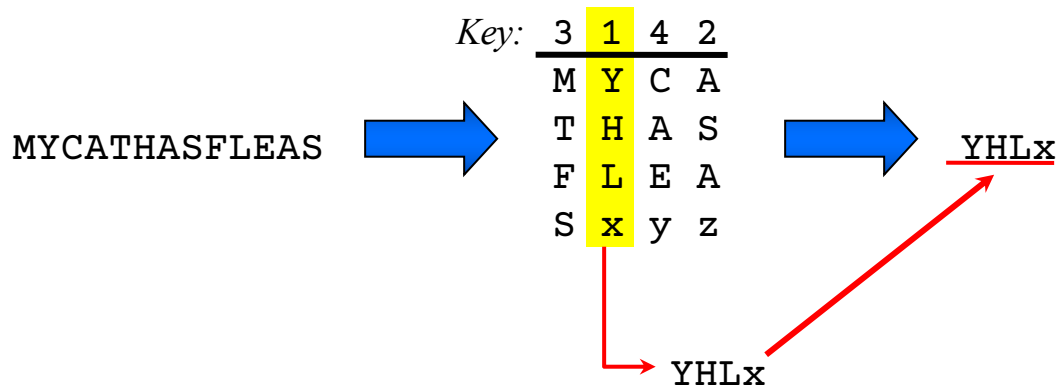




# Columnar transposition cipher

## Add a key

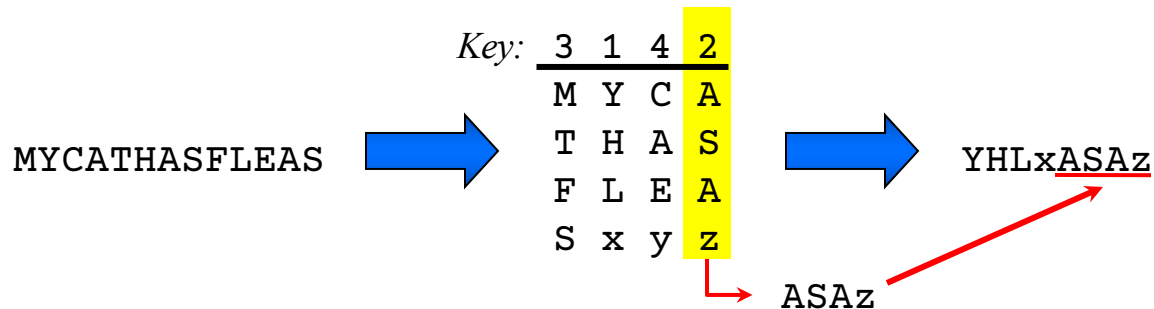
- Shuffle columns based on the sorting of letters in the key word
- Read down columns



# Columnar transposition cipher

## Add a key

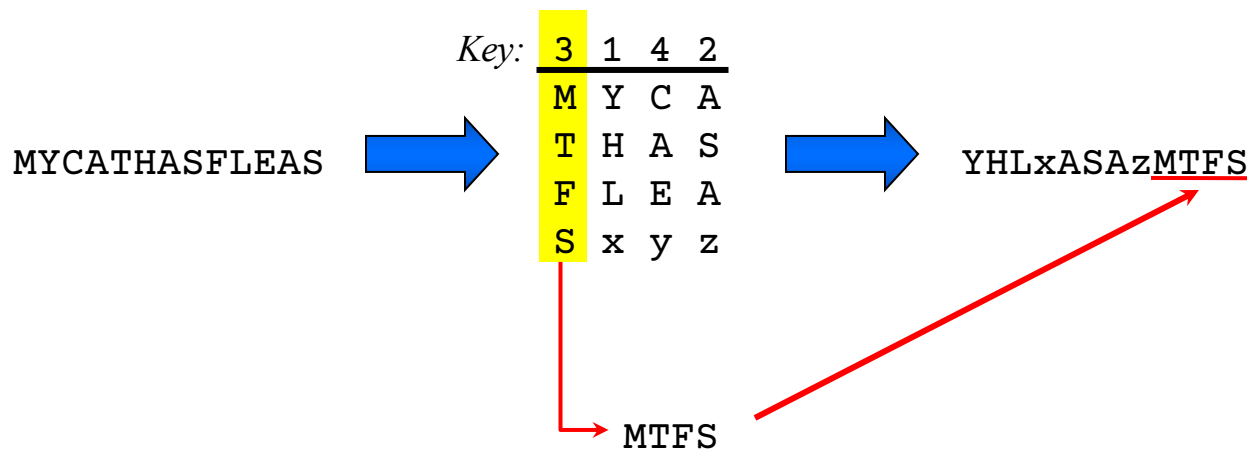
- Shuffle columns based on the sorting of letters in the key word
- Read down columns



# Columnar transposition cipher

## Add a key

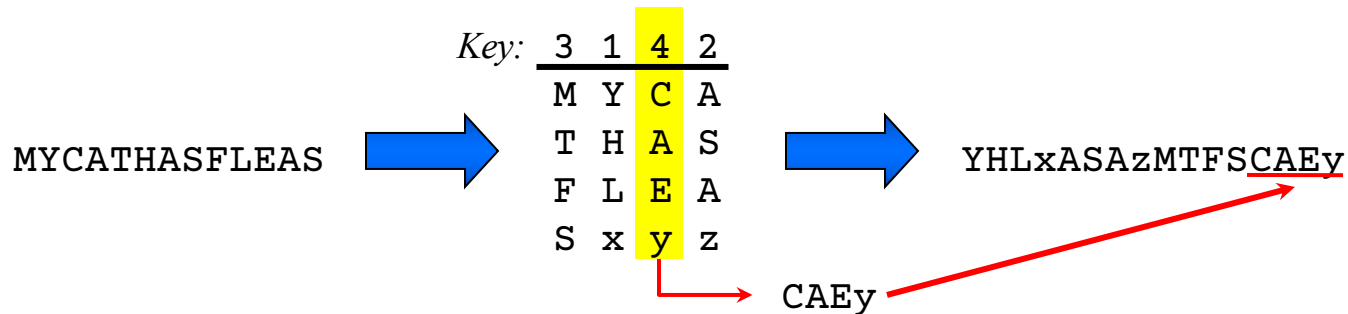
- Shuffle columns based on the sorting of letters in the key word
- Read down columns



# Columnar transposition cipher

## Add a key

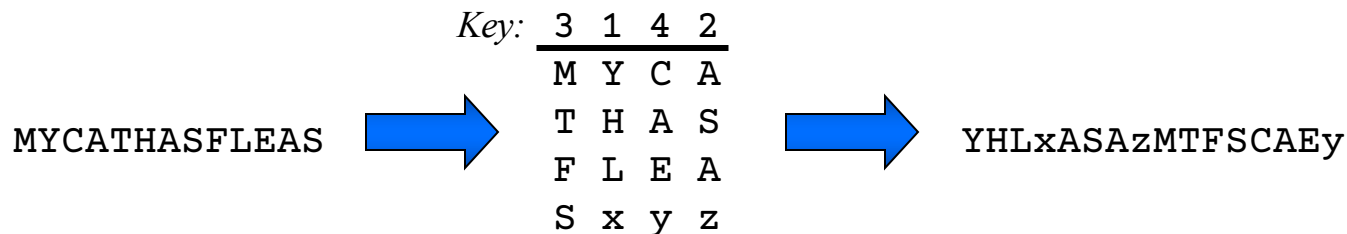
- Shuffle columns based on the sorting of letters in the key word
- Read down columns



# Columnar transposition cipher

## Add a key

- Shuffle columns based on the sorting of letters in the key word
- Read down columns



# Transposition cipher

- Not vulnerable to frequency analysis
- Entropy of characters does not change
  - But the entropy of digraphs and trigraphs increases because common sequences are broken through scrambling the characters
- The scytale is trivial to attack
  - Make all possible matrices that would fit the ciphertext
  - Write ciphertext across rows
  - See if the columns contain legible content
- Scrambled columns make it a bit harder
  - Need to permute columns of matrices

# Combined ciphers

- Combine transposition with substitution ciphers
  - German ADFGVX cipher (1918, World War I)
  - Playfair cipher (Charles Wheatstone, 1854)
- Great for increasing entropy of characters, digraphs, trigraphs, ...
- But was troublesome to implement (before computers)
  - Difficult with pencil-and-paper or electromechanical cryptography
  - Except for the simplest ciphers, requires memory to store blocks of data for transposition

# Principles of a good cryptosystem

1. **Foundational principles:** timeless rules & design philosophies
2. **Security principles:** technical characteristics to make ciphers resilient
3. **Practical requirements:** ensure the cipher is usable in real-world systems



# 1. Foundational Principles

These are guiding ideas that underpin all secure cipher design

- **Kerckhoffs's Principle – 1883**

- Auguste Kerckhoffs – 19<sup>th</sup> century Dutch linguist and cryptographer
- Published *La Cryptographie Militaire* in 1883

- **Shannon's Principles – 1949**

- American mathematician and founder of modern information theory
- Published *Communication Theory of Secrecy Systems* in 1949

# Kerckhoffs's Principle of Cryptography

Kerckhoffs's Principle:

*A cryptosystem should be secure even if everything about the system, except the key, is public knowledge*

In other words ...

*“One ought to design systems under the assumption that the enemy will immediately gain full familiarity with them.” - Claude Shannon, 1949*

What this means:

If you don't know the key, you will have to do an exhaustive search (try all combinations). The amount of effort to decrypt data without knowing the key is proportional to the length of the key

# Shannon's Properties: Confusion and Diffusion

Claude Shannon defined two operational goals of a cipher:

- **Confusion:** *hide the relationship between the key and ciphertext*
  - There is no direct correlation between a bit of the key and the resulting ciphertext
  - Every bit of ciphertext depends on various bits of the key. You cannot find a connection between a bit of the key and a bit of the ciphertext.
  - Generally implemented through substitution
- **Diffusion:** *spread information about the plaintext widely across the ciphertext*
  - The plaintext information is **spread** throughout the cipher so that a change in one bit of plaintext will change, on average, half of the bits in the ciphertext will change.
  - Generally implemented through transposition
  - The goal is to make the relationship between the plaintext and ciphertext complicated

## 2. Security Properties (a-c)

These describe what makes a cipher resistant to attack.

a.	<b>Indistinguishability from random</b>	Ciphertext should appear statistically random, with high entropy and no detectable patterns that leak information about plaintext or key.
b.	<b>Non-invertibility without the key</b>	There must be no shortcut to recovering plaintext or key except brute force. This is true even if the attacker has many $(P, C)$ pairs or can supply an arbitrary amount of plaintext to be encrypted and see the resulting ciphertext.
c.	<b>Adequate key space</b>	Keys must be long enough to make exhaustive search infeasible. <i>A cipher is no stronger than its key length:</i> <ul style="list-style-type: none"><li>• If the key is too short, an attacker may be able to perform a brute-force search.</li><li>• A cipher may be a lot weaker than its key length.</li></ul>

## 2. Security Properties (d-h)

d.	<b>No weak keys</b>	No subset of keys should result in weak or easily breakable encryption.
e.	<b>No restrictions</b>	All key values and plaintext values should be valid, without special cases that simplify attacks.
f.	<b>Resistance to attack models</b>	The cipher should withstand ciphertext-only, known-plaintext, chosen-plaintext, and chosen-ciphertext attacks.
g.	<b>Robust against side channels</b>	The cipher should not leak information through timing, power use, or memory access patterns.
h.	<b>Composability</b>	The cipher must remain secure when used repeatedly or combined with other operations, since real systems rarely encrypt just one block in isolation.

### 3. Practical Requirements

These ensure the cipher is not only secure but also usable.

a.	<b>Efficiency</b>	Encryption and decryption should be fast enough to encourage consistent use.
b.	<b>Minimal expansion</b>	Ciphertext should not grow significantly larger than plaintext. Small overheads are acceptable but not multiples of the plaintext size.
c.	<b>Simplicity and universality</b>	Algorithms should be implementable in straightforward ways and support arbitrary keys and plaintexts.
d.	<b>Transparency</b>	The algorithm should be public and subjected to extensive cryptanalysis before widespread use.

# About keys

Keys should be

**Protected**

If attackers get hold of the keys, they can decrypt ciphertext

**Random**

Keys should be unpredictable

**Short term (ideally): Used for a limited time**

Less (P,C) data gives cryptanalysts less data to analyze

# Keys and the power of 2

- Each extra bit added to a key doubles the search space
- Suppose it takes 1 millisecond to search through all keys with a 20-bit key

key length	number of keys	search time*	10,000 computers
20 bits	1,048,576	0.001 seconds	0.0000001 seconds
21 bits	2,097,152	0.002 seconds	0.0000002 seconds
32 bits	$4.3 \times 10^9$	4.1 seconds	.0004 seconds
56 bits	$7.2 \times 10^{16}$	2.2 years	2 hours
64 bits	$1.8 \times 10^{19}$	557 years	20 days
128 bits	$3.4 \times 10^{29}$	$1.03 \times 10^{22}$ years	$1.03 \times 10^{18}$ years
256 bits	$1.2 \times 10^{77}$	$1.1 \times 10^{67}$ years	$1.1 \times 10^{64}$ years

The universe is estimated to be  $1.3 \times 10^{10}$  years old

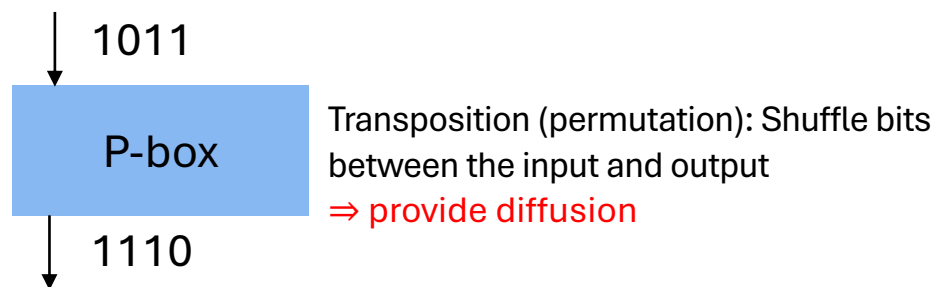
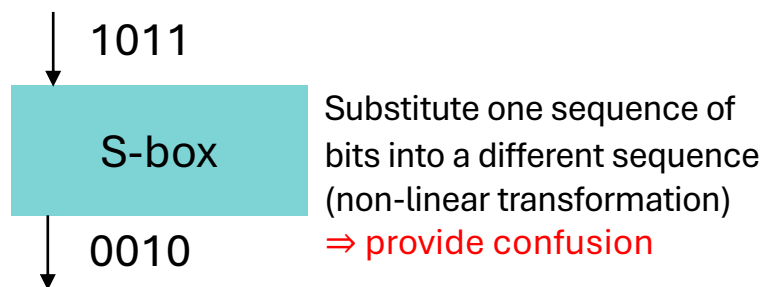
\*Assume the computer can test 1 billion keys per second



# Computer Cryptography

# Properties

- Operate on arbitrary binary data:  $P = \{0, 1\}^n$
- Kerckhoffs's Principle: the secrecy resides in the key
- Shannon's properties
  - **Confusion**: no direct correlation between a bit of the key and the resulting ciphertext
  - **Diffusion**: Changing one bit of input should change, on average,  $\frac{1}{2}$  of output bits
- Main mechanisms

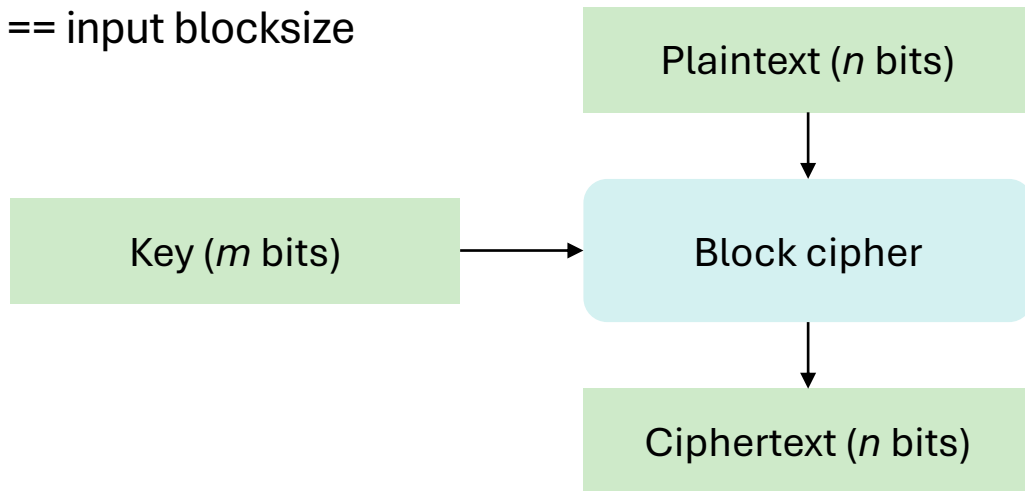


# Block ciphers

Block ciphers dominate computer cryptography

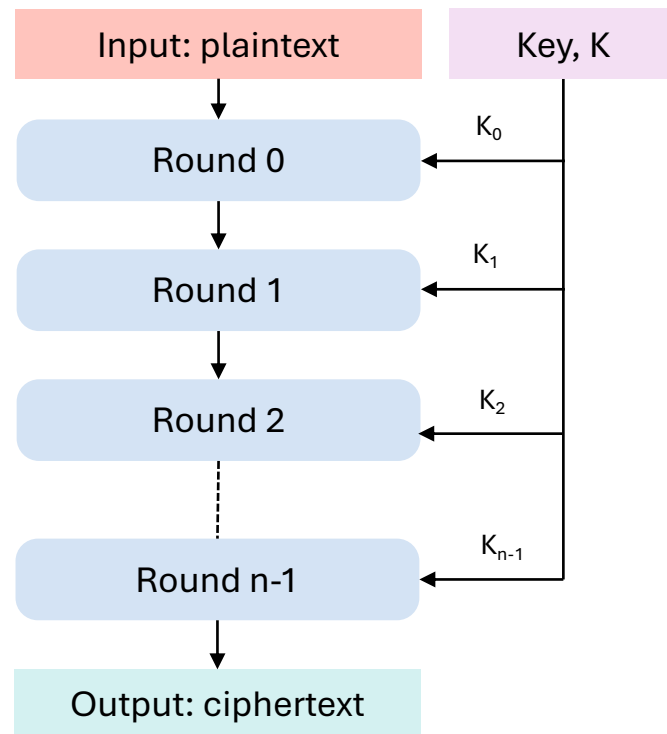
Encrypt a fixed number of bits (a *block*) at a time

Output blocksize (usually) == input blocksize



# Block ciphers

- Block ciphers encrypt a block of plaintext at a time
- DES & AES are two popular block ciphers
  - DES: 64-bit blocks
  - AES: 128-bit blocks
- Block ciphers are usually **iterative ciphers**
  - The encryption process is an iteration through several **round** operations
  - A single round does not provide perfect confusion or diffusion



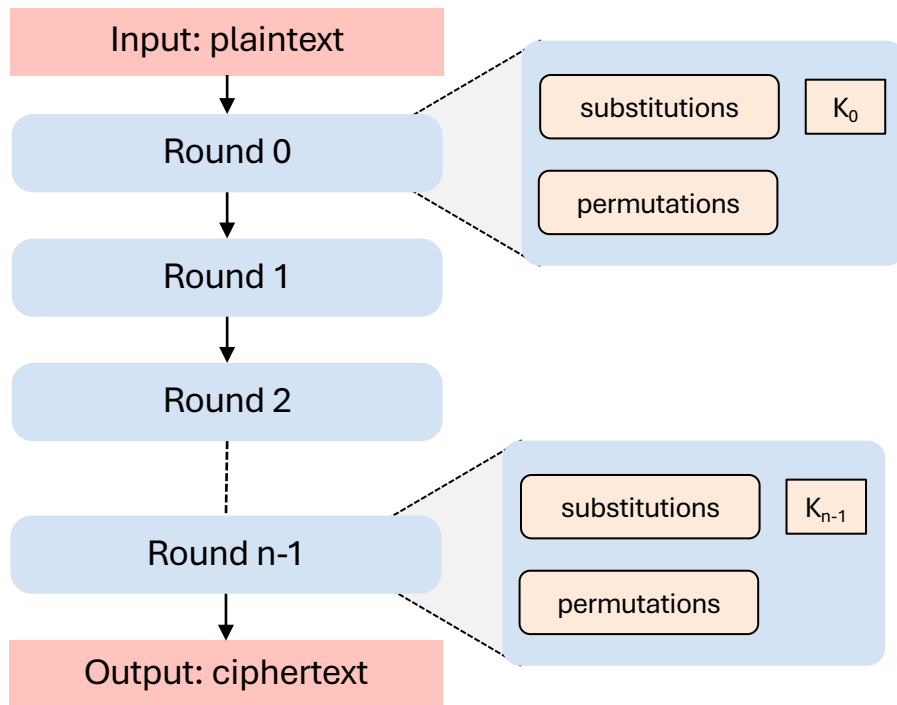
# Structure of block ciphers

- Multiple **rounds** of combining the plaintext with the key
- Optional:
  - Convert key to an internal **subkey** – different for each round
- DES: 16 rounds
- AES: 10-14 rounds, depending on key length

Sounds easy ... but is difficult to design

# Block cipher rounds

Each round consists of substitutions & permutations = **SP Network**



## Substitution = **S-box**

- Table lookup
- Converts a small block of input to a block of output

## Permutation

- Scrambles the bits in a prescribed order

## Key application per round

- **Subkey**,  $K_n$ , per round derived from the key
- Can drive behavior of s-boxes
- May be XORed with the output of each round

## Create Confusion & Diffusion

- **Confusion**: no direct correlation between a bit of the key and resulting ciphertext
- **Diffusion**: Changing one bit of input should change, on average,  $\frac{1}{2}$  of output bits

# Two Families of Block Cipher Designs

**Both designs use multiple rounds (iterations) and derive a per-round subkey from the master key**

- **Feistel Network**

- Each block is split into two sub-blocks
- One sub-block is permuted with a function per round and exclusive-ORed with the other sub-block
- The sub-block are then swapped

- **SPN (Substitution Permutation Network)**

- The input is divided into multiple sub-blocks
- Each gets a substitution and the output is mixed via a permutation
- Per-round subkeys are typically used to XOR the results

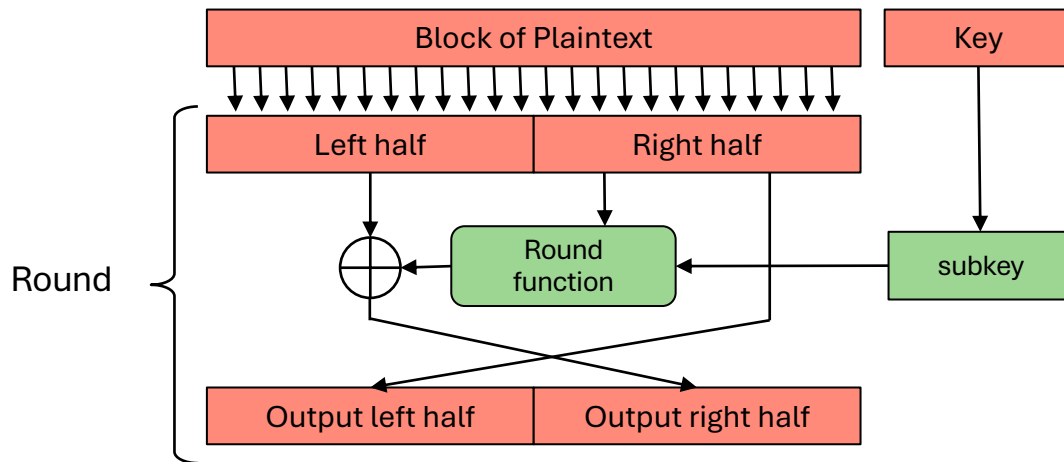
## Data Encryption Standard

- Developed in the early 1970s by IBM and modified by the NSA
- Adopted as a federal standard in 1976
- Block cipher, 64-bit blocks, 56-bit key
- Substitution followed by a permutation
  - Transposition and XORs based on a subkey derived from the key
  - 16 rounds

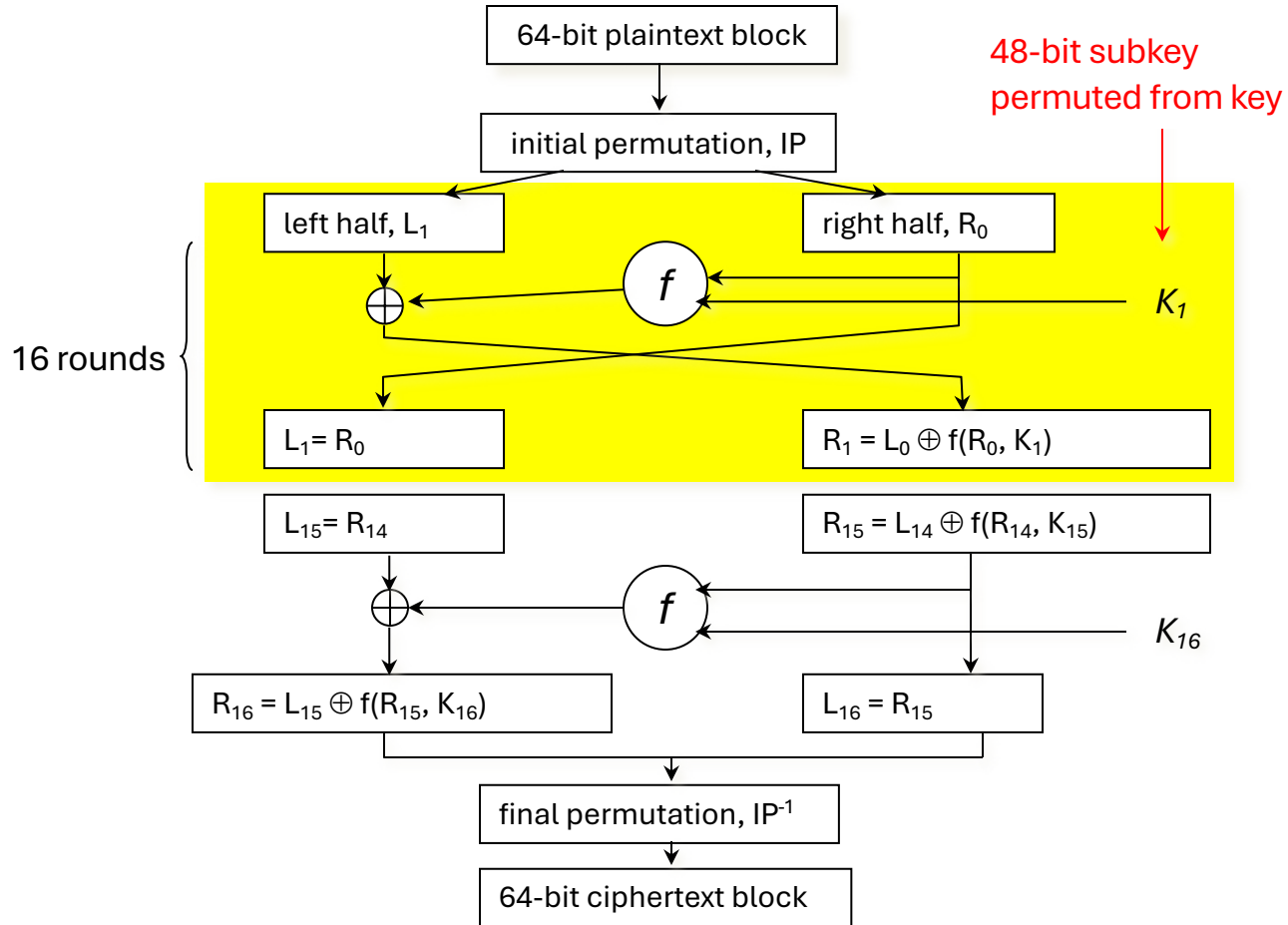


# Feistel cipher

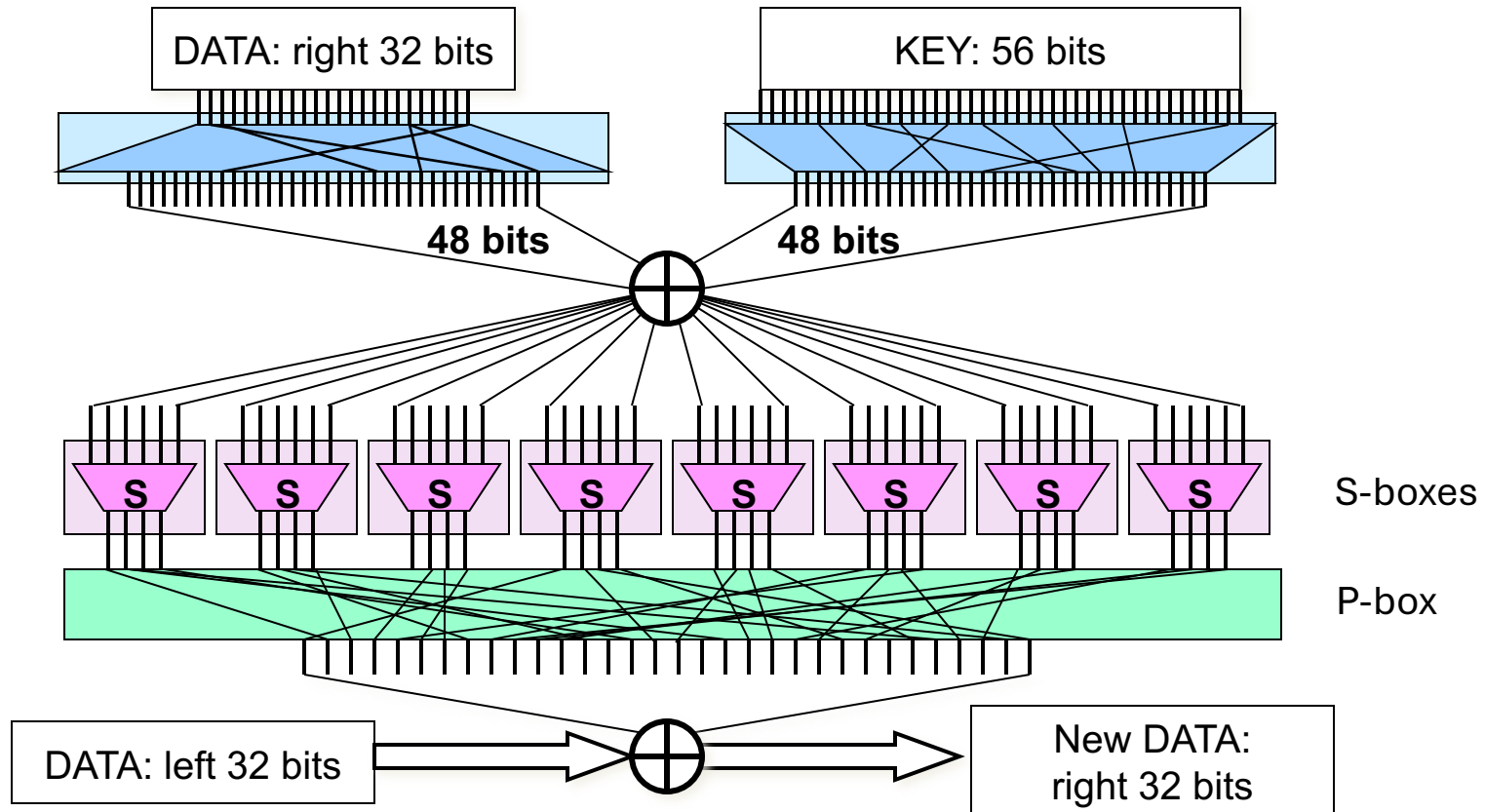
- DES is a type of **Feistel cipher**, which is a form of a **block cipher**
- **Plaintext block is split in two**
  - *Round* function applied to one half of the block
  - Output of the round function is XORed with the other half of the block
  - Halves are swapped
- This is a Feistel Network rather than an SP Network



# DES

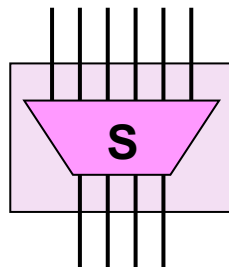


# DES: $f$ per round



# DES: S-boxes

- After compressed key is XORed with expanded block
  - 48-bit result moves to substitution operation via eight **substitution boxes** (s-boxes)



- Each S-box has
  - 6-bit input
  - 4-bit output
- 48 bits divided into eight 6-bit sub-blocks
- Each block is operated by a separate S-box
- Net result: 48-bit input generates 32-bit output
- **S-boxes are key components of DES's security**

S-boxes are used in symmetric block ciphers to add confusion: hide the relationship of any ciphertext from any plaintext & key bits.

Implemented as a table lookup

# Is DES secure?

- 56-bit key makes DES relatively weak
  - $2^{56} = 7.2 \times 10^{16}$  keys
  - Brute-force attacks possible
- By the late 1990's:
  - DES cracker machines built to crack DES keys in a few hours
  - DES Deep Crack: 90 billion keys/second
  - Distributed.net: tested 250 billion keys/second
- 2000s < 1 day
  - 2006: COPACOBANA: Custom FPGA-based DES cracker for < \$10,000
  - 2012: cloud-based service  
crack MS-CHAPv2 authentication (which uses DES) on sale for \$20

<https://boingboing.net/2012/09/24/exhaust-all-of-des-and-crack-a.html>

# Increasing The Key Size

Could double encryption work for DES?

Useless if we could find a key  $K$  such that:

$$E_K(P) = E_{K_2}(E_{K_1}(P))$$

This does not hold for DES (luckily!)

# Double DES

## Vulnerable to *meet-in-the-middle* attack

If we know some pair  $(P, C)$ , then:

- [1] Encrypt  $P$  for all  $2^{56}$  values of  $K_1$
- [2] Decrypt  $C$  for all  $2^{56}$  values of  $K_2$

For each match where  $[1] == [2]$

- Test the two keys against another  $P, C$  pair
- If match, you are assured that you have the key
- The complexity is  $2 \times 2^{56}$  rather than  $2^{2 \times 56}$

# Triple DES (3DES) – key lengths

Triple DES with two 56-bit keys (112-bit key):

$$C = E_{K1}(D_{K2}(E_{K1}(P)))$$

Triple DES with three 56-bit keys (168-bit key):

$$C = E_{K3}(D_{K2}(E_{K1}(P)))$$

Decryption used in middle step for compatibility with DES ( $K_1=K_2=K_3$ )

$$C = E_K(D_K(E_K(P))) \equiv C = E_{K1}(P)$$



# DES Disadvantages

- DES has been shown to have some weaknesses
  - Key can be recovered using  $2^{47}$  chosen plaintexts or  $2^{43}$  known plaintexts
  - *Note that this is not a practical amount of data to get for a real attack!*
- Short block size (8 bytes =  $2^8 = 64$  bits)
- The biggest weakness of DES is its 56-bit key
  - Exhaustive search requires  $2^{55}$  iterations on average
- 3DES solves the key size problem: we can have keys up to 168 bits
  - Differential & linear cryptanalysis is not effective here: the three layers of encryption use 48 rounds instead of 16 making it infeasible to reconstruct s-box activity
- But DES is relatively slow – and 3DES is 3x slower
  - It was designed with hardware encryption in mind: and 3DES is 3x slower than DES

# AES (Advanced Encryption Standard)

- U.S. NIST held a competition for a new algorithm
  - Received 15 submissions
  - No NSA tampering was allowed
  - Three variations (key lengths) of the Rijndael family of ciphers won
- Block cipher: 128-bit blocks
  - AES is an SP Network, not a Feistel cipher – it uses the entire block in each round
  - DES used 64-bit blocks but encrypted half the data in each round
- Successor to DES as a standard encryption algorithm
  - DES: 56-bit key
  - AES: 128, 192, or 256-bit keys

# AES ... successor to DES

From NIST:

*Assuming that one could build a machine that could recover a DES key in a second (i.e., try  $2^{56}$  keys per second), then it would take that machine approximately 149 trillion years to crack a 128-bit AES key. To put that into perspective, the universe is believed to be less than 20 billion years old.*

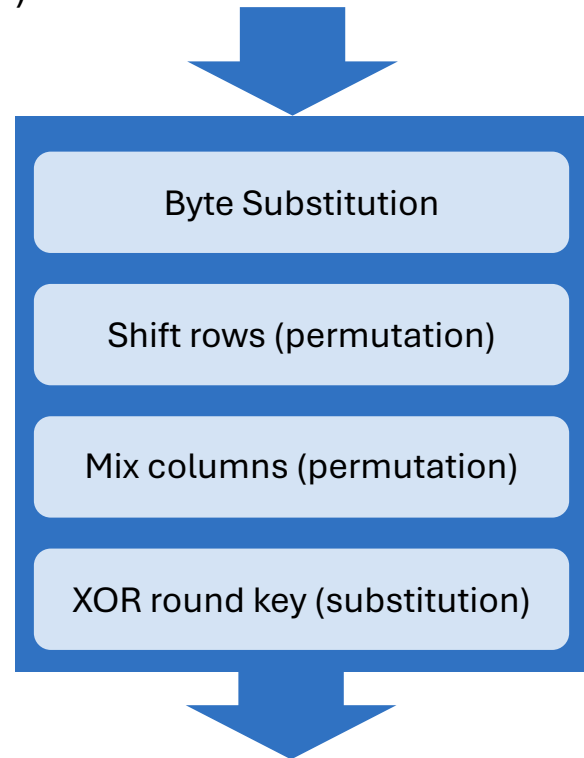
<https://www.nist.gov/news-events/news/2001/12/commerce-secretary-announces-new-standard-global-information-security>

# AES (Advanced Encryption Standard)

- Iterative cipher, just like most other block ciphers
  - Each round is a set of substitutions & permutations
- Variable number of rounds
  - DES always used 16 rounds
  - AES:
    - 10 rounds: 128-bit key
    - 12 rounds: 192-bit key
    - 14 rounds: 256-bit key
  - A **subkey** (“round key”) derived from the key is computed for each round
    - DES used this too, as do all multi-round ciphers

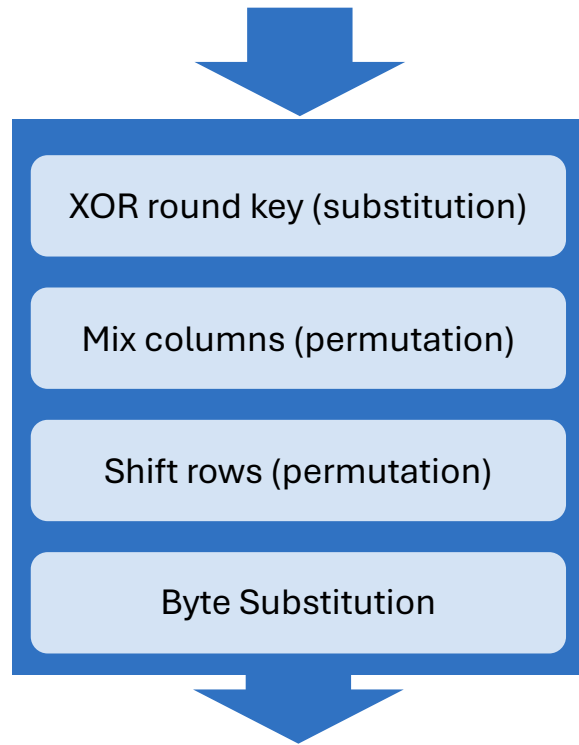
# Each AES Round

- **Step 1: Byte Substitution (s-boxes)**
  - Substitute 16 input bytes by looking each one up in a table (**s-box**)
  - Result is a **4x4 matrix**
- **Step 2: Shift rows**
  - Each row is shifted to the left (wrapping around to the right)
  - 1<sup>st</sup> row not shifted; 2<sup>nd</sup> row shifted 1 position to the left; 3<sup>rd</sup> row shifted 2 positions; 4<sup>th</sup> row shifted three positions
- **Step 3: Mix columns**
  - 4 bytes in each column are transformed
  - This creates a new 4x4 matrix
- **Step 4: XOR round key**
  - XOR the 128 bits of the round key with the 16 bytes of the matrix in step 3



# AES Decryption

Same rounds  
... but in reverse order



# AES Advantages

- **Larger block size:** 128 bits vs 64 bits
- **Larger & varying key sizes:** 128, 192, and 256 bits
  - 128 bits is complex enough to prevent brute-force searches
- **No significant academic attacks beyond brute force search**
  - Resistant against linear cryptanalysis thanks to bigger S-boxes
    - S-box = lookup table that adds non-linearity to a set of bits via transposition & flipping
  - DES: 6-bit inputs & 4-bit outputs
  - AES: 8-bit inputs & 8-bit outputs
- **Typically 5-10x faster in software than 3DES**

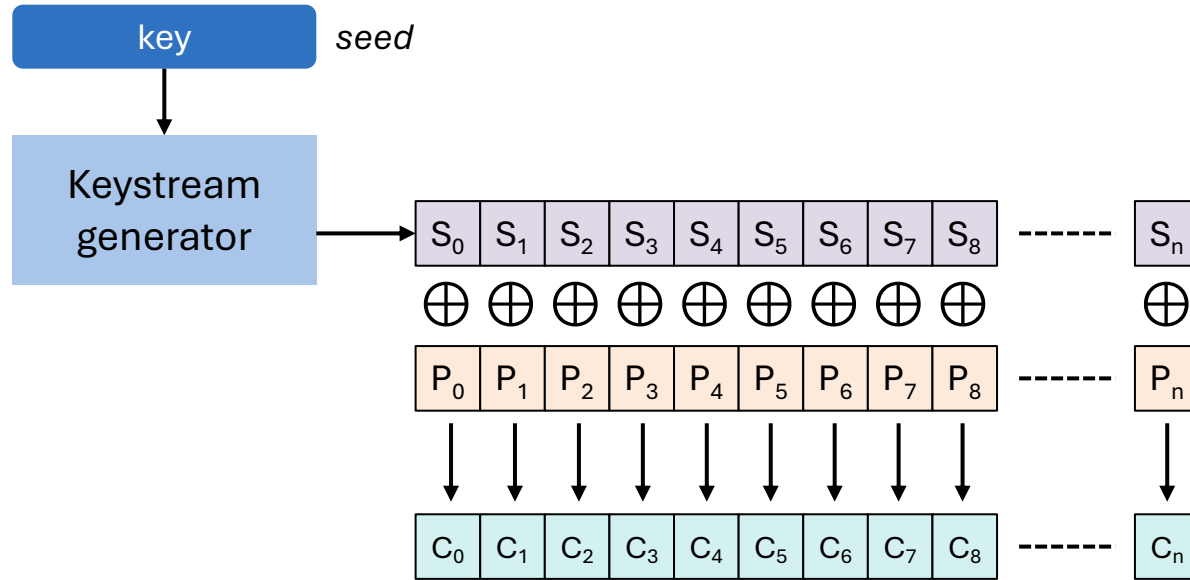
# Attacks against AES

- Attacks have been found
  - This does **not** mean that AES is insecure!
- Because of these weaknesses:
  - AES-128 has a computational complexity of  $2^{126.1}$  (~126 bits)
  - AES-192 has a computational complexity of  $2^{189.7}$  (~190 bits)
  - AES-256 has a computational complexity of  $2^{254.9}$  (~255 bits)
- Increasing AES security
  - The security of AES can be increased by increasing the number of rounds in the algorithm
  - However, AES-128 still has a sufficient safety margin to make exhaustive search attacks impractical



# Stream ciphers – simulate a one-time pad

**Keystream generator** produces a sequence of **pseudo-random** bytes



$$C_i = S_i \oplus P_i$$

# Stream ciphers

Just like with the one-time pad, you can never reuse a key

$$C = A \oplus K$$

$$C' = B \oplus K$$

$$C \oplus C' = A \oplus K \oplus B \oplus K = A \oplus B$$

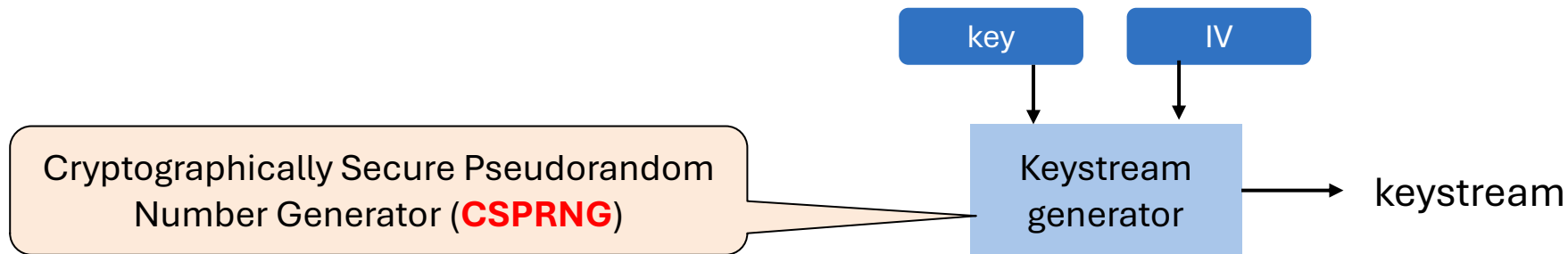
Guess  $A$  to get  $K$  and see if  $B$  makes sense

Or... if you have **known plaintext**  $A$  and the corresponding ciphertext  $C$ , you can extract the key:

$$K = A \oplus C$$

# So how can you use the same key?

- A stream cipher seeds the keystream generator with:
  - The key
  - A unique random value, called a **nonce** (number used once) or **initialization vector (IV)**
- The initialization vector is not secret but is unique to each message
  - It can be added as a header to the ciphertext
- This ensures the keystream generated will be different even if the same key is used multiple times



# Popular symmetric ciphers

AES (Advanced Encryption Standard)	<ul style="list-style-type: none"><li>• FIPS standard since 2002</li><li>• 128, 192, or 256-bit keys; operates on 128-bit blocks</li><li>• By far the most widely used symmetric encryption algorithm</li></ul>
DES (Data Encryption Standard)	<ul style="list-style-type: none"><li>• FIPS standard from 1976-2002</li><li>• 56-bit key; operates on 64-bit (8-byte) blocks</li><li>• Triple DES recommended since 1999 (112 or 168 bits)</li><li>• Not actively used anymore; AES is better by any measure</li></ul>
Blowfish	<ul style="list-style-type: none"><li>• Key length from 23-448 bits; 64-bit blocks</li><li>• Optimized for 32-bit CPUs</li></ul>
Twofish	<ul style="list-style-type: none"><li>• Successor to Blowfish; key length from 128, 192, 256 bits; 128-bit blocks</li><li>• Competed against AES for standardization</li></ul>
ChaCha20	<ul style="list-style-type: none"><li>• Stream cipher</li><li>• 256-bit key generated from a user-supplied key</li><li>• One of the fastest encryption algorithms</li></ul>

# Around the world

AES	<ul style="list-style-type: none"><li>• U.S., Canada, EU, India, UK</li></ul>
ARIA	<ul style="list-style-type: none"><li>• South Korea</li><li>• SP network structure based on AES</li><li>• 128, 192, or 256 bits – 12, 14, 16 rounds</li></ul>
SM4	<ul style="list-style-type: none"><li>• China (variation of a Feistel network)</li><li>• 128-bit key, 128-bit block – 32 rounds</li></ul>
ZUC	<ul style="list-style-type: none"><li>• China: 128-bit stream cipher, used in mobile communications)</li><li>• 128-bit initialization vector, 128-bit key size feeds into a linear feedback shift registers</li></ul>
GOST R 34.12-2015	<ul style="list-style-type: none"><li>• Russia – family of two block ciphers: Feistel network</li><li>• 128-bit key – 64-bit blocks – 32 rounds / 256-bit key, 128-bit blocks – 10 rounds</li><li>• 128-bit (Magma): Feistel network; 256-bit: SPN (similar to AES)</li></ul>
Camellia	<ul style="list-style-type: none"><li>• Japan (AES is also popular)</li><li>• Feistel network: 18 rounds (128-bit keys) or 24 rounds (192, 256-bit)</li></ul>

# The End