

CPEN 442 – Introduction to Cybersecurity

Module 2



Cryptography

CPEN 442 – Introduction to Cybersecurity

Module 2 – Cryptography

Part 1:
Preamble

Part 2:
Ancient Crypto

Part 3:
Symmetric
Crypto

Part 4:
Public-key
Crypto

Part 5:
Integrity and
Authentication

CPEN 442 – Introduction to Cybersecurity

Module 2 – Cryptography



Part 1 – Preamble

- Definitions
 - Cryptology
 - Goals
 - Confusion and Diffusion
 - Symmetric vs Asymmetric
- Attacker
 - Kerckhoff's principle
 - Brute-force attacks
 - Cryptography attack models

Part 2:
Ancient Crypto

Part 3:
Symmetric
Crypto

Part 4:
Public-key
Crypto

Part 5:
Integrity and
Authentication

What is cryptography?

- **Cryptology** is a science that studies:
 - Cryptography (“**secret writing**”): making secret messages
 - Turning a **plaintext** (an ordinary readable message) into a **ciphertext** (a secret message that is “hard” to read)
 - Cryptanalysis: breaking secret messages
 - Recovering the **plaintext** from the ciphertext
- The point of cryptography is to send secure messages over an insecure medium (like the Internet).
- Cryptanalysis studies cryptographic systems to look for weaknesses or leaks of information.

What are the goals of cryptography?

Cryptography aims to achieve these properties:

One second! Do you remember what is confidentiality, integrity, and authenticity?

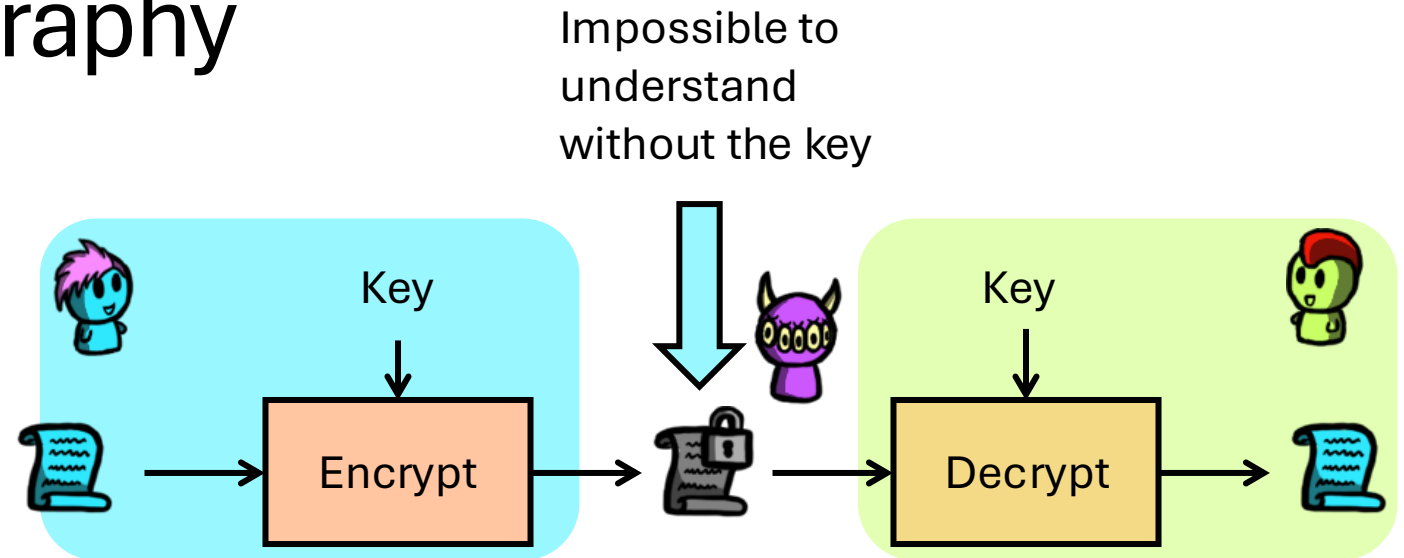


- **Confidentiality**: prevent Eve from *reading* Alice's messages.
- **Integrity**: prevent Mallory from *modifying* Alice's messages without being detected.
- **Authenticity**: preventing Mallory from *impersonating* Alice.
- To get **availability**, we need other techniques (redundancy, etc.).

The goals of cryptography

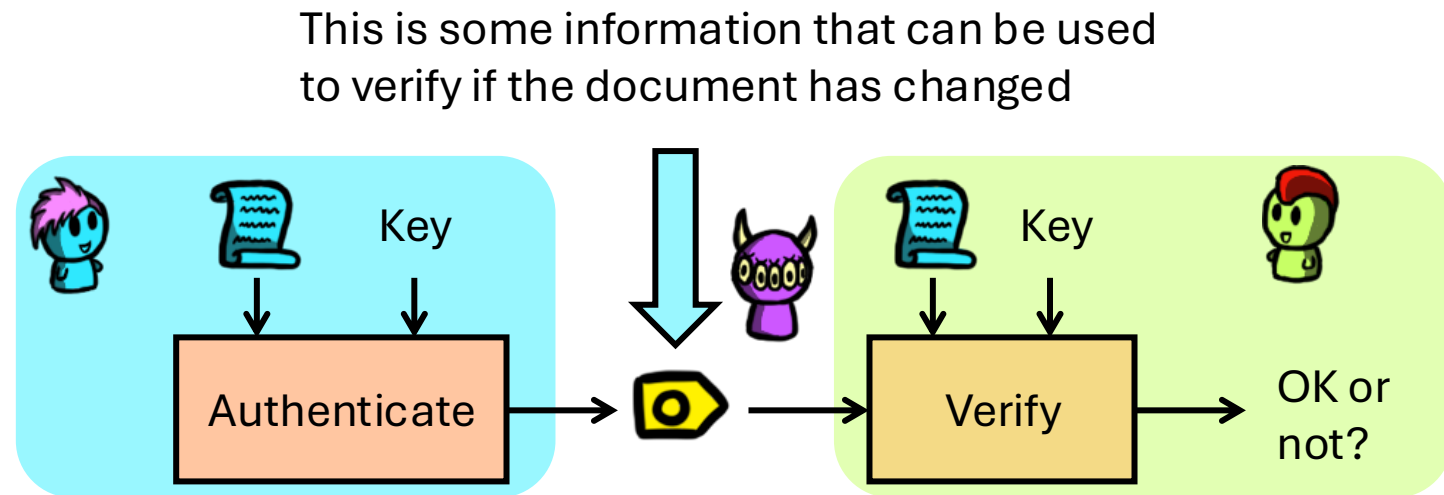
Confidentiality, visually:

- We will see the keys can be the same (symmetric) or different (asymmetric).



Integrity and authentication, visually:

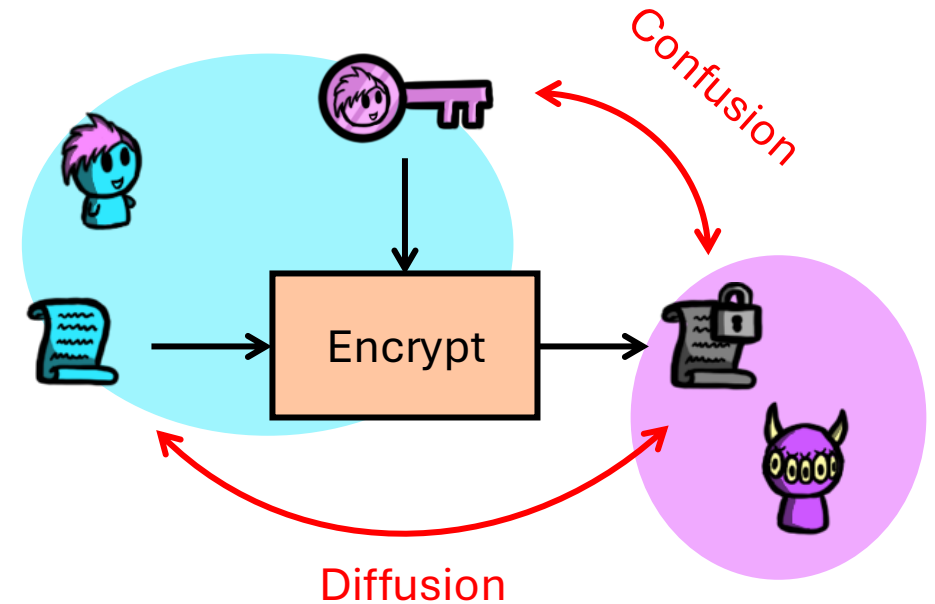
- Assume the message is transmitted through another (possibly insecure) channel.
- This could be computed from the encrypted message instead.



Confusion and diffusion

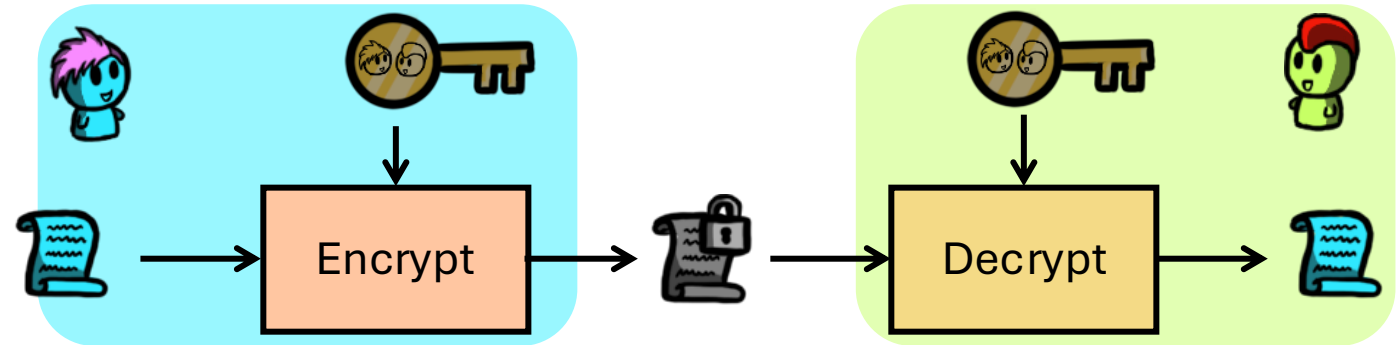
Two important properties of secure cryptosystems:

- **Confusion:** the relationship between the **secret key** and **ciphertext** should be as obscure as possible
- **Diffusion:** the relationship between the **plaintext** and **ciphertext** should be as obscure as possible
 - Ideally, changing one single bit of the plaintext should “re-randomize” the ciphertext (i.e., on average, change half of the bits in the ciphertext)



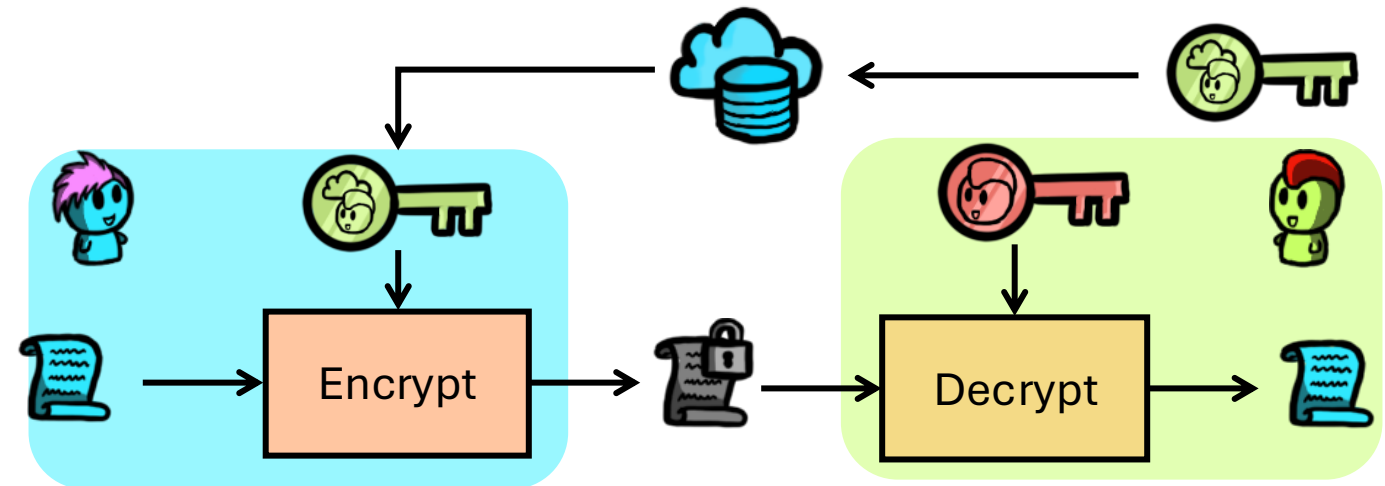
Symmetric Crypto

- We use the **same secret key** to encrypt and to decrypt
- Also called **secret-key** crypto



Asymmetric Crypto

- We use one key to encrypt (usually, the **public key**), and another one to decrypt (the **private key**).
- Usually called **public-key** crypto





About key sizes and message sizes.

What do you think are larger in size (e.g., bits):
keys or messages?



If my key is 128 bits, how many possible keys are
there?



If my message is one page (around 1500
characters) in ASCII (128 possible characters).
How many possible messages are there?



How many of these messages “make sense”?



Almost always, messages are larger.



$$2^{128} \approx 3.4 \cdot 10^{38}$$



$$128^{1500} \approx 6.5 \cdot 10^{3160}$$



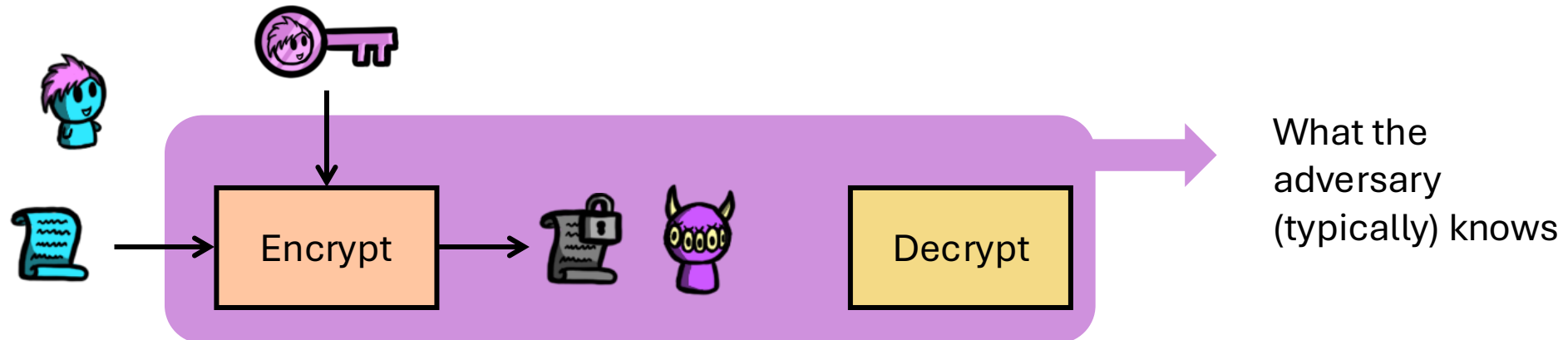
Who knows... but usually just a tiny TINY fraction of
that number.



We will come back to this in two slides...

Kerckhoff's principle

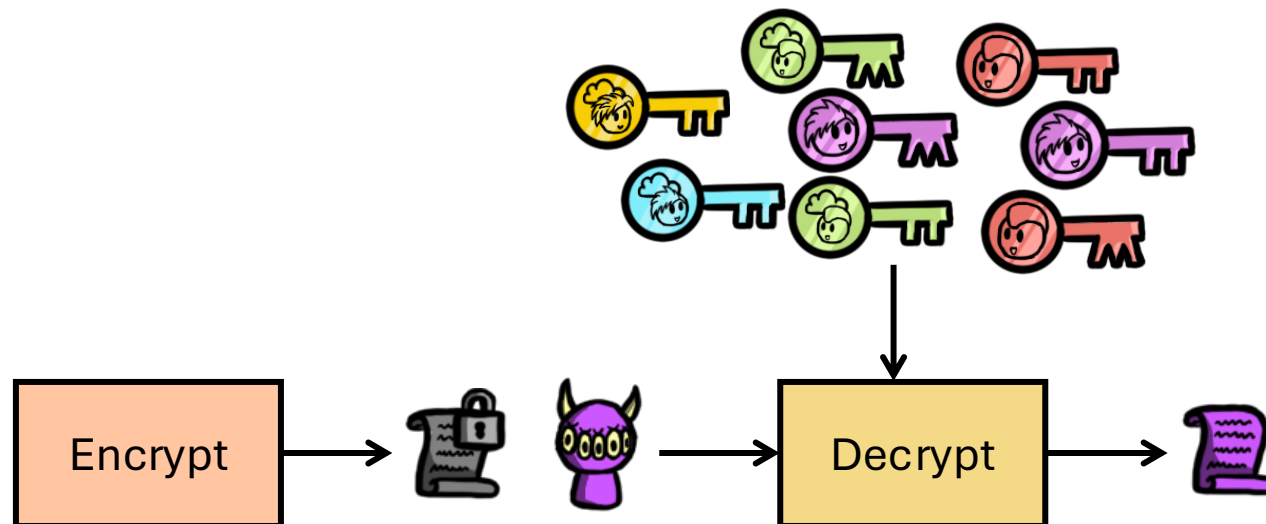
- **Kerckhoff's principle:** a cryptosystem should be secure even if everything about the system, except the **key**, is public knowledge.
- **Shannon's maxim:** one ought to design systems under the assumption that the enemy will immediately gain full familiarity with them.
- Do not use “secret” encryption methods (security by obscurity).
- Have **public algorithms** that use a **secret key** instead.
- If the adversary learns the secret: it's easy to change a key (small-ish number); it's not feasible to design a brand-new system.



Kerckhoff's principle

Kerckhoff's principle has a number of implications:

- The system is at most as secure as the total number of possible keys.
- Eve can **try every possible key**, until she finds the right one.
- Many times, there are **shortcuts** to finding the key
 - We will see some examples later, in the newspaper cryptogram



Remember from two slides ago: there are very few messages that “make sense” from the pool of “all possible messages”. It is very likely that one single key leads to a decryption that makes sense!

Does it
make
sense?

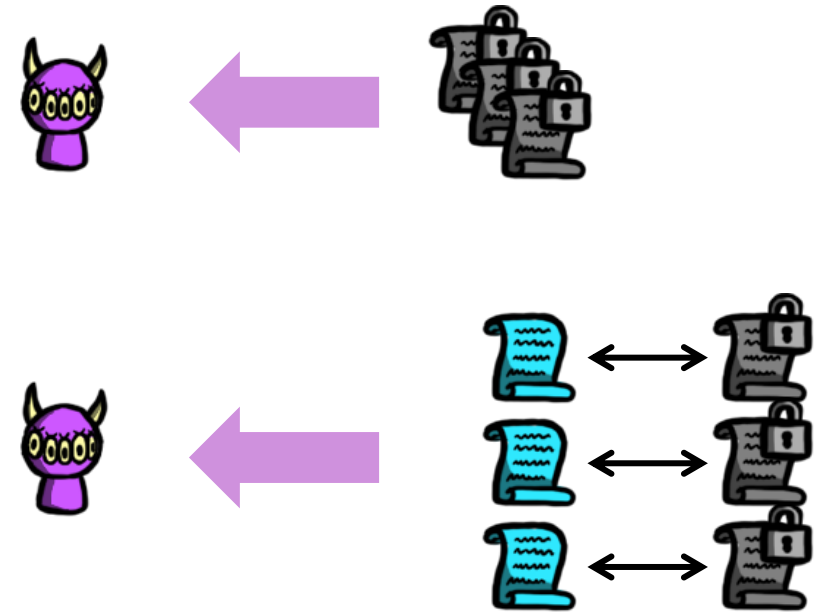


Cryptography Attack Models

For each of these, let's think of real-world scenarios where they are reasonable threat models!

When talking about cryptography, we usually assume that Eve knows the cryptography algorithm (Kerckhoff's principle). What else does she know?

- **Ciphertext-only attack:** Eve has at least one encrypted message and tries to break it (guess the key and/or plaintext).
 - Brute forcing (trying every key) is one way of carrying a ciphertext-only attack.
- **Know plaintext attack:** Eve knows one (or many) plaintext and ciphertext pairs. She tries to guess the key.



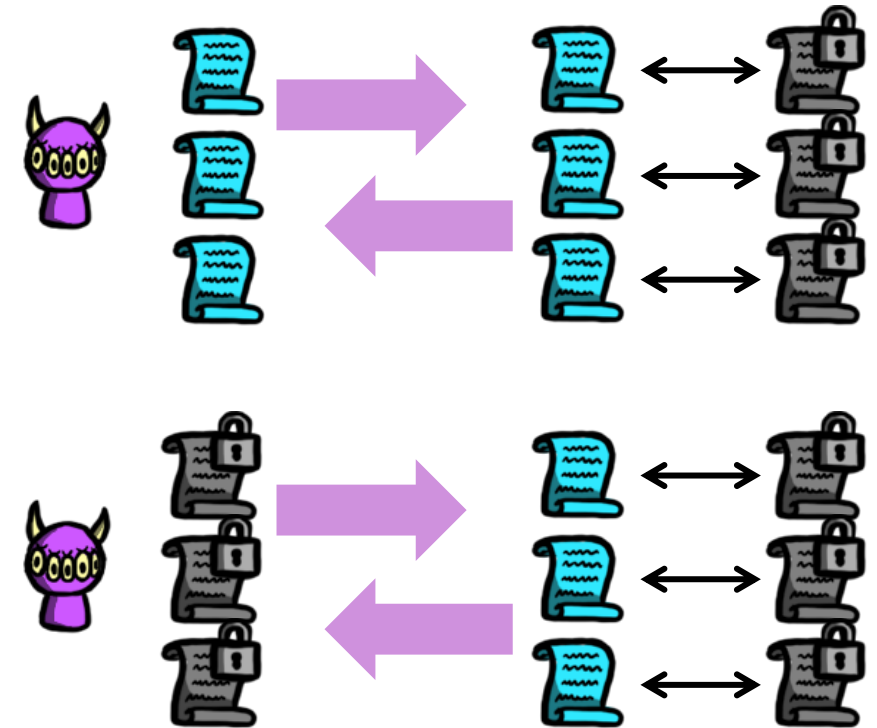


Cryptography Attack Models

For each of these, let's think of real-world scenarios where they are reasonable threat models!

When talking about cryptography, we usually assume that Eve knows the cryptography algorithm (Kerckhoff's principle). What else does she know?

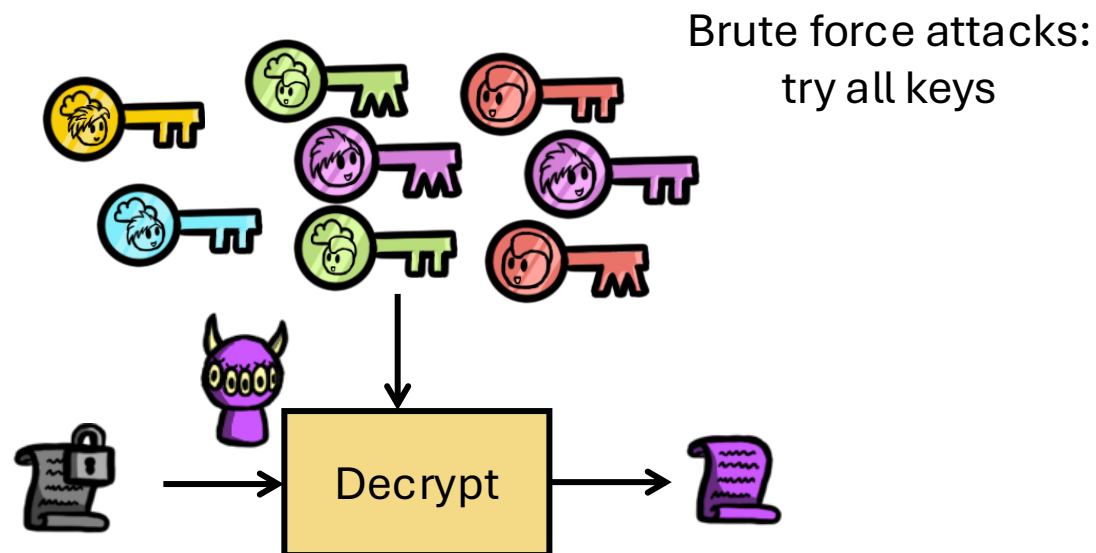
- **Chosen plaintext attack:** Eve can choose any arbitrary plaintext and get the ciphertext (possibly many times). She wants to guess the key.
 - Can be *adaptive*: Eve choose plaintexts after seeing the previous encryptions
- **Chosen ciphertext attack:** Eve can choose any arbitrary ciphertext and get the plaintext (possibly many times). She wants to guess the key.
 - Can also be *adaptive*.



Note: in cryptographic proofs, it is usually assumed that an adversary that can do CCA can also do CPA

Brute-force Attacks

- Ideally, we want a cryptosystem (i.e., a key generation algorithm, plus encrypt and decrypt functions) to defend in all these four attack models.
- However: there is always an attack that works: **brute-forcing**.
- This is why cryptosystems only provide **computational security** (i.e., security against a computationally-bounded adversary).
 - There is one exception, that we will see later



Does this mean crypto is useless? Why?



CPEN 442 – Introduction to Cybersecurity

Module 2 – Cryptography



Part 2 – Ancient Crypto

Part 1:
Preamble

- Caesar cipher
- Vigenère cipher
- Substitution ciphers
- Playfair cipher
- Enigma machine
- Vernam cipher
- One-time pad

Part 3:
Symmetric
Crypto

Part 4:
Public-key
Crypto

Part 5:
Integrity and
Authentication

Ancient Cryptography

Cryptography is very old; it existed before the digital era.

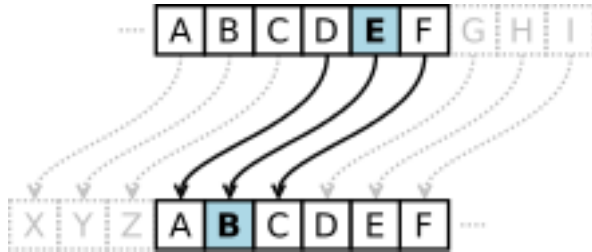
We are going to see some classic examples of cryptosystems:

- Caesar cipher
- Vigenère cipher
- Substitution ciphers
- Playfair cipher
- Enigma machine
- Vernam cipher
- One-time pad

These are all broken, never use them in the real world! (The one-time pad is not broken, but it's not useful.)

The Caesar Cipher

- The first well-known cipher, used by Julius Caesar (100 BC) to protect military messages.
- Very simple: it is a *substitution cipher* that replaces each letter with the one that is a fixed number of positions down the alphabet (e.g., Julius Caesar used a shift of 3):



source: Wikipedia

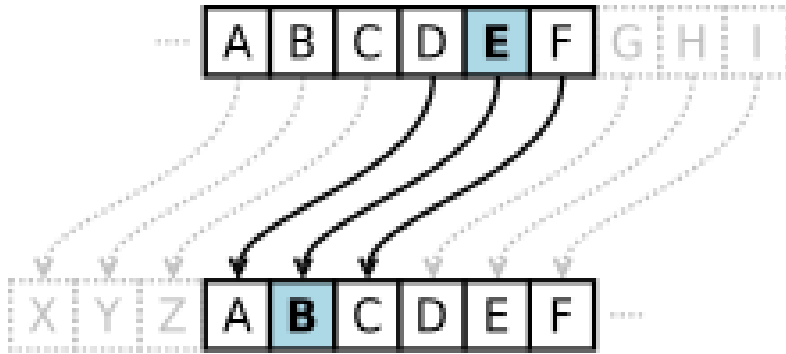
Plaintext: ATTACKTONIGHT

Key: -3

Ciphertext: XQQXZHQ LKFDEQ



How would you break a Caesar cipher?



source: Wikipedia

Plaintext: ATTACKTONIGHT

Key: -3

Ciphertext: XQQXZHQLKFDEQ

The Caesar cipher was broken 800 years later: frequency analysis!

- English text has a particular frequency for each letter, a “fingerprint”.
- The Caesar cipher just does a circular shift of this fingerprint, so it’s very easy to undo it.



Why did people think this was secure?



Security by obscurity

Vigenère cipher

- For a long time, attributed to Blaise de Vigenère, who published it in 1586.
 - Actually, invented by Giovanni Battista Bellaso in 1553.
- The Vigenère cipher uses a different circular shift for each letter, determined by the secret key.

- Example:

Plaintext: ATTACKTONIGHT

Key: MYKEYMYKEYMYK

Ciphertext: MRDEAWRYRGSGFD

1. Repeat the key to match plaintext size
2. Convert letters to numbers, do modular addition, and convert back to letters:
 - $A(0) + M(12) = M(12)$
 - $T(19) + Y(24) = 43 \rightarrow R(17)$
 - ...



How would you break a Vigenère cipher?

Plaintext: ATTACKTONIGHT
Key: MYKEYMYKEYMYK
Ciphertext: MRDEAWRYRGSGFD

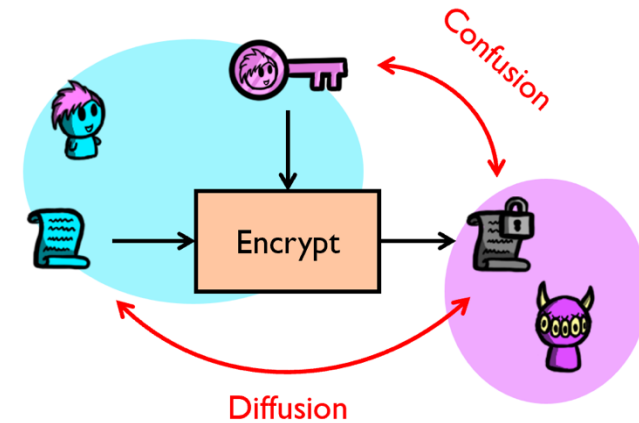
You will learn this in A1



How is the confusion and diffusion of this cipher?



Pretty bad...



Substitution ciphers

- The Caesar and Vigenère ciphers are examples of substitution ciphers.
- A substitution cipher **replaces** a unit of the plaintext (usually one letter) for a unit of the ciphertext, with the help of a key.
 - We could have a substitution using pairs of letters, triplets, etc.
- The *Caesar* cipher uses a single substitution for the entire plaintext: **monoalphabetic cipher**.
- The *Vigenère* cipher uses different substitutions (depending on the key): **polyalphabetic cipher**.

Plaintext: ATTACKTONIGHT

Key: -3



Ciphertext: XQQXZHQLKFDEQ

Caesar: always the same substitutions

Plaintext: ATTACKTONIGHT

Key:



Ciphertext: MRDEAWRYRG SFD

Vigenère: 5 different types of substitutions (with this key)

Substitution ciphers

- Monoalphabetic substitution ciphers can be more complex than the Caesar cipher.
- The most complex monoalphabetic cipher: pick a random permutation of the alphabet as substitution

Plaintext: ABCDEFGHIJKLMNOPQRSTUVWXYZ
 Key: VMRKHXUTEPLDNYBCFZOJAIWSQ
 Ciphertext: VMRKHXUTEPLDNYBCFZOJAIWSQ

In this example, to encrypt, we substitute the i th letter in the alphabet with the i th letter in the key.

How many possible keys are there in an alphabet of 26 letters?



$26! = 403,291,461,126,605,635,584,000,000$



Do you have to try every possible key to break something like this?



It is way easier than that! (if the plaintext is not random letters, but has some language structure, of course)





Daily cryptogram

- A daily cryptogram is a (monoalphabetic) substitution cipher.
- We will see why we don't need to try every single key to break this...



We know this is a definition written in English. How would you crack the plaintext?

wordplaysTMcom

Crossword Solver	Scrabble Word Finder	Boggle	Text Twist	Sudoku	Anagram Solver	Word Games	
Wordle	Scrabble Help	Words with Friends Cheat	Words in Words	Word Jumbles	Word Search	Scrabble Cheat	Cryptogram

DAILY CRYPTOGRAM [Daily Cryptogram Help ?](#)

Puzzle #1267 - CATEGORY: DEFINITIONS Puzzle # Find

T V J

M G Q P E S M P U

,

G . :

Q F P

P W R E A R M Z Q M G I

C E V R P Y Y

B A E M G I

U F M R F

C P E Y V G G P D

V K K M R P E Y

Y P C Z E Z Q P

Q F P

U F P Z Q

K E V O

Q F P

R F Z K K

- -

Q F P G

F M E P

Q F P

R F Z K K .



Daily cryptogram

wordplays™com

Crossword Solver | Scrabble Word Finder | Boggle | Text Twist | Sudoku | Anagram Solver | Word Games

Wordle | Scrabble Help | Words with Friends Cheat | Words in Words | Word Jumbles | Word Search | Scrabble Cheat | Cryptogram

DAILY CRYPTOGRAM

[Daily Cryptogram Help ?](#)

Puzzle #1267 - CATEGORY: DEFINITIONS

Puzzle # [Find](#)

J O B I N T E R V I E W , N . : T H E
T V J M G Q P E S M P U , G . : Q F P

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

W H I C H P E R S O N N E L O F F I C E R S
U F M R F C P E Y V G G P D V K K M R P E Y

S E P A R A T E T H E W H E A T F R O M T H E C H A F F
Y P C Z E Z Q P Q F P U F P Z Q K E V O Q F P R F Z K K

- - T H E N H I R E T H E C H A F F .
- - Q F P G F M E P Q F P R F Z K K .

[Get a Hint](#) [Solve the Puzzle](#) [New Puzzle](#) [Clear](#)

Breaking weak ciphers

- How would you automate the attack, if you had a very long ciphertext?
- In any language, the frequencies of each letter are not uniform. This means that, with **frequency analysis**, we can break substitution ciphers easily.

What if we made substitutions of pairs of characters?



We can compute the frequencies of pairs of characters, for example



Letter ↕	Relative frequency in the English language ^[1]			
	Texts ▼		Dictionaries ↕	
E	12.7%		11.0%	
T	9.1%		6.7%	
A	8.2%		7.8%	
O	7.5%		6.1%	
I	7.0%		8.6%	
N	6.7%		7.2%	
S	6.3%		8.7%	
H	6.1%		2.3%	
R	6.0%		7.3%	
D	4.3%		3.8%	
L	4.0%		5.3%	
C	2.8%		4.0%	
U	2.8%		3.3%	
M	2.4%		2.7%	
W	2.4%		0.91%	
F	2.2%		1.4%	
G	2.0%		3.0%	
Y	2.0%		1.6%	
P	1.9%		2.8%	
B	1.5%		2.0%	
V	0.98%		1.0%	
K	0.77%		0.97%	
J	0.15%		0.21%	
X	0.15%		0.27%	
Q	0.095%		0.19%	
Z	0.074%		0.44%	

Wikipedia: “letter frequency”

Index of Coincidence

- The Index of Coincidence (IC) is a metric that measures how likely it is that, if you grabbed two random letters from a text, both letters were the same letter.

$$IC = c \left(\frac{n_A}{N} \times \frac{n_A - 1}{N - 1} + \frac{n_B}{N} \times \frac{n_B - 1}{N - 1} + \dots \right)$$

$c = 26$ for English (the number of letters in the alphabet)

- Different languages have different ICs (e.g., English is 1.73, French is 2.02...)
- It can be used as a metric of how English-like a text is. Can be used as a tool when cracking a cipher.
- E.g., it could be used to detect the length of the key of a Vigenère cipher...
- There are more sophisticated **fitness tests** (e.g., using digram, trigram, or quadgram frequencies).

Letter ↕	Relative frequency in the English language ^[1]			
	Texts ▼		Dictionaries ↕	
E	12.7%		11.0%	
T	9.1%		6.7%	
A	8.2%		7.8%	
O	7.5%		6.1%	
I	7.0%		8.6%	
N	6.7%		7.2%	
S	6.3%		8.7%	
H	6.1%		2.3%	
R	6.0%		7.3%	
D	4.3%		3.8%	
L	4.0%		5.3%	
C	2.8%		4.0%	
U	2.8%		3.3%	
M	2.4%		2.7%	
W	2.4%		0.91%	
F	2.2%		1.4%	
G	2.0%		3.0%	
Y	2.0%		1.6%	
P	1.9%		2.8%	
B	1.5%		2.0%	
V	0.98%		1.0%	
K	0.77%		0.97%	
J	0.15%		0.21%	
X	0.15%		0.27%	
Q	0.095%		0.19%	
Z	0.074%		0.44%	

Wikipedia: “letter frequency”

Affine ciphers

- The affine cipher is an example of a monoalphabetic substitution cipher.
- To encrypt:
 1. Map each plaintext letter to a number (e.g., $A \rightarrow 0$, $B \rightarrow 1$, $C \rightarrow 2$, ..., $Z \rightarrow 25$)
 2. For each letter x , compute $y = (xa + b) \bmod 26$.
 3. Then, convert the y 's back to letters.

What is the key?
How do we decrypt?



Does this always
work?



The key is (a, b)

We decrypt by doing $x = a^{-1}(y - b) \bmod 26$



There needs to be a modular inverse for a to decrypt.
This is only true when a and 26 are coprime (their only common divisor is 1). We will see this later.





How would you break an affine cipher?

Affine Cipher

1. Map each plaintext letter to a number (e.g., $A \rightarrow 0$, $B \rightarrow 1$, $C \rightarrow 2$, ..., $Z \rightarrow 25$)
2. For each letter x , compute $y = (xa + b) \bmod 26$.
3. Then, convert the y 's back to letters.



Frequency analysis,
trying all keys, etc.

The Playfair cipher

- Invented by Wheatstone (in 1854) for the telegraph, but has the name of Lord Playfair, who made it popular.
- Used later for military purposes (e.g., World War 1, although it was already deemed as insecure back then).
- It is a **digram substitution cipher** (replaces two letters for two other letters).

How it works:

- The **alphabet space** is 25 characters (e.g., we assume J and I are the same letter, represented as I).
- The **key** is a permutation of those 25 characters (e.g., random permutation, or a password plus padding).
- Process plaintext to avoid identical consecutive letters (e.g., add an X in between every two repeated letters).
- Separate letters into **pairs**
- Apply the substitution following Playfair's rules.

P	L	A	Y	F	A
I	R	E	X	M	P L 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

"hide the gold in the tree stump" →

"HI DE TH EG OL DI NT HE TR EX ES TU MP"

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

HI

Shape: Rectangle
Rule: Pick Same Rows,
Opposite Corners

BM

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

DE

Shape: Column
Rule: Pick Items Below Each
Letter, Wrap to Top if Needed

OD

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

EX

Shape: Row
Rule: Pick Items to Right of Each
Letter, Wrap to Left if Needed

XM

Ciphertext = "BM OD ZB XD NA BE KU DM UI XM MO UV IF"



How would you break a Playfair cipher?



It requires more effort than anything we have seen. It can be done with hill-climbing heuristics. The idea is to maximize how “reasonable” the plaintext looks while we keep making small changes to the key.

P L A Y F^A
I R E X^A M^P L 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

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

HI

Shape: Rectangle
Rule: Pick Same Rows,
Opposite Corners

BM

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

DE

Shape: Column
Rule: Pick Items Below Each
Letter, Wrap to Top if Needed

OD

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

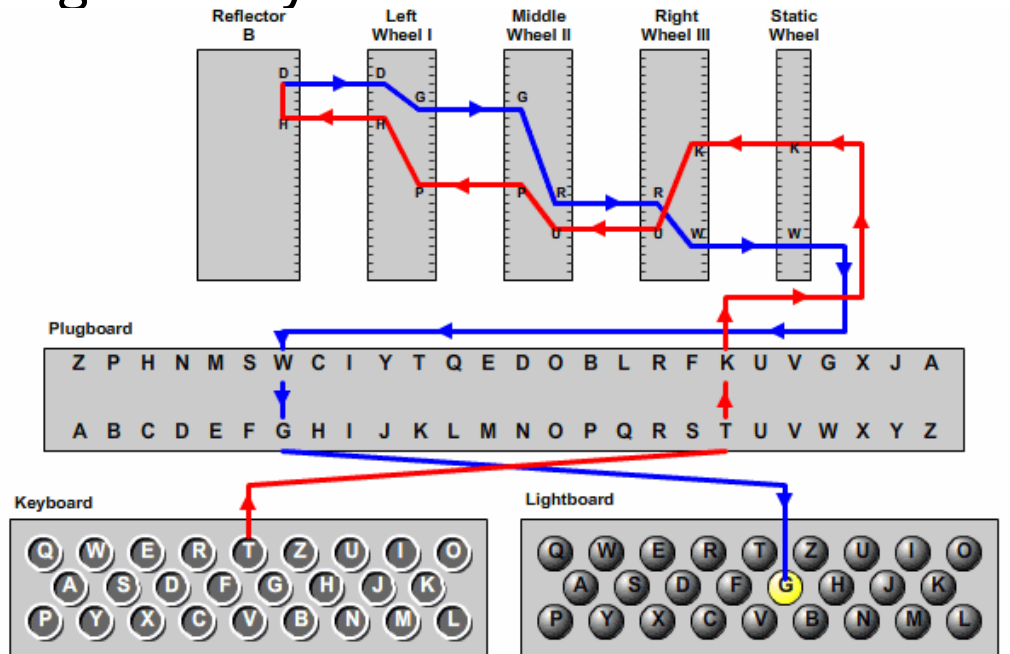
EX

Shape: Row
Rule: Pick Items to Right of Each
Letter, Wrap to Left if Needed

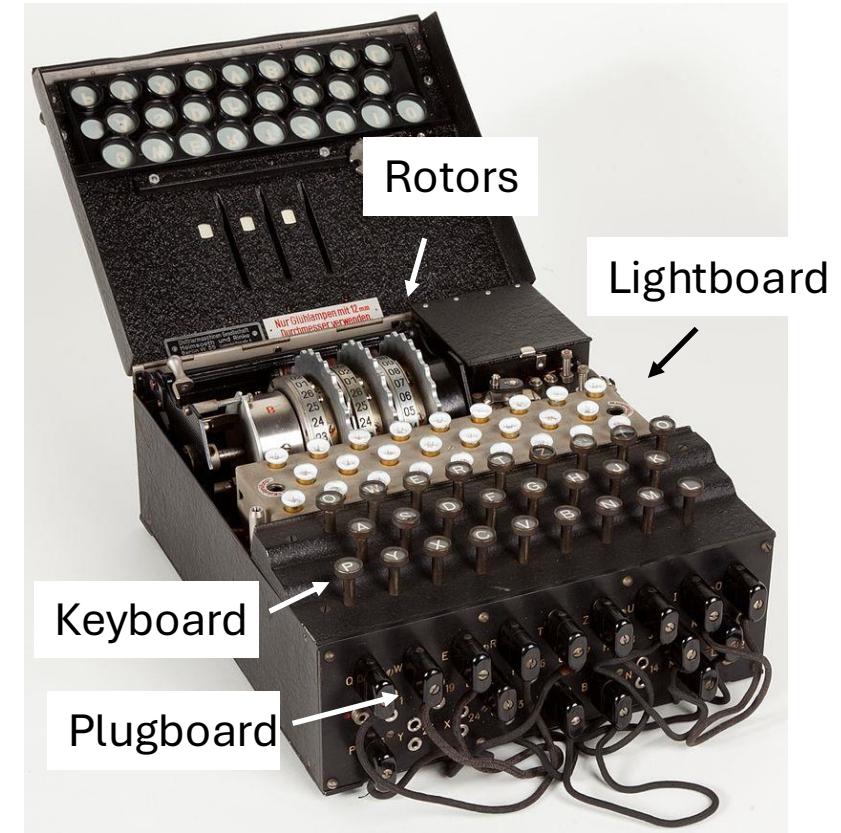
XM

The Enigma machine

- Used by the Nazi Germany during World War II.
- Implemented a substitution cipher, but the mapping will change with rotating rotors after pressing one key.

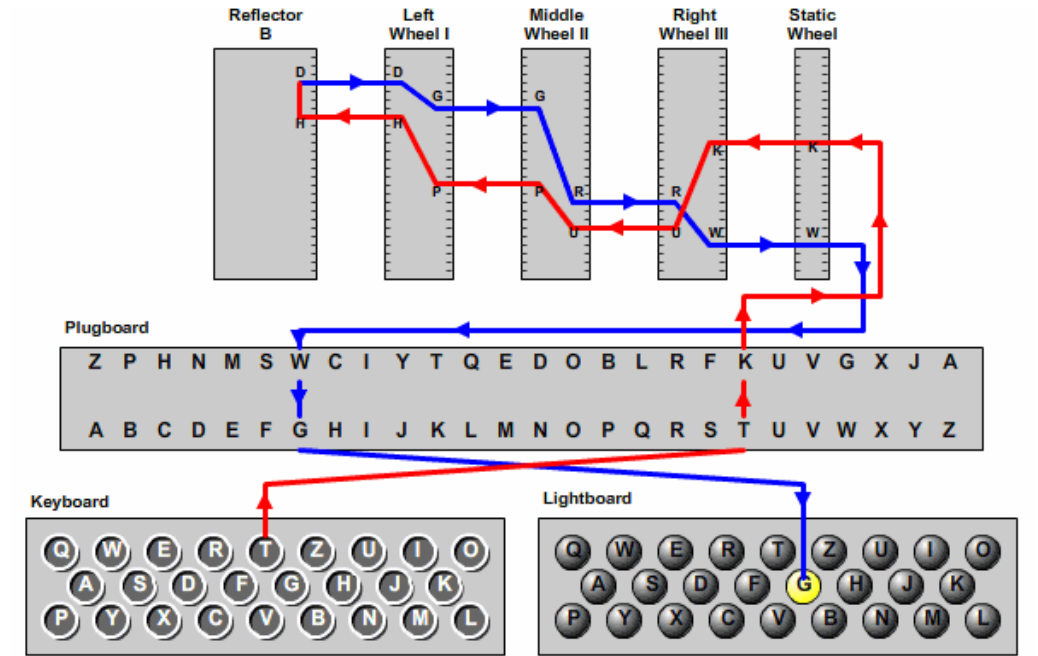


© 2006, by Louise Dade



The Enigma machine

- The “secret key” is the initial configuration of the machine.
- There were 5 possible rotors, from which you select three in a particular order.
- Each ring has 26 possible initial positions.
- The plugboard had $150 \cdot 10^{12}$ combinations.
- The total number of possible keys was around $159 \cdot 10^{18}$



© 2006, by Louise Dade

How many
rotor
permutations?

$5 \cdot 4 \cdot 3 = 60$ possible
rotor permutations

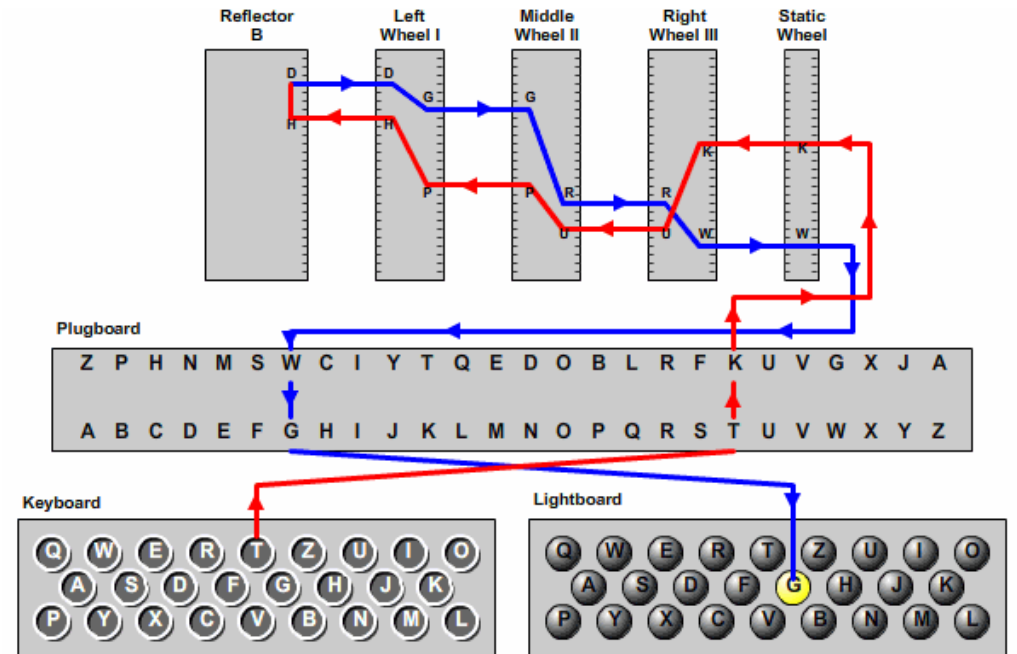
How many initial
rotor settings?

$26 \cdot 26 \cdot 26 = 17,576$ possible
initial rotor settings



How would you break the Enigma machine?

- **Problem:** a letter could never match itself!! (an encryption of a letter could never be that same letter)
- E.g., ciphertext: WJELDUYBNHJXP
 WETTER
 WETTER
 WETTER
- If you know the message has the word weather in German (WETTER)...
- This “small” leakage allows eventually breaking the cipher.
- Alan Turing built the Bombe Machine to crack Enigma Code in 20 minutes.



© 2006, by Louise Dade



What kind of attack model is this?

- Ciphertext-only attack
- Known plaintext attack
- Chosen plaintext attack
- Chosen ciphertext attack



Known
plaintext
attack

Vernam Cipher

- Named after Gilbert Stanford Vernam, who invented it in 1917.
- Encrypt plaintexts bit-by-bit, by XOR-ing with the key
- Plaintext (t bits): $M = [m_1, m_2, \dots, m_t]$
- Key (t bits): $K = [k_1, k_2, \dots, k_t]$
- Ciphertext (t bits): $C = [c_1, c_2, \dots, c_t] = [m_1, m_2, \dots, m_t] \oplus [k_1, k_2, \dots, k_t]$
- XOR reminder:
 $0 \oplus 0 = 0,$ $0 \oplus 1 = 1,$ $1 \oplus 0 = 1,$ $1 \oplus 1 = 0$

How do we
decrypt?



We XOR with
the key again!



Is this a
symmetric or
asymmetric
cipher?



Symmetric: we
encrypt and decrypt
with the same key



The One-Time Pad

- Vernam cipher: $C = M \oplus K$
- If the key K of Vernam cipher is randomly (uniformly) generated and never re-used, it is called the **one-time pad**.

- Example: “HELLO” encoded in ASCII-7:

 1101000 1100101 1101100 1101100 1101111

- A possible encryption with a one-time pad:

 0100010 0000100 0010010 0111111 0011110

 1001010 1100001 1111110 1010011 1110001



How would you break the one time pad?



1101000 1100101 1101100 1101100 1101111



0100010 0000100 0010010 0111111 0011110



1001010 1100001 1111110 1010011 1110001

*Secure against an adversary with unlimited computing resources and time!!



This is unbreakable! Indeed, we get something called information-theoretic* security (much stronger than computational security!)



Why doesn't the "try every key" work here?



Given a ciphertext, there exists a key that leads to every possible plaintext. And all keys have the same probability of happening....



Shortcomings of the One-Time Pad

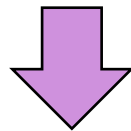
- We have perfect **confidentiality**! What about **integrity**?
- Assume your boss has your salary (in binary) encrypted with a one-time pad, and you have access to the ciphertext. What can you do with this?



000000000000000010101111110010000



11111101010011111001011101000101



XOR with 10000...



01111101010011111001011101000101



Decrypt



100000000000000010101111110010000



Just flip the most-significant bit... this will also flip the most-significant bit of the plaintext, which most likely will be zero!

Shortcomings of the One-Time Pad

- To encrypt an n-bit message, we have to use a secret n-bit key... and we cannot re-use it.
 - Why not just keep the n-bit message “secret”?
- What happens if we re-use the key? (**Two-Time Pad**)

“HELLO”

 1101000 1100101 1101100 1101100 1101111   1001010 1100001 1111110 1010011 1110001

 0100010 0000100 0010010 0111111 0011110

“BYE ”

 1100010 1111001 1100101 0100000 0100000   1000000 1111101 1110111 0011111 0111110

How do we attack this, if we get the ciphertexts?



Just XOR the ciphertexts:

$$C_1 \oplus C_2 = (M_1 \oplus K) \oplus (M_2 \oplus K) = M_1 \oplus M_2$$



Completely insecure!!
 But still requires a bit of work to break.



A visual example

- We get these two images, that have been encrypted with a one-time pad (but with the same pad/key).



C_1



C_2



$C_1 \oplus C_2$



43



- What do we do with them?

CPEN 442 – Introduction to Cybersecurity

Module 2 – Cryptography



Part 3 – Symmetric Crypto

Part 1:
Preamble

Part 2:
Ancient Crypto

- Definition
- Example: textbook RSA
 - Description
 - Shortcomings
- Key sizes
- Hybrid cryptography

Part 4:
Public-key
Crypto

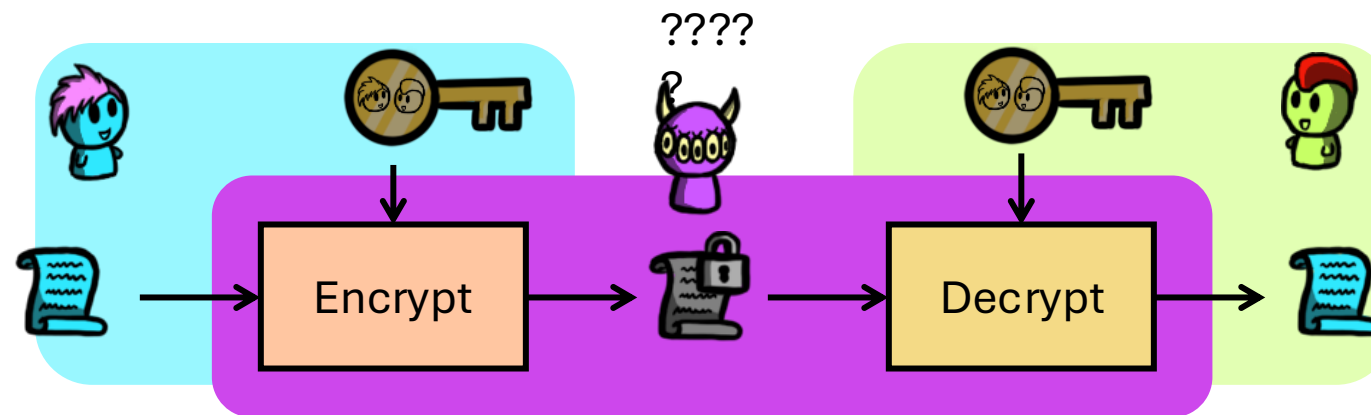
Part 5:
Integrity and
Authentication

Symmetric (or Secret-key) cryptography

- We use the **same key** to encrypt and to decrypt



This “key exchange” is easy to do in person, but harder to do over the Internet. Eve could be watching! (we’ll see how to fix this later)

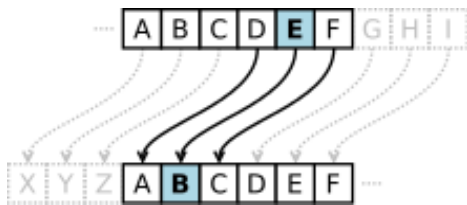


The encrypt and decrypt boxes do not have to do the same operations with the key.
As long as the same key can be used to encrypt/decrypt, it is symmetric crypto



Ancient crypto: which ones are symmetric?

Caesar cipher



Affine cipher

$$y = (xa + b) \bmod 26.$$

Also, which ones use the same algorithm for encrypting and decrypting?



Vigenère cipher

Plaintext: ATTACKTONIGHT

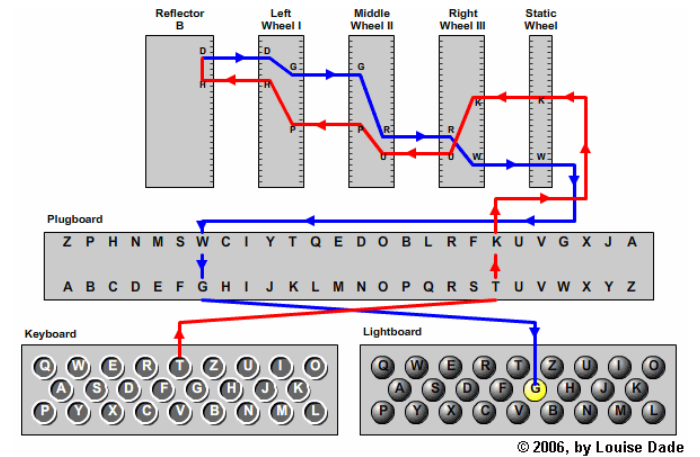
Key: MYKEYMYKEYMYK



Ciphertext: MRDEAWRYRGSFD

Vernam cipher

$$C = M \oplus K$$



Computational security

- We saw a bunch of weak ciphers (Caesar, Playfair, etc.)
- We saw a cipher with “perfect” or “information-theoretic” security.
- Next, we will see strong (modern) ciphers.
- For all of these, remember: they have “**computational**” security.
 - This means they can be broken if Eve does enough work.
 - At **worst**, Eve tries every key.
 - Usually, there are shortcuts (we saw “shortcuts” for the ancient ciphers).
- Let’s reason about how much work Eve has to make.



The cost of brute-force attacks



Let's say we have keys of b bits.
How many possible keys are there?



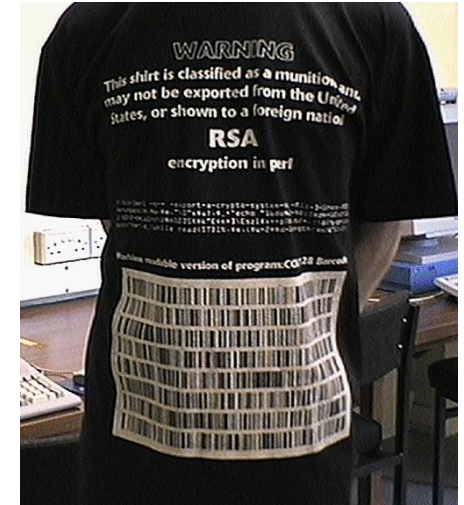
Let's say the adversary can do s decryptions per second.
How long will it take for the adversary to try *all* keys?



How long does it take *on average* for the adversary to find the right key?

Computational security: some numbers

- 40-bit crypto: was the legal export limit for a long time in the US (cryptosystems were classified as munitions until the late 90's).
- 56-bit crypto: was the US government standard (DES) for a long time.
- 128-bit crypto: modern standard.
- 256-bit crypto: we “think” enough for post-quantum security in AES.



Key size	Computer ($\approx 1.7 \cdot 10^7$ keys/second)	Lab of 100 computers ($\approx 1.7 \cdot 10^9$ keys/second)	Bitcoin network in 2025 ($\approx 8 \cdot 10^{20}$ keys/second)
40-bit	18 hours	11 minutes	1.3 ns
56-bit	134 years	16 months	0.09 ms
128-bit	$6.3 \cdot 10^{23}$ years	$6.3 \cdot 10^{21}$ years	$1.3 \cdot 10^{10}$ years
256-bit	...	...	...

Sep
2026 is
around
1 ZH/s
(10^{21})

- $1.3 \cdot 10^{10}$ years: this is the age of the universe, and it is slightly larger than the expected lifetime of the sun.

Computational security: some numbers

- Wait, but computers get faster over time!
- Moore's law: "the number of transistors on a microchip doubles about every two years".
- We can just wait a few hundreds of years, then crack the key in less than one hour!

Do we believe this?

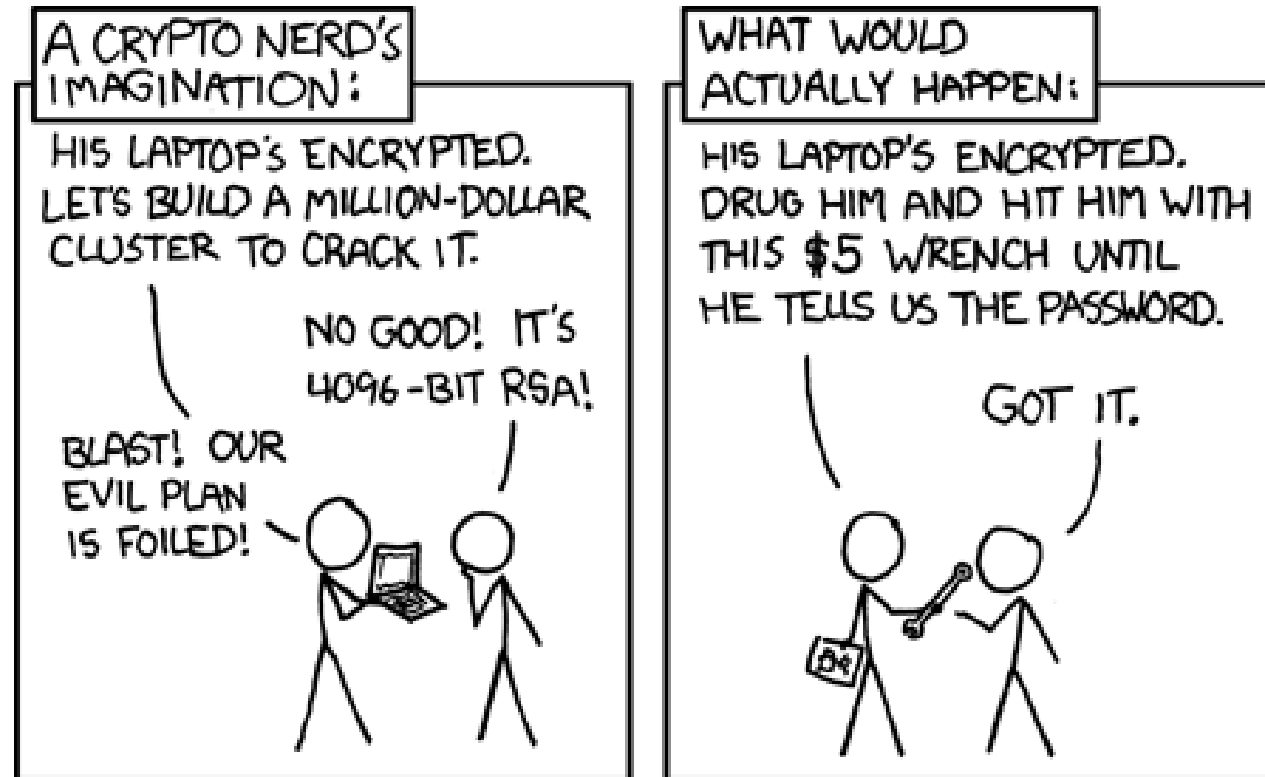


What about quantum computers?



- Actually, there is a much better strategy...

A better strategy

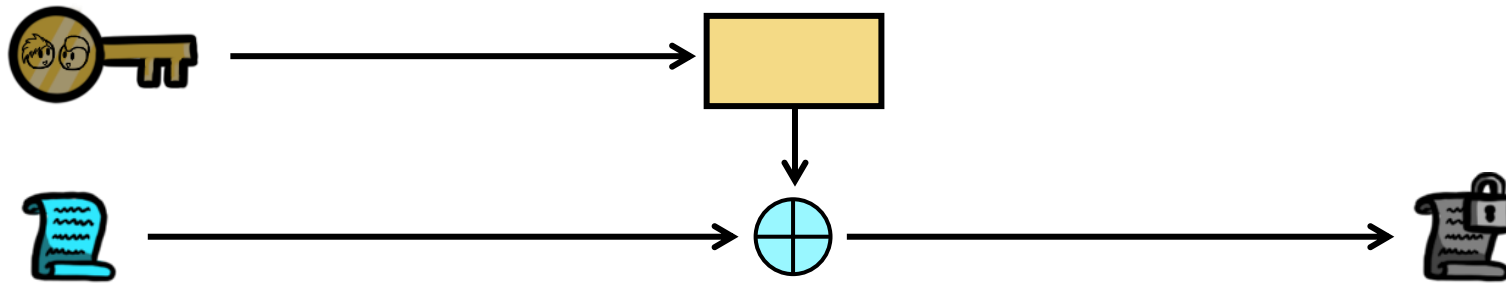


<https://xkcd.com/538/>

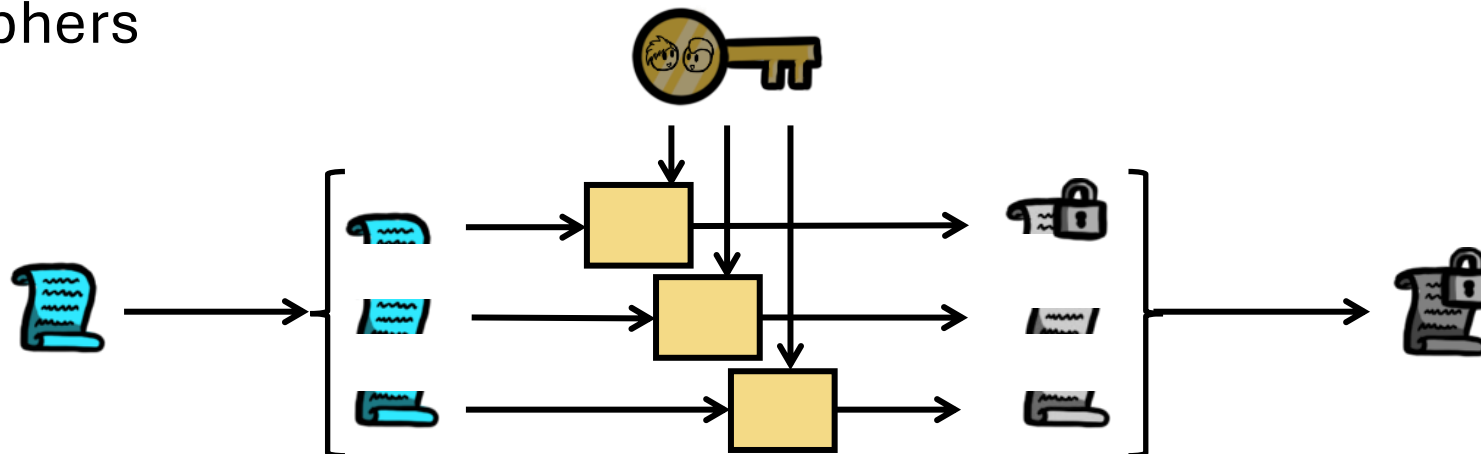
Types of symmetric cryptosystems

Secret-key cryptosystems come in two major classes:

- Stream ciphers

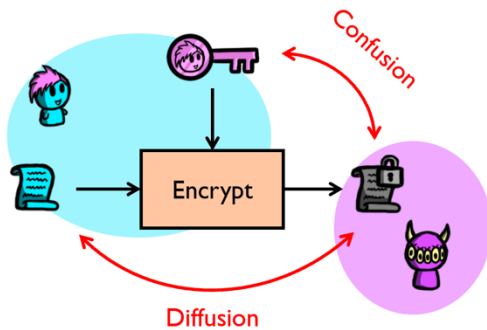
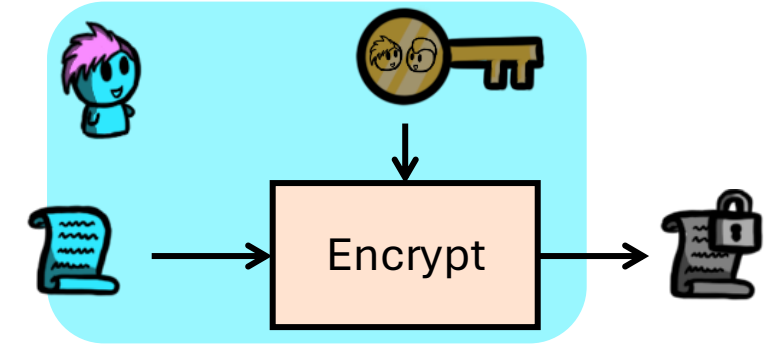


- Block ciphers

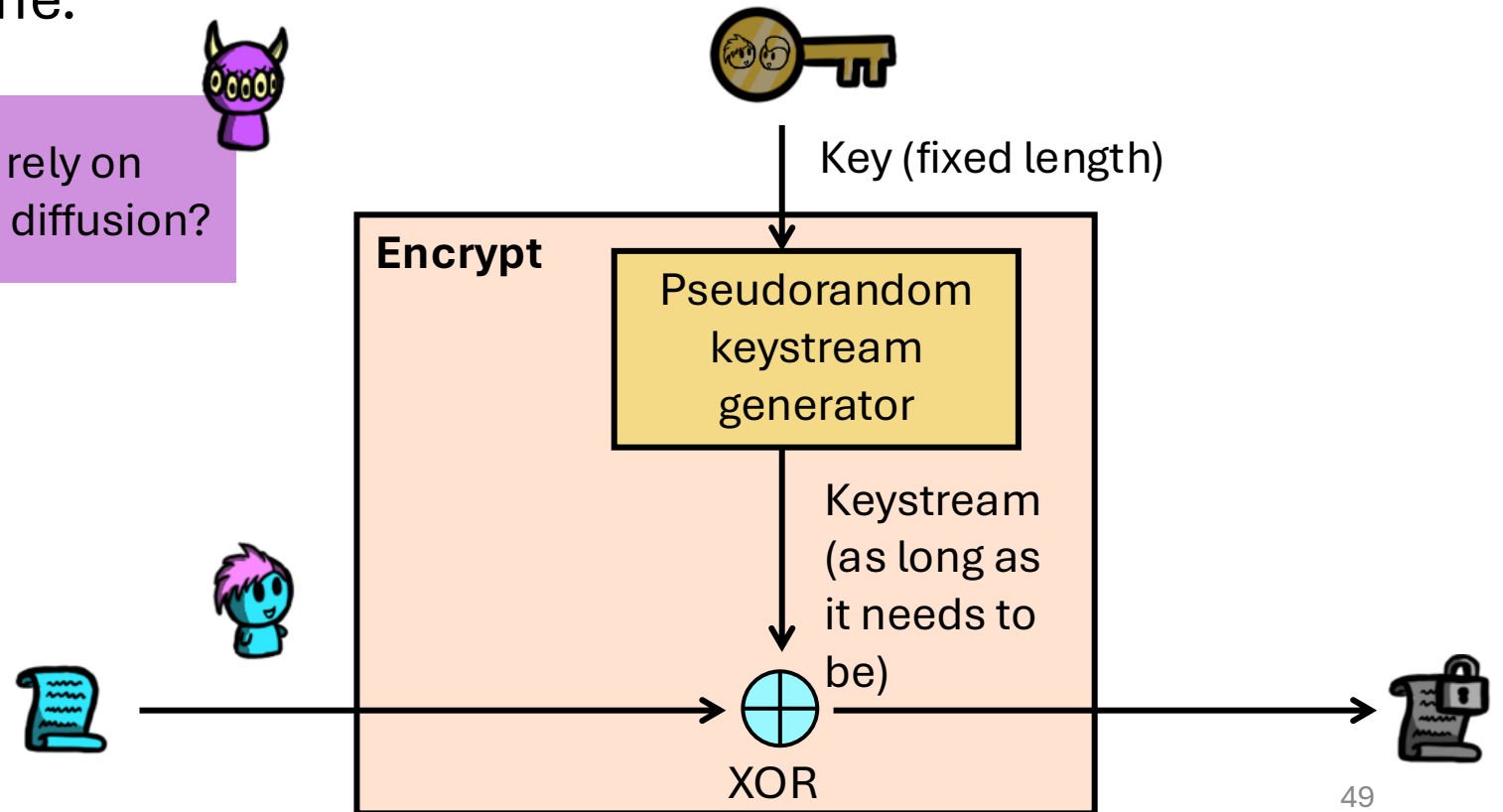


Stream ciphers

- A stream cipher operates one bit at a time.
- Basically, like the one-time pad, but using a **pseudorandom keystream**, instead of a truly random one.

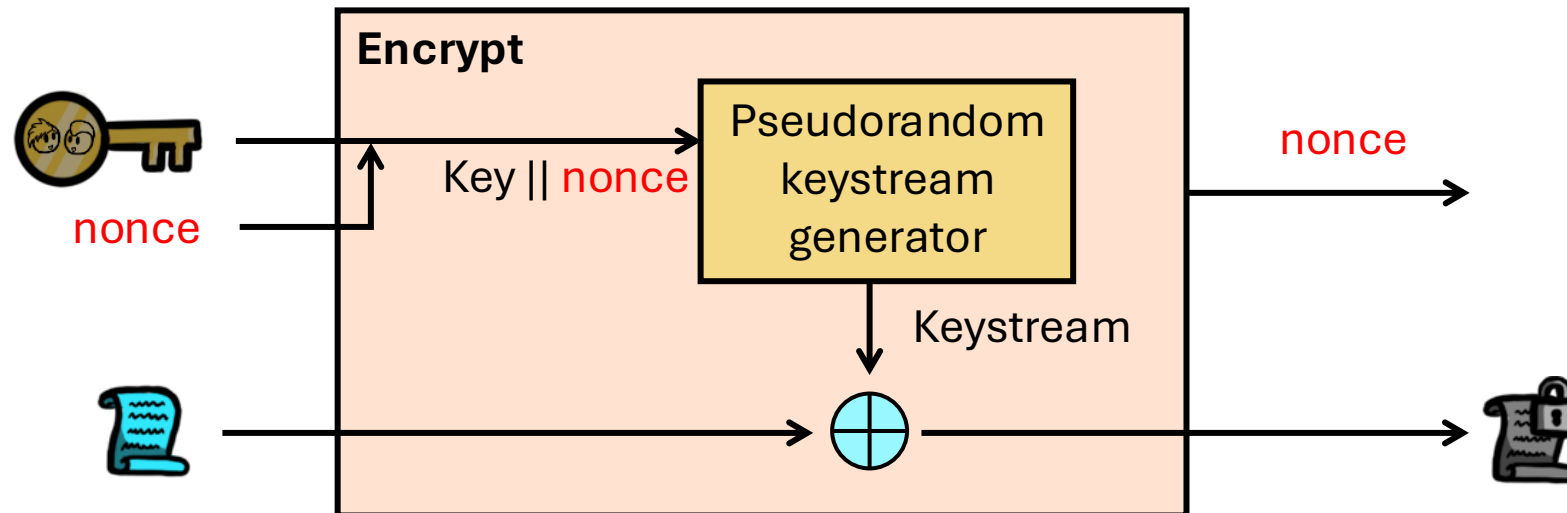


Do these rely on confusion or diffusion?



Stream ciphers: preventing keystream re-use

- Stream ciphers can be very **fast**
 - This is useful if you need to send a lot of data securely.
 - You can usually encrypt bit-by-bit
- However, be careful with **key(stream) re-use!** (remember the two-time pad)
- Solution: concatenate the key with a **nonce**. (This is also called the “salt”)

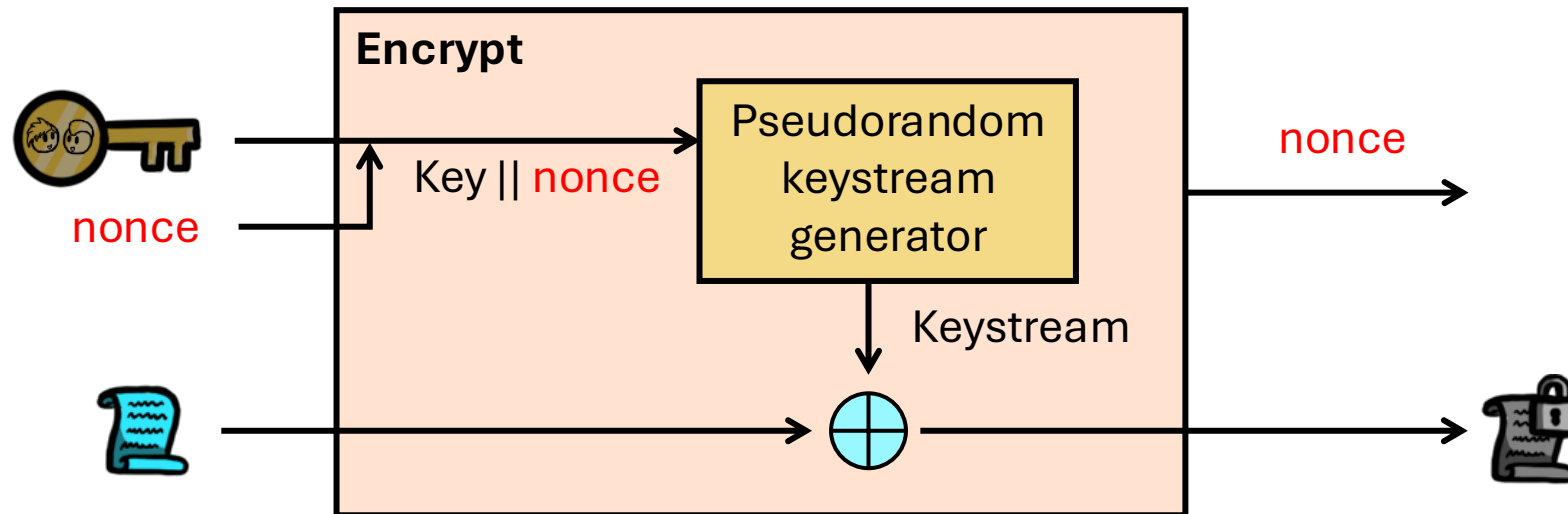


Why do we need to send the nonce here?



Using the nonce

- We pick the key once.
- For every new message we want to encrypt, we choose a **fresh nonce**.
- This prevents keystream re-use!





Nonces vs Keys

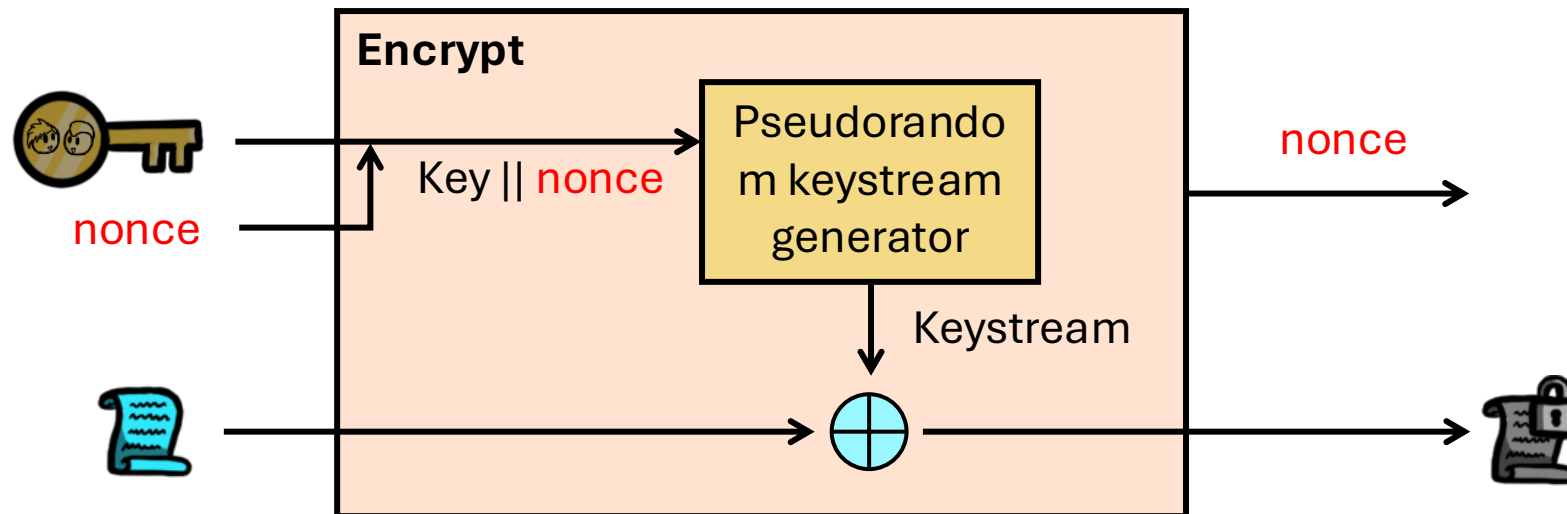
- Very important conceptual difference!! Keys are secret, nonces are not



What do we gain by making the key longer?



What do we gain by making the nonce longer?

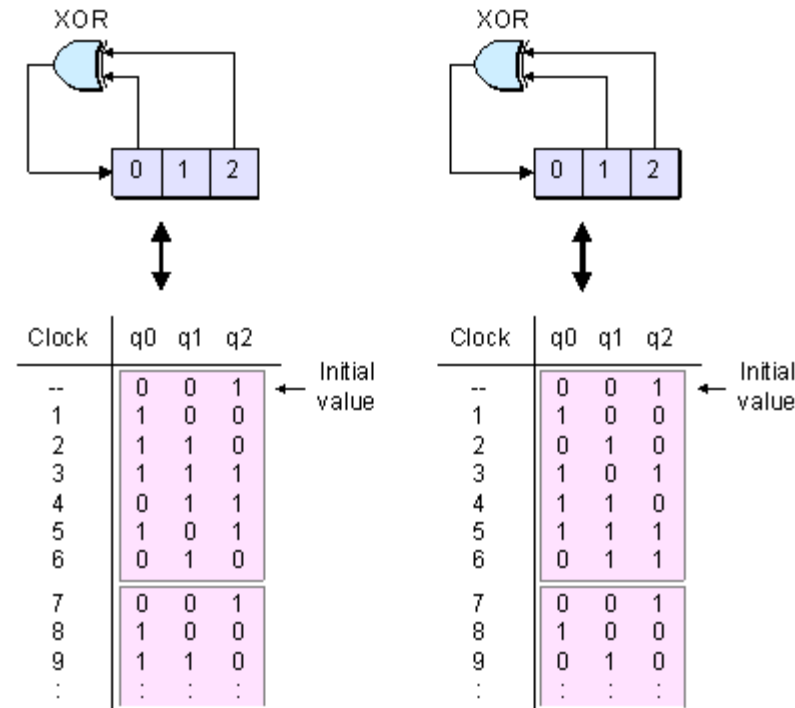


What do we gain by making the nonce secret?



Pseudo-random number generators (example)

- There is some math involved in here, that we do not have time to see.
- A common approach is using Linear-Feedback Shift Registers (LFSRs)
- A toy example:



Notice the sequence has a period of 7



The A5/1 cipher

- Created in 1987 and used in GSM (mobile communications).
- Initially secret, but reverse engineered in 1999.
- This is **not secure**! But it's a nice toy *example*.
- A5/1 is a combination of three LFSRs:
 - The registers are of size 19, 22, and 23.
 - They sum 64, which is the size of the secret key.
 - The period of the sequence is actually 2^{64} .
- You don't need to know any details here: just have a general idea of how this “works”.
 - (You don't need to know about those orange register positions either...)

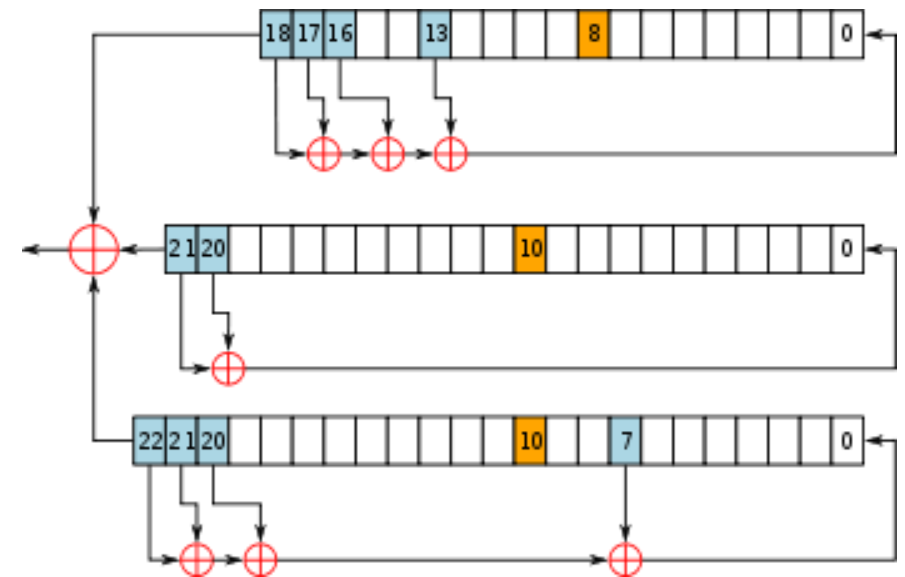


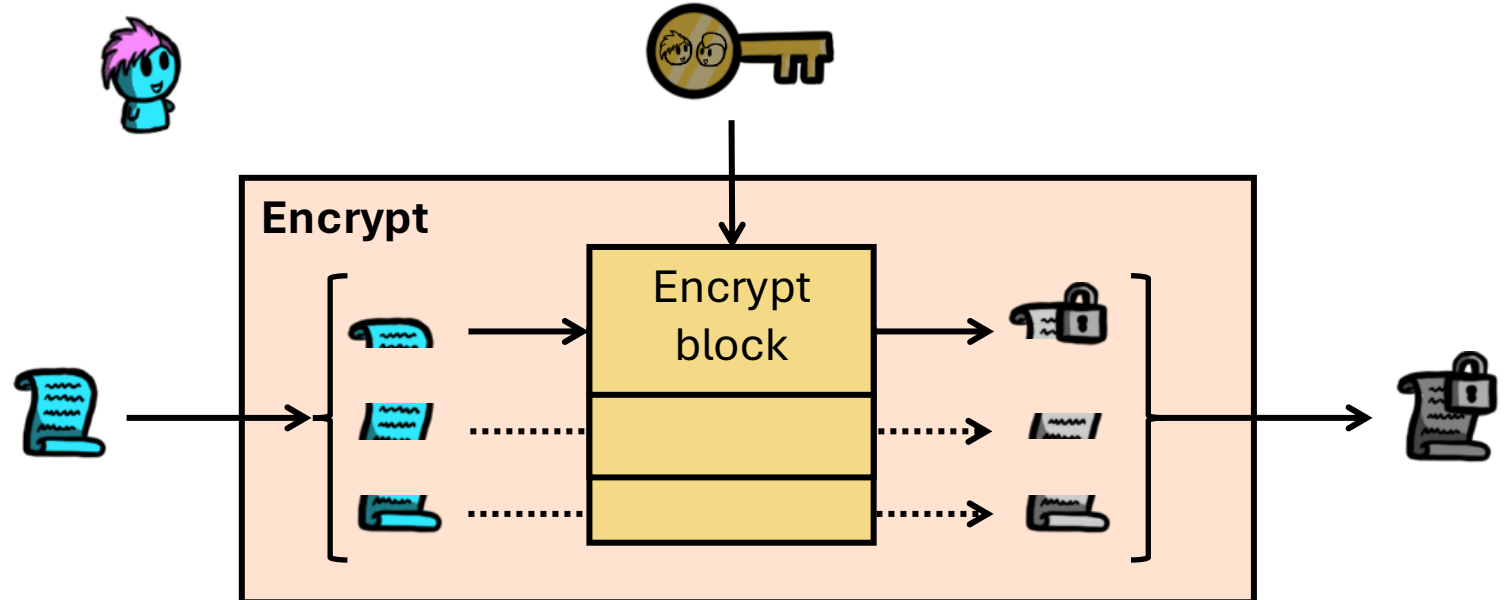
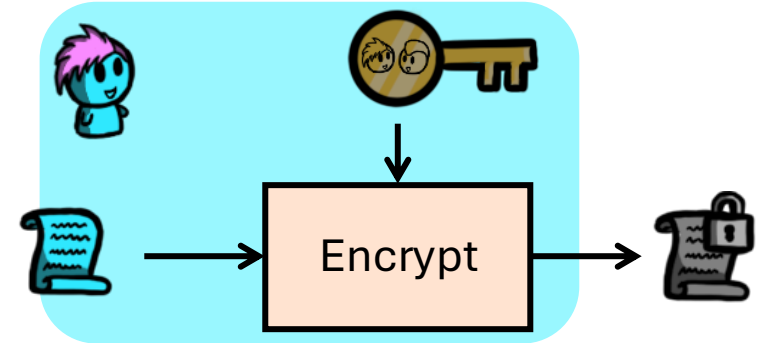
Image from:
<https://en.wikipedia.org/wiki/A5/1>

Relevant Stream Ciphers

- WEP and PPTP are great examples of how not to use stream ciphers.
- **RC4** was the most common stream cipher on the Internet but is now deprecated.
- **ChaCha20** is increasingly popular (Chrome and Android).
 - Usually paired with Poly1305 for authentication.
- **SNOW3G** in mobile phone networks.
 - Although AES is also used in 5G

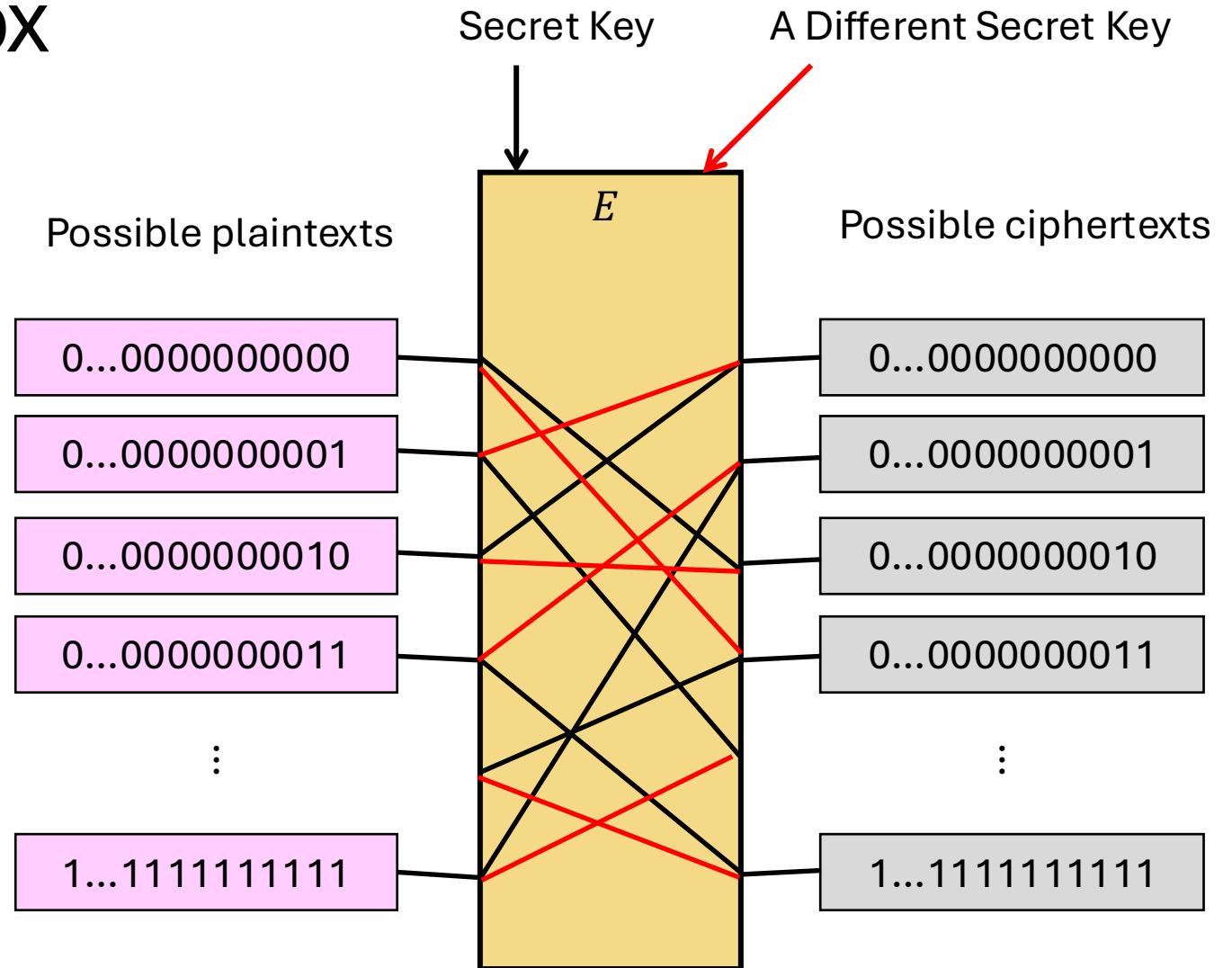
Block ciphers

- A block cipher operates one block at a time.
- Blocks are usually 64 or 128 bits long.
- These “block encryption” boxes provide both confusion and diffusion.
- If the plaintext is smaller than one block: padding
- If it is larger: multiple blocks
 - What we do with multiple blocks is called **mode of operation** of the block cipher.



The ideal “Encrypt” box

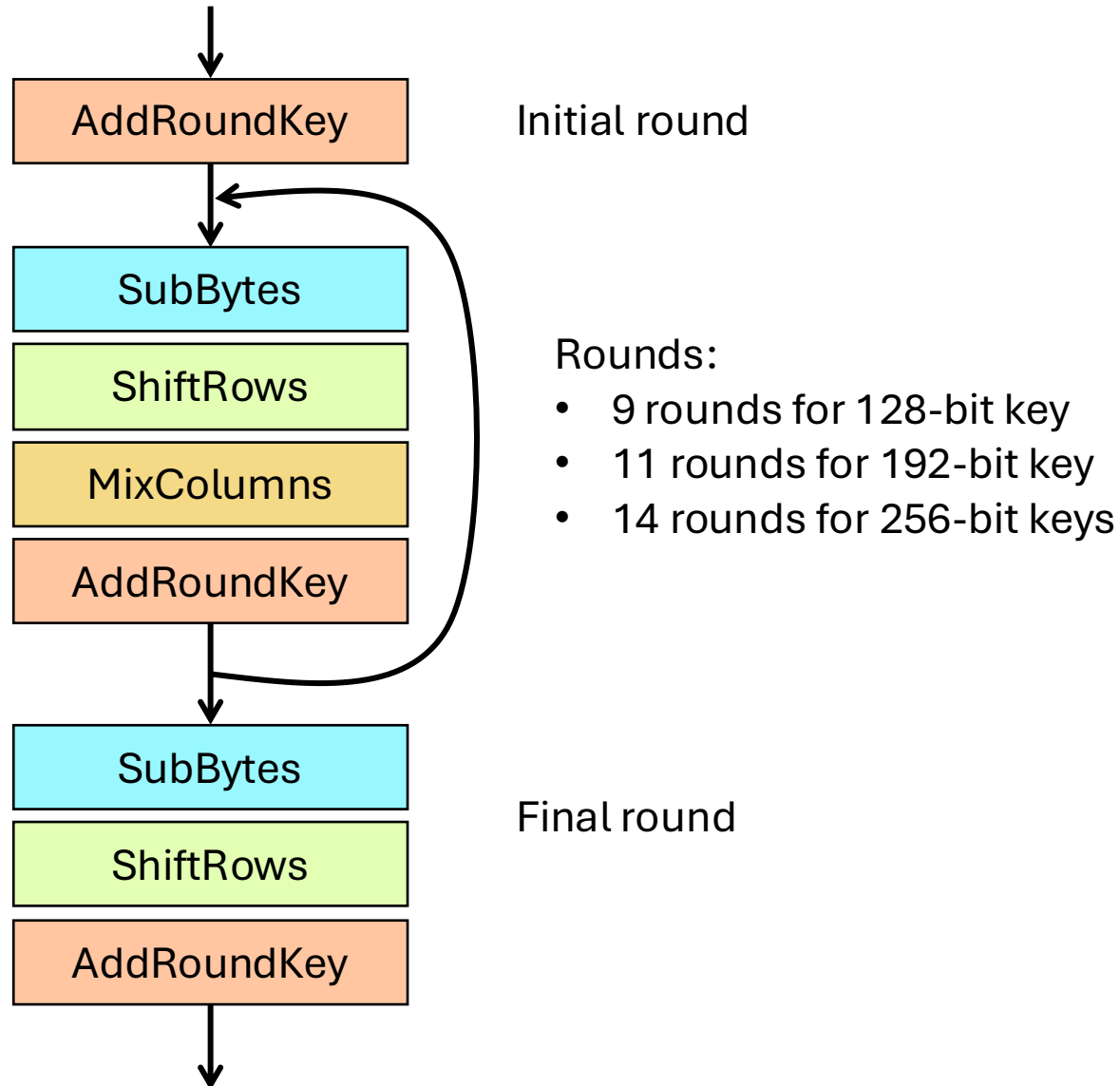
- The *ideal* encrypt box is a **random permutation** that is characterized by the secret key.
- Example on the right: assume the block size is 128 bits.
- Every new key chooses one permutation at random.



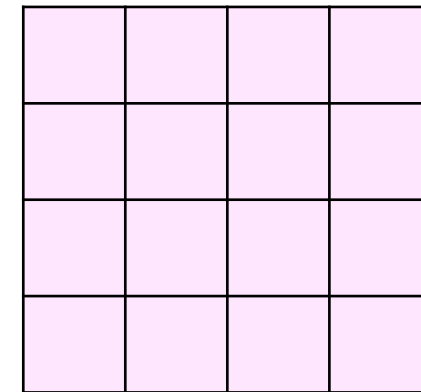
Designing a realistic encrypt box.

- We do not know how to instantiate that ideal encrypt box.
- We design algorithms that try mimic this random permutation.
- In 1977, the National Institute of Standards and Technology (NIST) adopted the Data Encryption Standard (DES) algorithm.
- This was the “official” encryption algorithm for a long time.
- In 1997, NIST announced the Advanced Encryption Standard (AES) competition.
- In 1998, 15 AES candidates were announced.
- In 2000, NIST announced its selection of AES (Rijndael).
- Today: if you are going to use a block cipher, you should always use **AES (Rijndael)**, unless you have a very, very good reason.

Advanced Encryption Standard (AES)

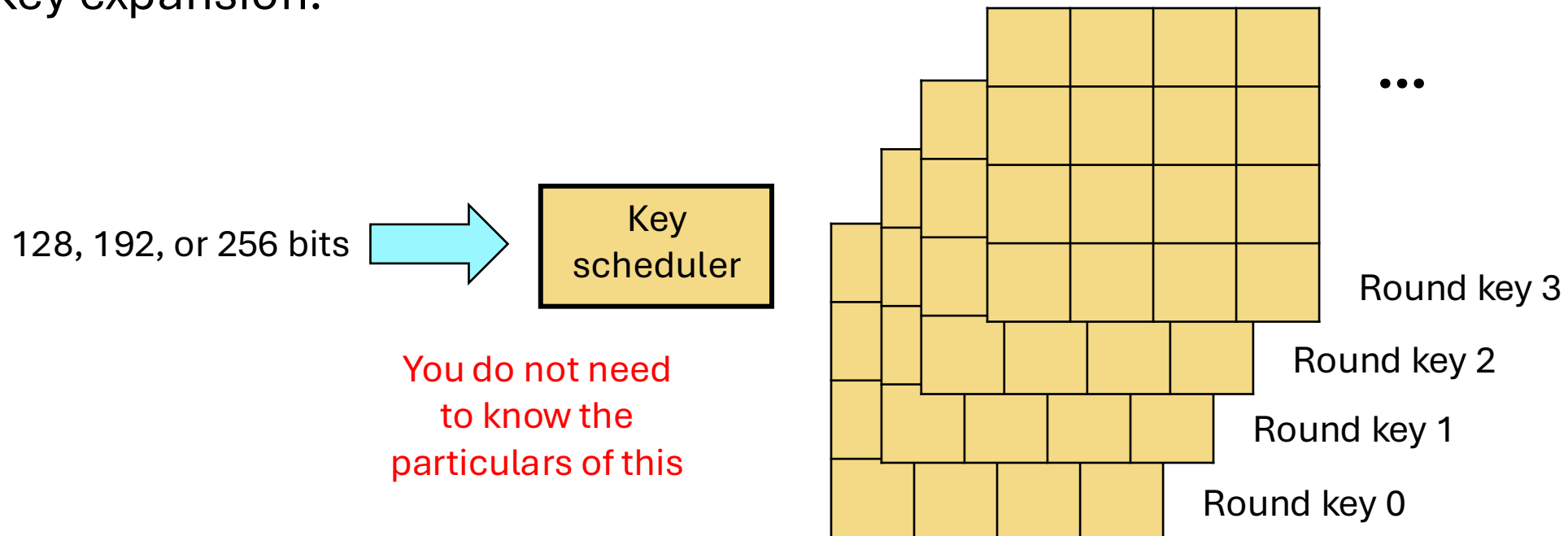


- Based on substitution-permutation networks
- Block size of 128 bits (fixed)
- Key sizes of 128, 192, or 256 bits
- AES operates on a 4x4 matrix where each element is a byte (128 bits total)



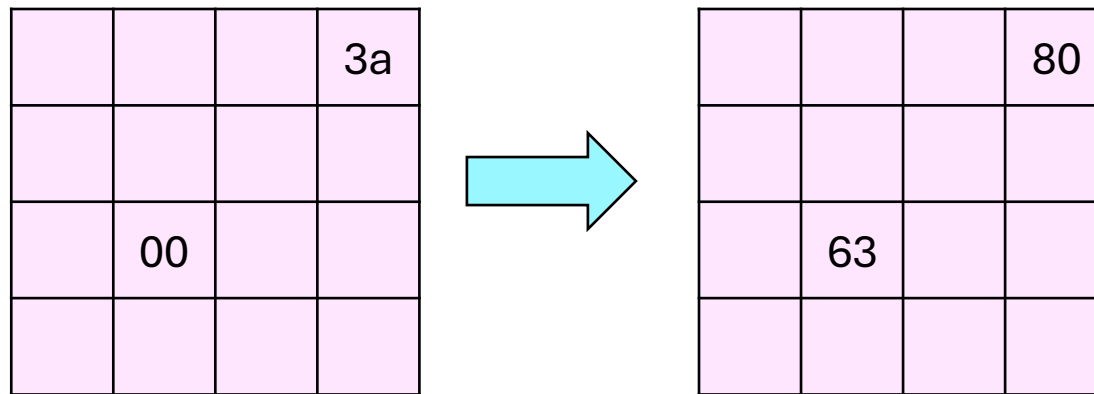
AES Key Schedule

- The AES key gets “expanded” into different sub-keys to use in the AddRoundKey steps of the AES algorithm.
- Key expansion:



Step 1: SubBytes

- Convert each byte using a substitution box (S-box)
- Each byte is replaced for another byte
- For decryption: we use the inverse substitution
- Example (representing bytes in hex)



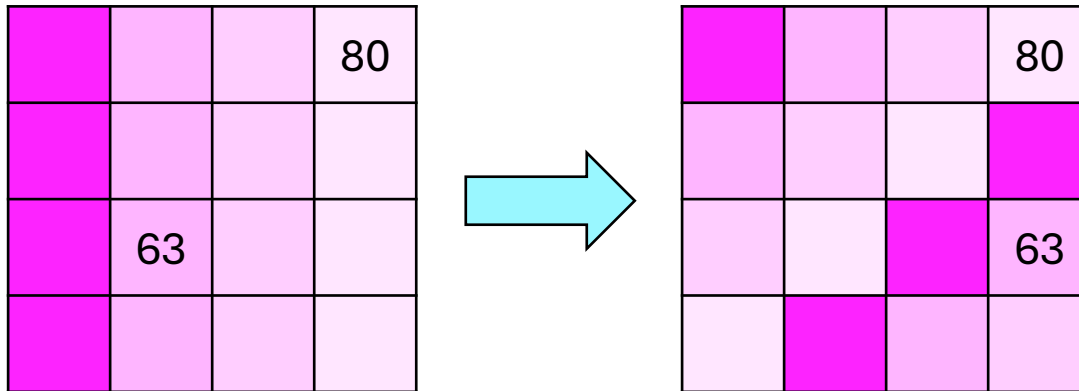
- This adds **confusion** (very non-linear, destroys patterns)

AES S-box

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Step 2: ShiftRows

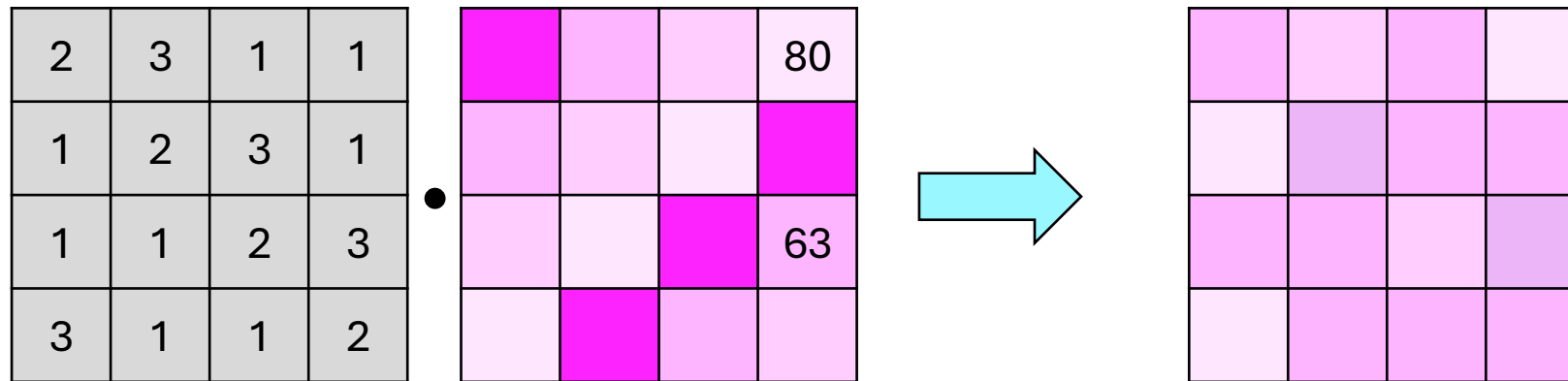
- Shift the i th row ($i \in [0,1,2,3]$) i positions to the left



- This adds **diffusion** (take this with a grain of salt, it does not make sense to think of these properties at a “step-by-step” level)
- To decrypt, you do the opposite

Step 3: MixColumns

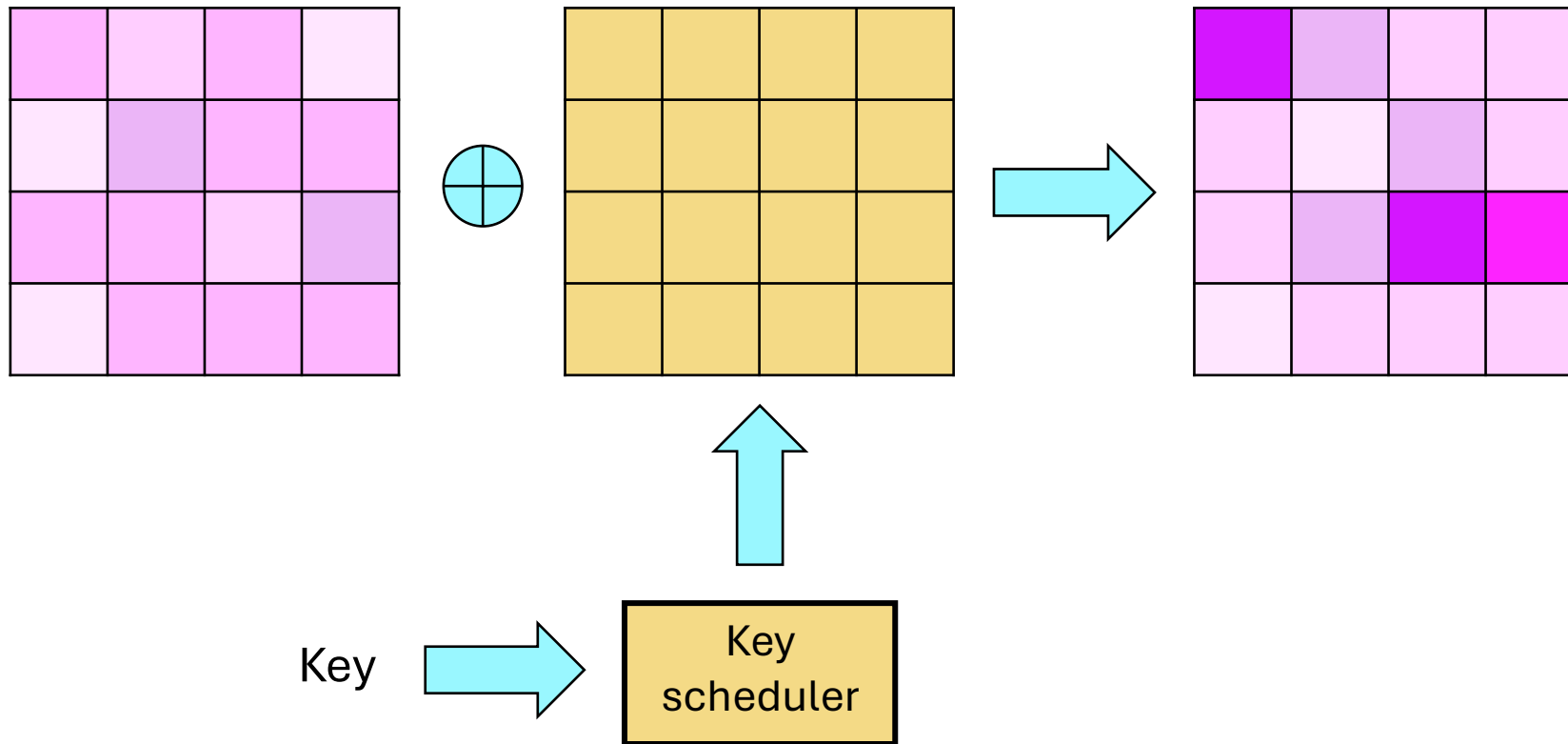
- Multiply each column using a linear transformation (matrix multiplication)



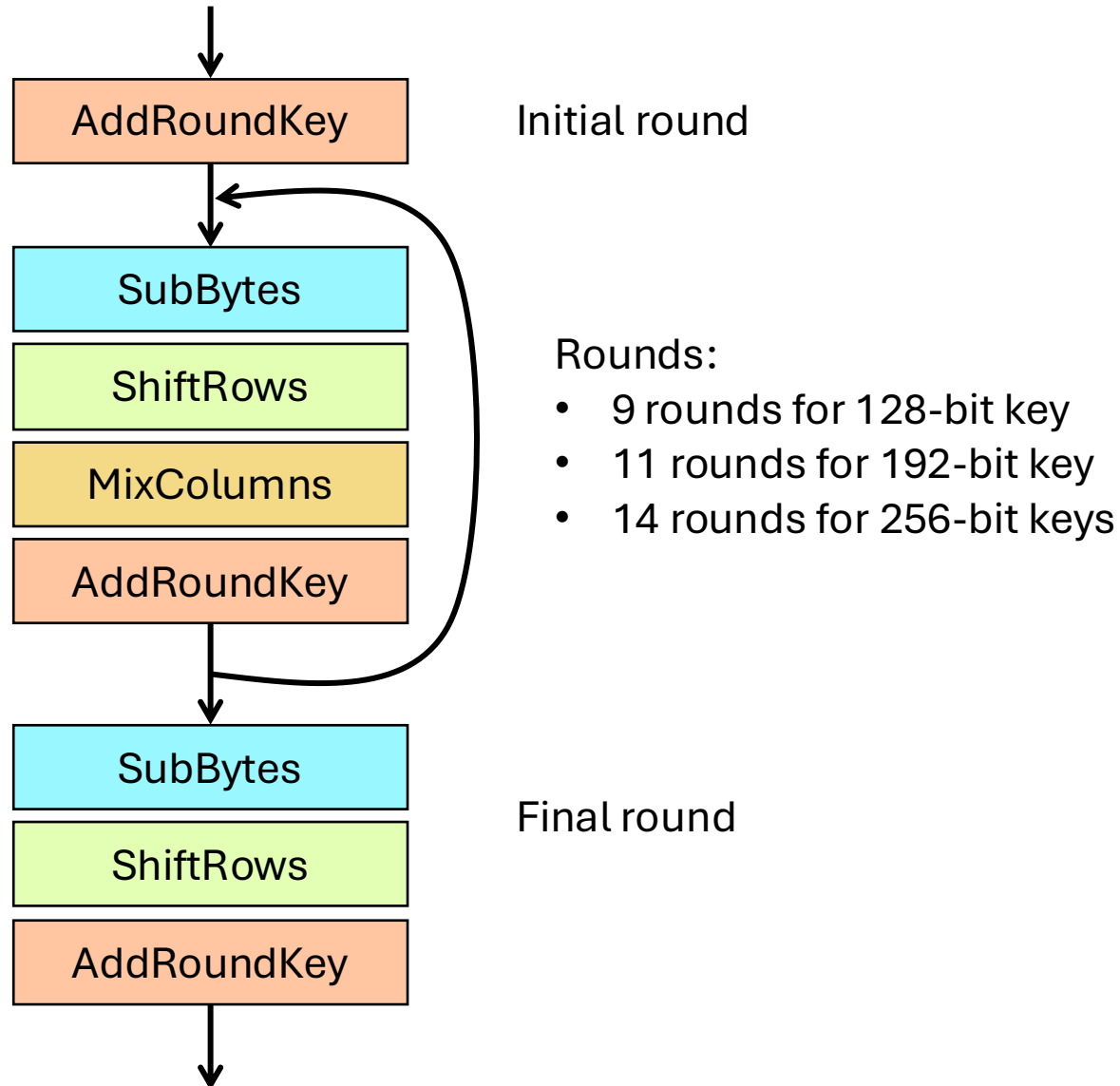
- AES arithmetic is in Galois Field using polynomials... but you don't need to know anything about this, other than it's adding some more **diffusion**, and it's an invertible operation.

Step 4: AddRoundKey

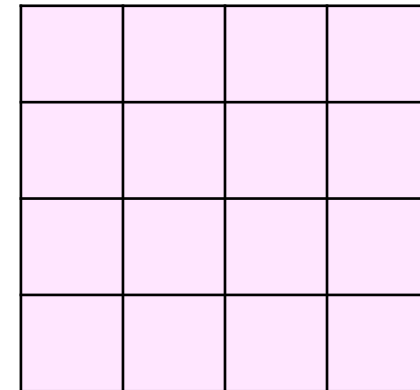
- XOR the matrix with another matrix derived from the AES key, from some “key schedule” that generates a “subkey” for each round



Advanced Encryption Standard (AES)



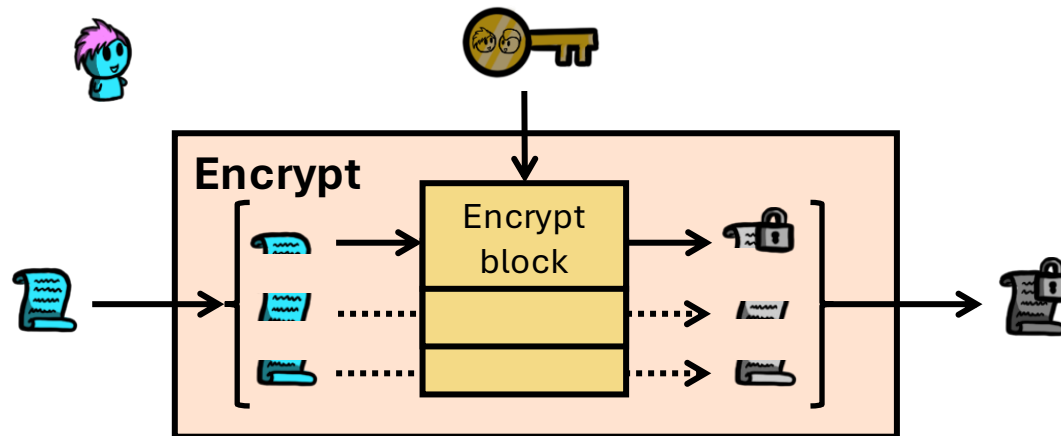
- Based on substitution-permutation networks
- Block size of 128 bits (fixed)
- Key sizes of 128, 192, or 256 bits
- AES operates on a 4x4 matrix where each element is a byte (128 bits total)



Modes of operation

An encrypt box encrypts a plaintext block into a ciphertext block.

- Smaller plaintext: we use padding (e.g., PKCS#7)
- Larger plaintext: split and use modes of operation.



We will see the following modes of operation, but there are more...

- Electronic Code Book (ECB)
 - Do not use this!!
- Cipher Block Chaining (CBC)
 - You use this with authentication (MAC) only!
- Counter Mode (CTR)
 - Interesting but you use GCM instead

An important one that we do not have time to see:

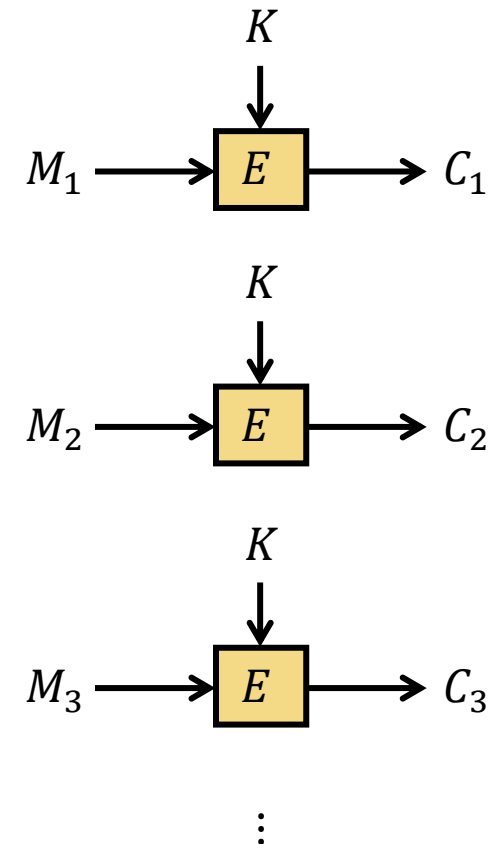
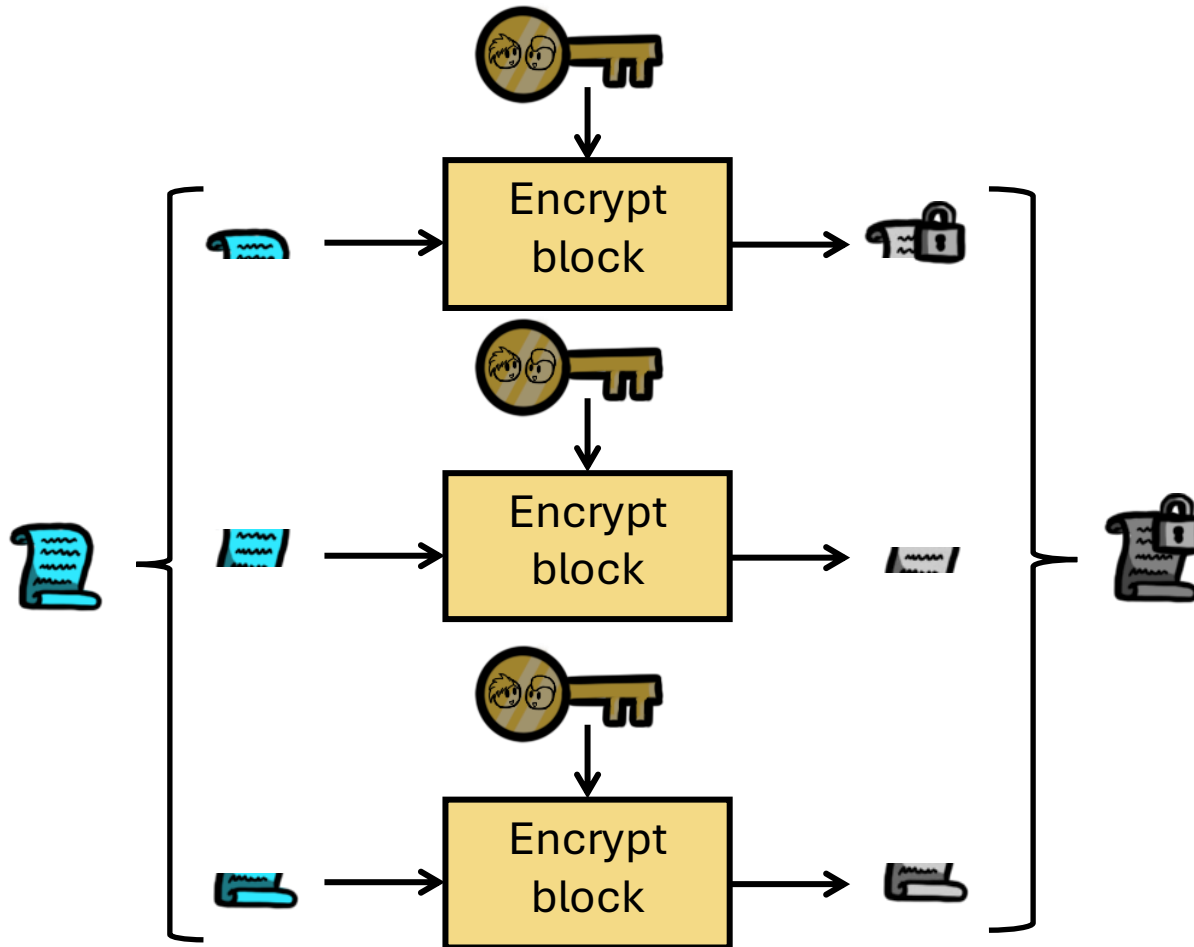
- Galois Counter Mode (GCM)
 - This mode is similar to CTR, but also adds authentication

Electronic Code Book (ECB) mode

- **ECB**: encrypt each successive block separately

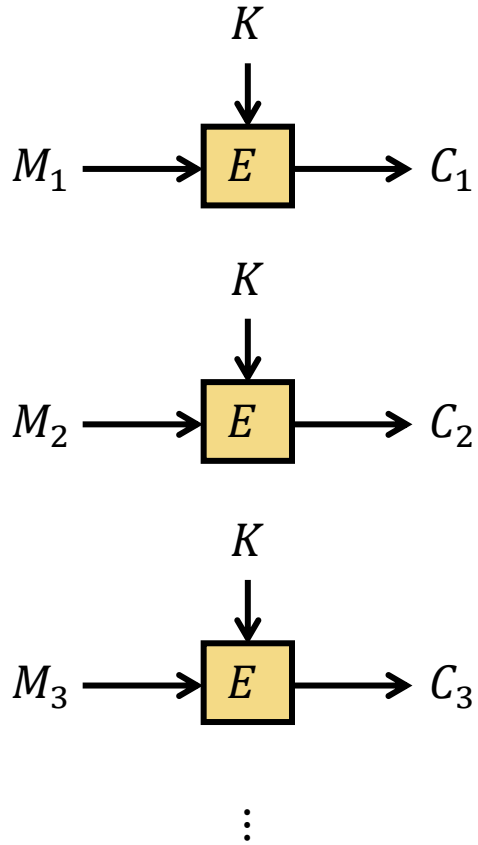


Enough drawings,
cause we need
space now...

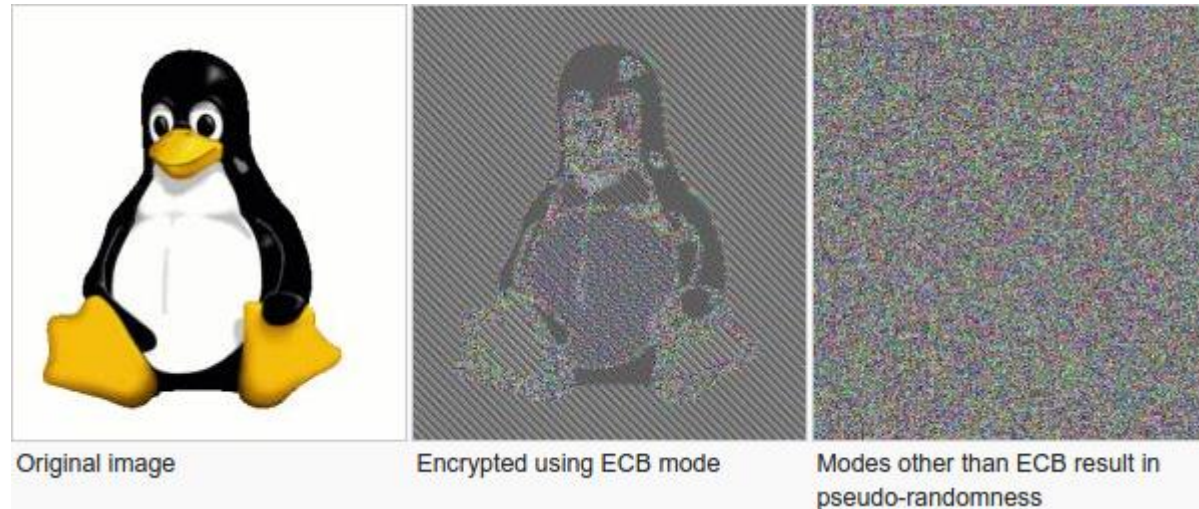




Electronic Code Book (ECB) mode

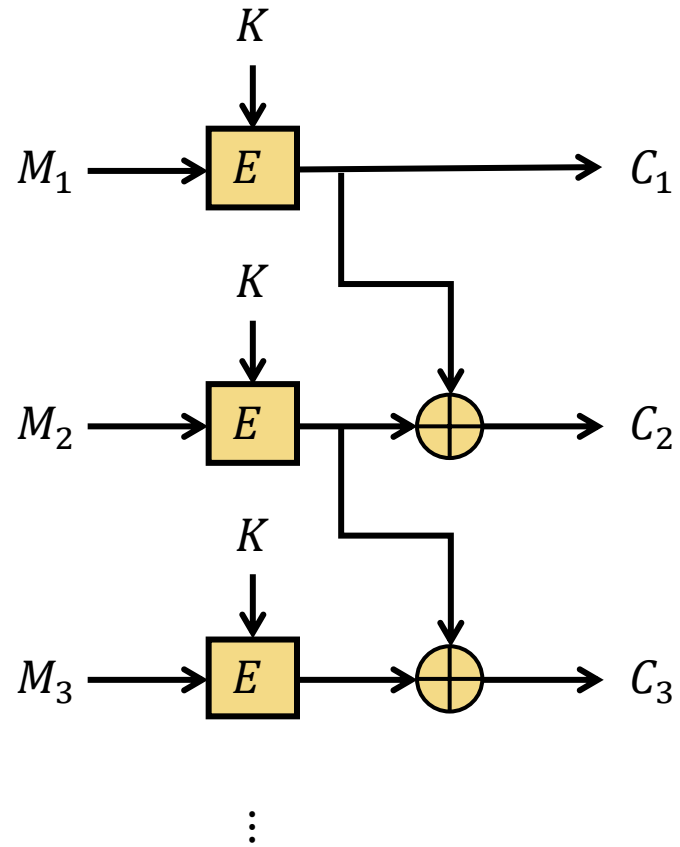


What happens if the plaintext M has some blocks that are identical $M_i = M_j$?





Improving ECB



- We can provide “feedback” among different blocks, to avoid repeating patterns.

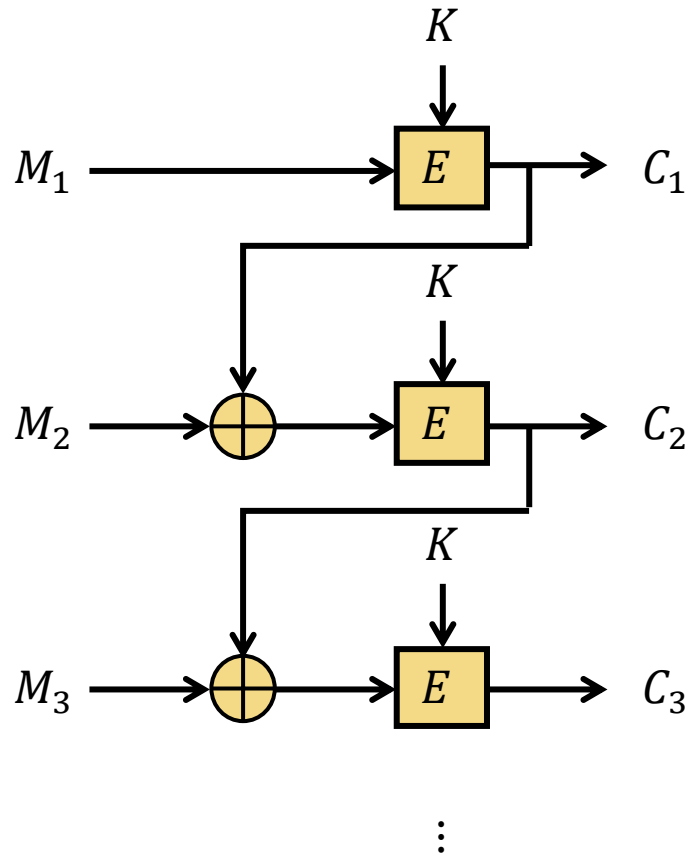
Does this avoid repeating patterns?
Any issues here?





Improving ECB

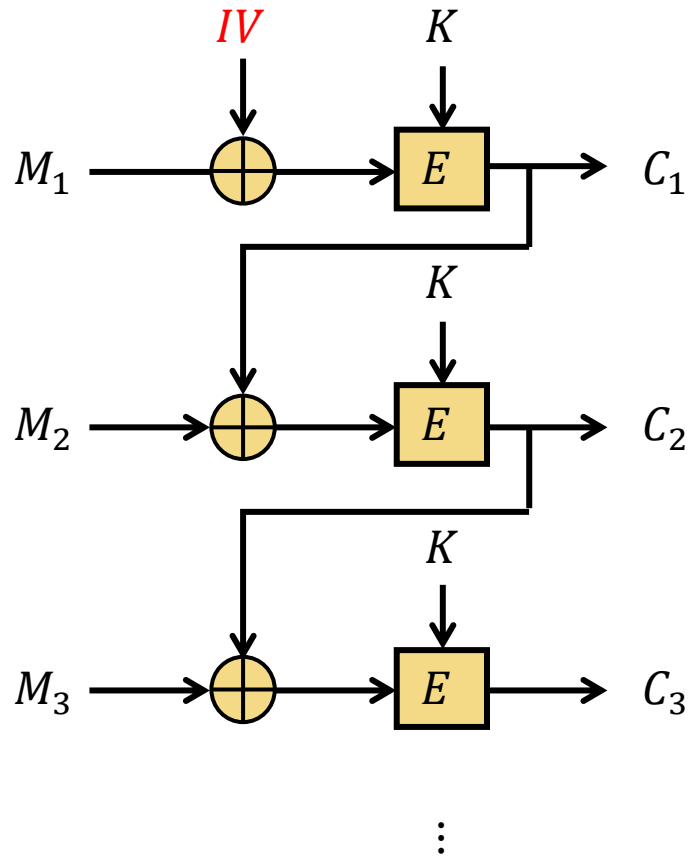
- Another way of providing “feedback”



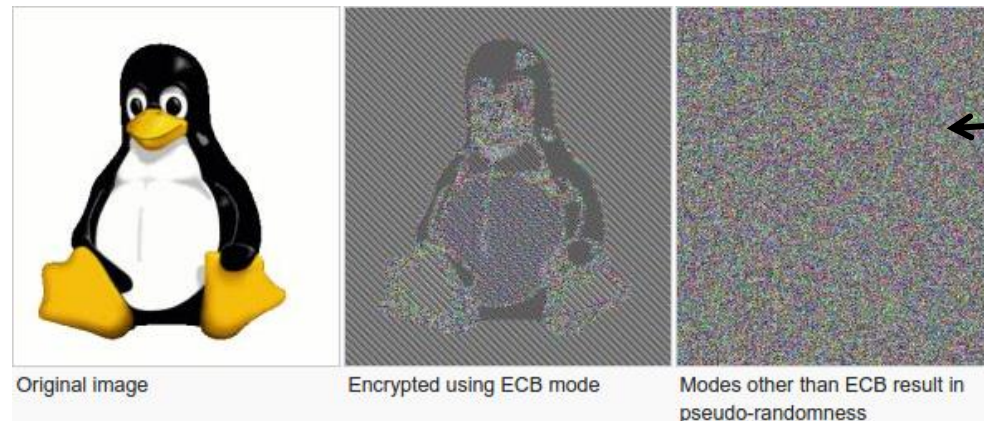
Does this avoid repeating patterns?
Any issues here?

What would happen if we encrypt the same message twice with the same key?

Cipher Block Chaining (CBC) mode

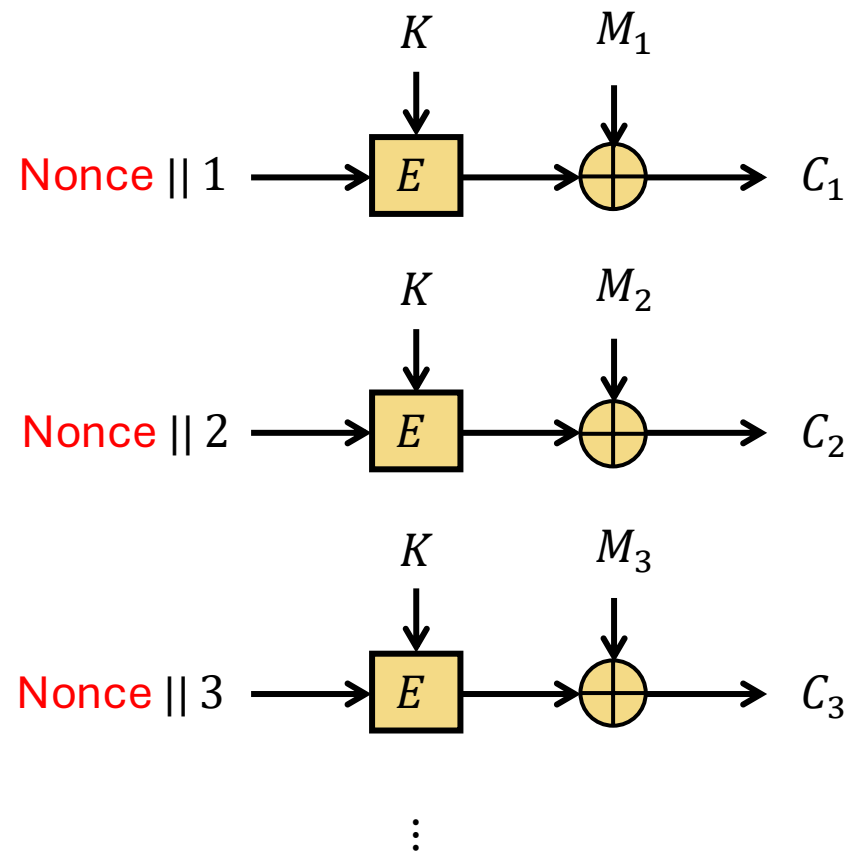


- This solves the issue of repeating patterns in blocks or the whole plaintext.
- The **IV** is the **Initialization Vector**.
 - Also called **nonce**.
 - Also called **salt**.
- We also share the IV in the clear!
 - Remember it is “conceptually different” than the key
 - Do not re-use it!



Counter mode (CTR)

- Counter mode turns a block cipher into a stream cipher
- GCM also adds authentication, ~~we might see it later...~~



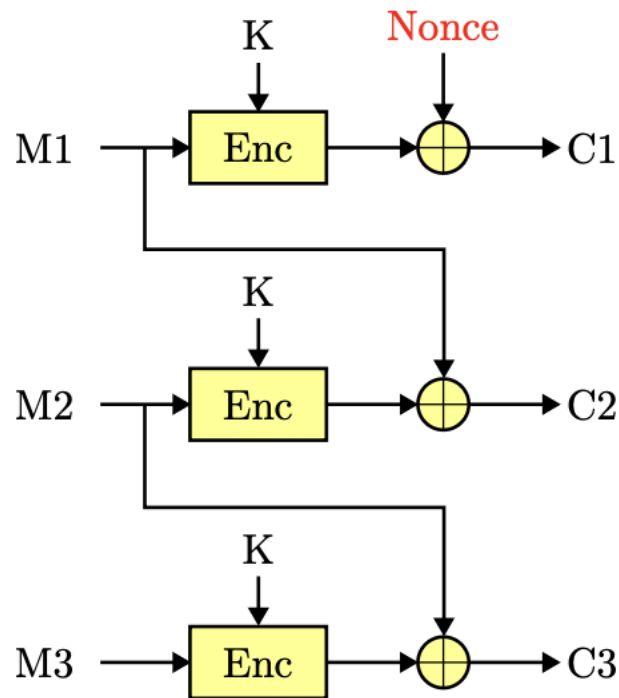
Symmetric cryptography, concluding remarks

- Good choices: AES-GCM or ChaCha20+Poly1305
 - GCM and Poly1305 provide “authentication” (and integrity). You do not need to know the details.
- That’s it, we have solved confidentiality!
- However... this requires having a shared key between sender and receiver.
- How do we do this?
 - Meet in person
 - Diplomatic courier
 - ...
- Or... we invent new technology... (next up)
- (we still don’t have integrity and authentication, that’s in the final part of the module)

Question from midterm last year

- The students were given “new” modes of operation and had to answer questions about them.

For each mode of operation below, answer the following questions and include a *brief explanation* of your answer:



- Given ciphertext C , can Eve guess *repeating patterns* about the plaintext M ?
- Given two ciphertexts generated with different nonces, can Eve learn anything about the corresponding plaintexts that cannot be learned from each ciphertext separately?

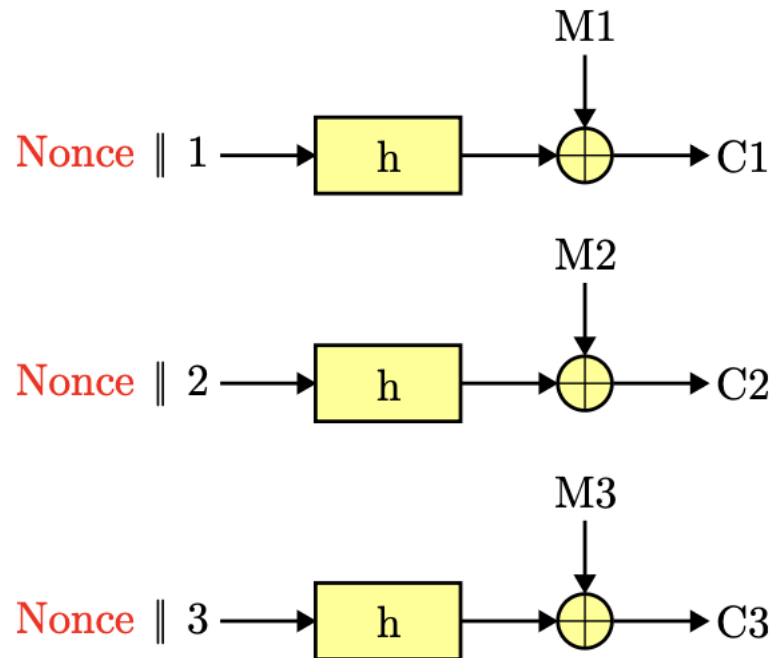
This was confusing in the exam, so someone asked me, and I clarified that you could answer whether the adversary can tell if the corresponding plaintexts are equal or not.

- Does the mode of operation provide confidentiality? In other words: can Eve learn meaningful information about the plaintext from the ciphertext in reasonable time? (If the only information that can be learned is from the two questions above, you can just say so in the answer to this question.)

Question from midterm last year

- We have not seen hash function yet (h); these are functions that are very hard to invert (they are one-way functions).

For each mode of operation below, answer the following questions and include a *brief explanation* of your answer:



- Given ciphertext C , can Eve guess *repeating patterns* about the plaintext M ?
- Given two ciphertexts generated with different nonces, can Eve learn anything about the corresponding plaintexts that cannot be learned from each ciphertext separately?

This was confusing in the exam, so someone asked me, and I clarified that you could answer whether the adversary can tell if the corresponding plaintexts are equal or not.

- Does the mode of operation provide confidentiality? In other words: can Eve learn meaningful information about the plaintext from the ciphertext in reasonable time? (If the only information that can be learned is from the two questions above, you can just say so in the answer to this question.)

CPEN 442 – Introduction to Cybersecurity

Module 2 – Cryptography



Part 4 – Public-key Crypto

Part 1:
Preamble

Part 2:
Ancient Crypto

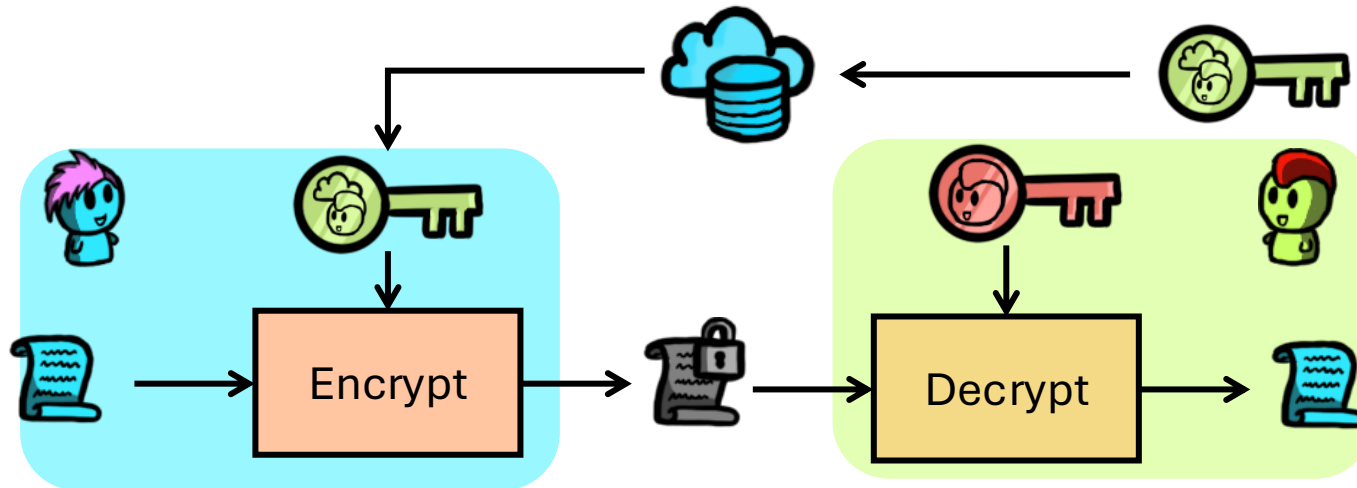
Part 3:
Symmetric
Crypto

- Definition
- Example: textbook RSA
 - Description
 - Shortcomings
- Key sizes
- Hybrid cryptography

Part 5:
Integrity and
Authentication

Public-key (or Asymmetric) cryptography

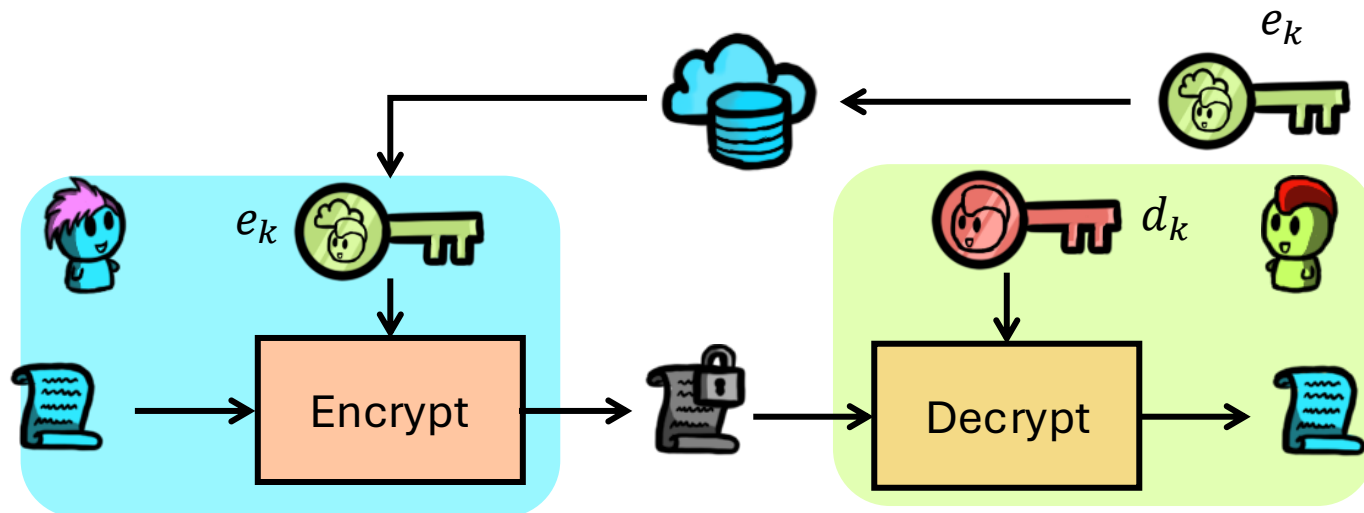
- Invented (in public) in the 1970's.
- We use a **different key** to encrypt and decrypt.
- The encryption key is public, and the decryption key is private



- Alice and Bob do **not** need to agree on a shared secret!
- Some common examples: RSA, ElGamal, ECC, NTRU, McEliece

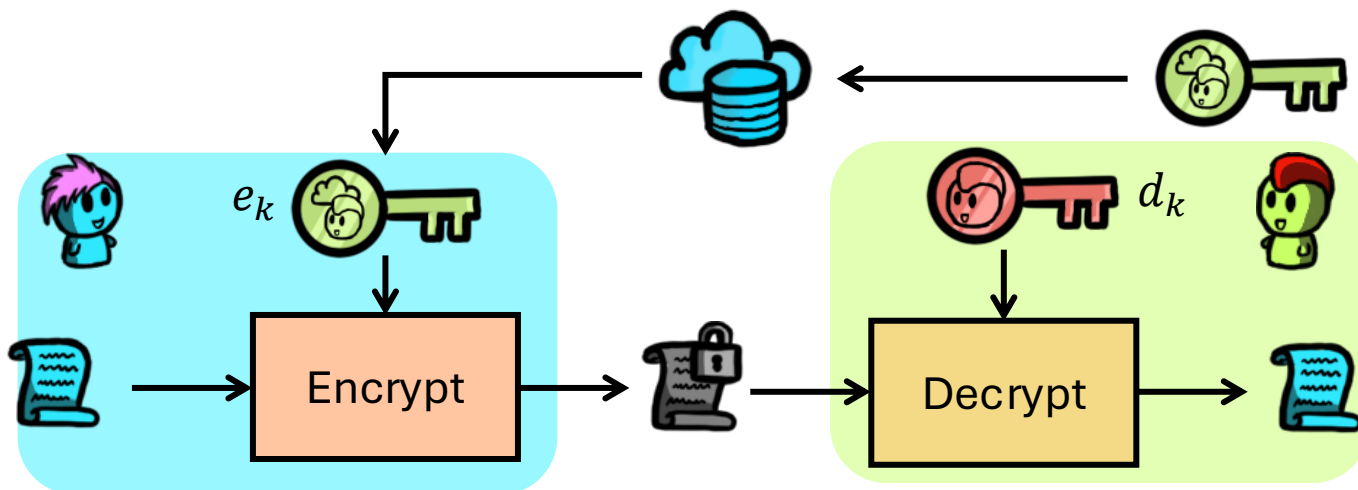
How does it work?

- Bob creates a **key pair** (e_k, d_k) .
 - It must be **hard** to derive d_k from e_k .
- Bob gives everyone a copy of his public encryption key e_k .
- Alice uses Bob's key to encrypt a message for Bob $C = E_{e_k}(M)$.
- Bob uses his private decryption key d_k to decrypt the message: $M = D_{d_k}(C)$.



How does it work?

- We want a mathematical operation that is very easy to do in one direction using e_k ...
- .. and very hard to “un-do” given e_k .
- But easy to “un-do” given d_k .
- These are called “**trapdoor functions**”



- We will see the most famous public-key crypto algorithm: RSA.
 - The simplest form of RSA, which is what we will see, is called Textbook RSA.
- Nowadays, Elliptic Curve Cryptography (ECC) is usually preferred (more efficient).
- Quantum computers break RSA and ECC: post-quantum algorithms are slowly getting adoption.



Primer on Modular Arithmetic

- This is what you need to know about modular arithmetic for textbook RSA. Some of you might know this already...
- Modular Arithmetic: integers that wrap around after reaching the “modulus”.
- Think of it like a clock: 1h, 2h, ... 11h, 12h, 1h, 2h...
- For example, if our modulus is n , then we write it as $a \bmod n$
- This means “the remainder when a is divided by n ”.

We use the symbol $\equiv$ when two numbers have the same remainder (i.e., they are “the same” number for this modulus).

Example: $17 \equiv 2 \pmod{5}$
We read it as “17 is congruent with 2 modulo 5”

What is
 $17 \bmod 5$
 $12 \bmod 10$
 $k \cdot n \bmod n$



In this course, we will sometimes use $=$ in place of $\equiv$ for simplicity, and if it is clear we will not even write “mod n ”.



Modular Arithmetic: addition

- You can add numbers before or after computing the “remainder” (taking the modulo operation):

$$a + b \pmod{n} = (a \pmod{n}) + (b \pmod{n})$$

Example:

$$170 + 30 \pmod{16} = 10 + 14 \pmod{16} = 8$$

$$170 + 30 \pmod{16} = 200 \pmod{16} = 8$$

- To compute a subtraction (the inverse of the addition), we just subtract, and if it is negative, we “add the modulus” until it is positive

Example:

$$-30 \pmod{16} = -30 + 32 \pmod{16} = 2$$



Modular Arithmetic: multiplication

- Like with the addition, you can compute the modulo operation of the numbers you are multiplying before or after doing the multiplication:

$$a \cdot b \pmod{n} = (a \pmod{n}) \cdot (b \pmod{n})$$

Example:

- $165 \cdot 250 \pmod{16}$
 - $= (165 \pmod{16}) \cdot (250 \pmod{16})$
 - $= 5 \cdot 10 \pmod{16}$
 - $= 50 \pmod{16} = 2$
- $165 \cdot 250 \pmod{16} = 41250 \pmod{16} = 2$

- The division (multiplicative inverse) is more complicated: we are working with integers!
- The multiplicative inverse of a modulo n does not always exist!



Modular Arithmetic: division (multiplicative inverse)

- The inverse of a modulo n only exists if the a and n are co-prime:
$$\gcd(a, n) = 1$$



Assume we are working modulo $n = 10$. Does the number 3 have a multiplicative inverse?



Which number (modulo n), when multiplied by 3, gives us a 1, i.e.,
$$3 \cdot b \equiv 1 \pmod{10}$$



Let's try it!

We multiply all the elements by 3:

$$\begin{aligned} 0 \cdot 3 &\equiv 0 \pmod{10} \\ 1 \cdot 3 &\equiv 3 \pmod{10} \\ 2 \cdot 3 &\equiv 6 \pmod{10} \\ 3 \cdot 3 &\equiv 9 \pmod{10} \\ 4 \cdot 3 &\equiv 2 \pmod{10} \\ 5 \cdot 3 &\equiv 5 \pmod{10} \\ 6 \cdot 3 &\equiv 8 \pmod{10} \\ 7 \cdot 3 &\equiv 1 \pmod{10} \\ 8 \cdot 3 &\equiv 4 \pmod{10} \\ 9 \cdot 3 &\equiv 7 \pmod{10} \end{aligned}$$



We multiply all the results by 7:

$$\begin{aligned} 0 \cdot 7 &\equiv 0 \pmod{10} \\ 3 \cdot 7 &\equiv 1 \pmod{10} \\ 6 \cdot 7 &\equiv 2 \pmod{10} \\ 9 \cdot 7 &\equiv 3 \pmod{10} \\ 2 \cdot 7 &\equiv 4 \pmod{10} \\ 5 \cdot 7 &\equiv 5 \pmod{10} \\ 8 \cdot 7 &\equiv 6 \pmod{10} \\ 1 \cdot 7 &\equiv 7 \pmod{10} \\ 4 \cdot 7 &\equiv 8 \pmod{10} \\ 7 \cdot 7 &\equiv 9 \pmod{10} \end{aligned}$$

Magic! We get the original numbers back!

This is because 7 is the multiplicative inverse of 3.

This means that “dividing by 3” is “multiplying by 7” if we work modulo 10.



Let's see a case that doesn't work

- What is the multiplicative inverse of 5 modulo 10?
- In other words, which number, when multiplied by 5, gives us 1 (mod 10)?
- This doesn't exist, since $\gcd(5,10) = 5 \neq 1$.

We multiply all the elements by 5:

$$\begin{aligned} 0 \cdot 5 &\equiv 0 \pmod{10} \\ 1 \cdot 5 &\equiv 5 \pmod{10} \\ 2 \cdot 5 &\equiv 0 \pmod{10} \\ 3 \cdot 5 &\equiv 5 \pmod{10} \\ 4 \cdot 5 &\equiv 0 \pmod{10} \\ 5 \cdot 5 &\equiv 5 \pmod{10} \\ 6 \cdot 5 &\equiv 0 \pmod{10} \\ 7 \cdot 5 &\equiv 5 \pmod{10} \\ 8 \cdot 5 &\equiv 0 \pmod{10} \\ 9 \cdot 5 &\equiv 5 \pmod{10} \end{aligned}$$

We obviously can't
recover the
original numbers...



Finding a multiplicative inverse

- A common way to find a multiplicative inverse of a number modulo n is to use the Extended Euclidean Algorithm

Extended Euclidean Algorithm:

Given integers a and b , find integers x and y such that:

$$a \cdot x + b \cdot y = \gcd(a, b)$$



How do we use this algorithm to find the multiplicative inverse of $a \bmod n$?

Hint: if we run the algorithm with a and n as inputs, we get x and y such that:

$$a \cdot x + n \cdot y = \gcd(a, n)$$

You don't need to know how it works exactly, but here is the pseudo-code (taken from [here](#))

Algorithm 3.4 Extended Euclidean algorithm

Input: Integers a and b

Output: Integers x , y , and d , where $d = \gcd(a, b) = ax + by$

```

Set  $d_0 = a$     Set  $x_0 = 1$     Set  $y_0 = 0$ 
Set  $d_1 = b$     Set  $x_1 = 0$     Set  $y_1 = 1$ 
While  $d_1 \neq 0$  Do
  Set  $q = \lfloor d_0 / d_1 \rfloor$ 
  Set  $d_2 = d_0 - qd_1$     Set  $x_2 = x_0 - qx_1$     Set  $y_2 = y_0 - qy_1$ 
  Set  $d_0 = d_1$     Set  $x_0 = x_1$     Set  $y_0 = y_1$ 
  Set  $d_1 = d_2$     Set  $x_1 = x_2$     Set  $y_1 = y_2$ 
End While
Return  $[d, x, y] = [d_0, x_0, y_0]$ 

```



Modular exponentiation

- **Warning!** If we have $a^b \pmod n$, we **cannot** reduce $b \bmod n$!

$$a^b \pmod n \neq a^{(b \bmod n)} \pmod n$$

- However, we have Fermat's little theorem: define $\varphi(n)$ as “the number of integers from 1 to n that are co-prime to n ”. Then,

$$a^b \pmod n = a^{(b \bmod \varphi(n))} \pmod n$$

You only need to know:

- If n is prime, then $\varphi(n) = n - 1$.
- If n is the product of two primes $n = p \cdot q$, then $\varphi(n) = (p - 1) \cdot (q - 1)$.

$$a^b \equiv a^{(b \bmod \varphi(n))} \pmod{n}$$

If $n = p \cdot q$ (p and q are prime numbers),
then $\varphi(n) = (p - 1) \cdot (q - 1)$

Examples of exponentiation

- We choose two primes, $p = 7$ and $q = 3$, and $n = p \cdot q = 21$.
- The ~~reduced totient function~~ is $\varphi(21) = (7 - 1)(3 - 1) = 12$.
- Let's see this in action. We pick $a = 2$, and start raising it to different powers:

```
for i in range(21):
    print(f"2^{i} mod 21 = {2**i % 21}")
```

This makes sense: $a^b = a^{b+12}$
(in fact, the period is 6 in this
case, but 12 is also a period).



$2^0 \bmod 21 = 1$	$2^6 \bmod 21 = 1$	$2^{12} \bmod 21 = 1$	$2^{18} \bmod 21 = 1$
$2^1 \bmod 21 = 2$	$2^7 \bmod 21 = 2$	$2^{13} \bmod 21 = 2$	$2^{19} \bmod 21 = 2$
$2^2 \bmod 21 = 4$	$2^8 \bmod 21 = 4$	$2^{14} \bmod 21 = 4$	$2^{20} \bmod 21 = 4$
$2^3 \bmod 21 = 8$	$2^9 \bmod 21 = 8$	$2^{15} \bmod 21 = 8$	...
$2^4 \bmod 21 = 16$	$2^{10} \bmod 21 = 16$	$2^{16} \bmod 21 = 16$	
$2^5 \bmod 21 = 11$	$2^{11} \bmod 21 = 11$	$2^{17} \bmod 21 = 11$	



Final recap

- $a \bmod n$ means “the remainder of a divided by n ”.
- The numbers “mod n ” are $0, 1, 2, \dots, n - 1$.
- Additions, subtractions, and multiplications works as we are used to. We can apply the modulo reductions before and after the operations.
- **Divisions are especial:**
 - We do not “divide by a ”, but multiply by the inverse (a^{-1}).
 - The inverse of a only exists if $\gcd(a, n) = 1$.
 - If the inverse exists, it is very easy to compute given n with the Extended Euclidean Alg.
- **Exponentiations are also especial**, in the sense that we cannot reduce exponents modulo n : we reduce the exponents with $\varphi(n)$.
 - If $n = p \cdot q$ for p, q prime, then $\varphi(n) = (p - 1) \cdot (q - 1)$.
 - This means that $a^b \equiv a^{b \bmod \varphi(n)} \pmod{n}$.

Textbook RSA

- RSA was the first popular public-key encryption method (published 1977).
- It relies on the practical difficulty of the factoring problem:
 - Given the product of two large prime numbers $n = p \cdot q$, it is very hard to factor n .
 - We work with modular arithmetic.
- High-level idea:
 - It is easy to find large integers e , d , and n such that:

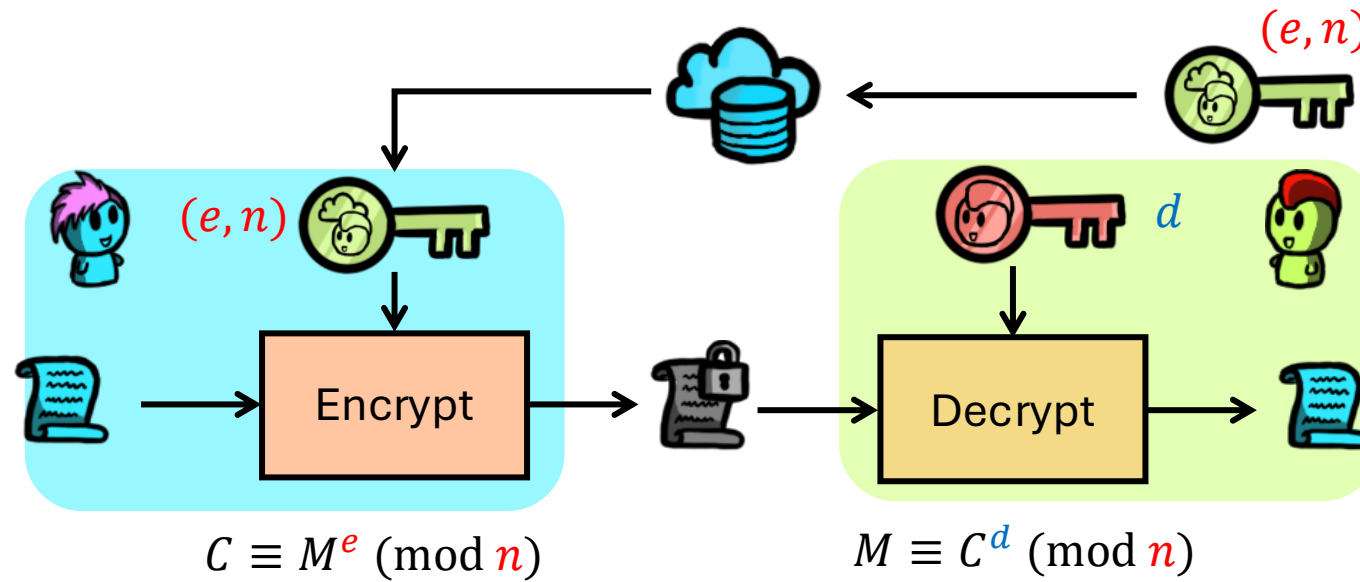
$$(M^e)^d \equiv M \pmod{n}$$

- But knowing e and n (and even M), it is extremely hard to find d .

Textbook RSA (simplified)

- Choose two large primes p and q (these are secret)
- Compute $n = p \cdot q$.
- Compute $\varphi(n) = (p - 1)(q - 1)$.
- Choose a number e such that $\gcd(e, \varphi(n)) = 1$.
- Determine d such that $d \equiv e^{-1} \pmod{\varphi(n)}$
- **Public key:** (e, n)
- **Private key** d (the other numbers can be discarded).
- Encryption of plaintext M : $C \equiv M^e \pmod{n}$.
- Decryption of ciphertext C : $M \equiv C^d \pmod{n}$.
- **Note:** the decryption indeed works.
$$(M^e)^d \equiv M \pmod{n}$$
- **Note:** factoring n breaks this! Why?
- This is textbook RSA, never do this!! (we'll see one of the reasons next).

Textbook RSA in a nutshell





Example of Textbook RSA

Textbook RSA cheat sheet

- Choose primes p and q .
- Compute $n = p \cdot q$.
- Choose e such that $\gcd(e, \varphi(n)) = 1$.
- Determine d such that $d \equiv e^{-1} \pmod{\varphi(n)}$
- **Public key:** (e, n)
- **Private key** d
- Encryption:
 $C \equiv M^e \pmod{n}$.
- Decryption:
 $M \equiv C^d \pmod{n}$.

Assume $p = 53$, $q = 101$, $e = 139$, $d = 1459$.

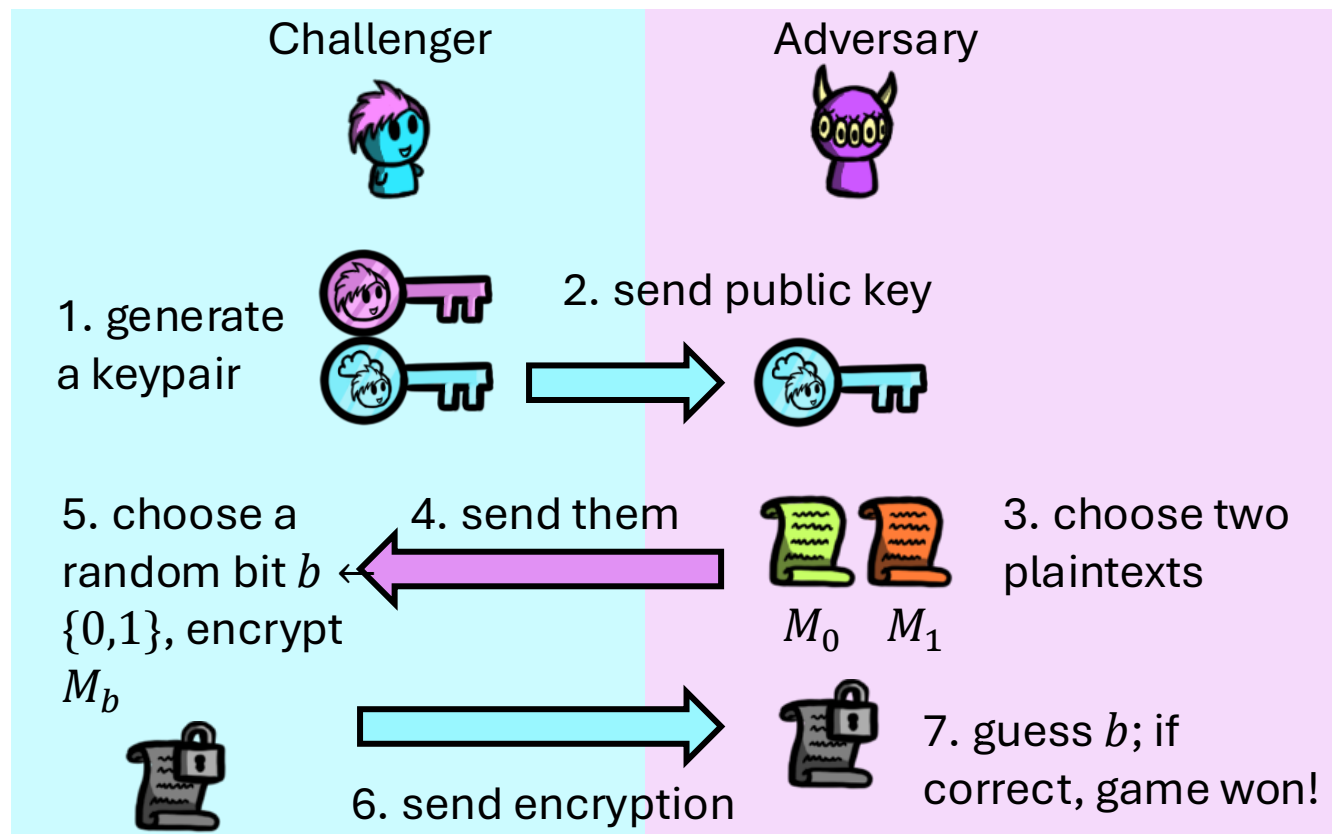
1. Compute n .
2. Compute $C_1 = E_e(1011)$. Verify decryption works.
3. Compute $C_2 = E_e(4)$. Verify decryption works.
4. Compute $D_d(C_1 \cdot C_2)$. What is happening? Why?

Malleability: we can transform a ciphertext into another ciphertext that decrypts to a “related” plaintext.

This is typically (but not always!) undesirable!

IND-CPA Game

- IND-CPA: Indistinguishability under chosen-plaintext attack
- An encryption scheme provides IND-CPA if the adversary cannot win the following game with probability larger than 0.5.



In textbook RSA: can the adversary win this game with probability higher than 0.5?





Chosen Ciphertext Attack in textbook RSA

Textbook RSA cheat sheet

- Choose primes p and q .
- Compute $n = p \cdot q$.
- Choose e such that $\gcd(e, \varphi(n)) = 1$.
- Determine d such that $d \equiv e^{-1} \pmod{\varphi(n)}$
- **Public key:** (e, n)
- **Private key** d
- Encryption:
 $C \equiv M^e \pmod{n}$.
- Decryption:
 $M \equiv C^d \pmod{n}$.

- We are Eve. Alice is using RSA and her public key is (e, n) .
- Bob sends a super-secret message M , encrypted as $C = E_e(M)$. We intercept C .
- Alice is convinced her textbook RSA is very secure, so she is willing to **decrypt any ciphertext** we send her except for C , and send us the decryption back.
- How can we guess M ?

Shortcomings of textbook RSA

- We saw that textbook RSA is not IND-CPA secure.
 - It is also vulnerable to chosen ciphertext attacks.
 - Choosing a very small e (for example $e = 3$) can lead to cases where $M^e < n$, so that computing a regular cubic root yields M .
 - There are other issues...
-
- That's why, in actual implementations of RSA, we use padding techniques (OAEP) to pre-process the plaintext before encryption:
 - This makes the encryption randomized every time (IND-CPA secure).
 - It also prevents the chosen ciphertext attack from the previous slide (making RSA non-malleable).

Public key sizes

- Recall that, without shortcuts, Eve would have to try all 2^{128} keys in order to read a message encrypted with a 128-bit key.
- Unfortunately, all of the public-key methods we know **do have shortcuts**.
- 128-bit RSA:
 - Option 1: try every possible key. This takes 2^{128} work, which is a lot!
 - Option 2: try to factor n . This takes just 2^{33} work, which is easy!
 - If we want Eve to have to do 2^{128} work, we need to use a much longer public key...

Comparison of key sizes

- This is a comparison of key sizes for roughly equal strength.

AES	RSA	ECC
80	1024	160
116	2048	232
128	2600	256
160	4500	320
256	14000	512

Takeaways:

- Symmetric crypto provides more security for equal key sizes than public-key crypto.
- For equal strength, symmetric crypto needs smaller keys, and it's also faster and requires less bandwidth.

- Quantum computers can break RSA/ECC and others way faster than this (e.g., see [Shor's algorithm to break RSA](#)).

Hybrid cryptography

- **Secret-key crypto**: shorter keys, faster, same key to encrypt and decrypt.
- **Public-key crypto**: longer keys, slower, different key to encrypt and decrypt.
- But public-key cryptography is very convenient (no shared secret).
- **Hybrid crypto**: get the best of both worlds
 - Pick a random 128-bit key K for a secret-key cryptosystem.
 - Encrypt the (possibly large) message M with the key K (e.g., using AES).
 - Encrypt the key K using a public-key cryptosystem.
 - Send the encrypted message and the encrypted key to Bob.
- This (or a similar) hybrid approach is used for almost every cryptography application on the Internet today.



Knowledge check!



Public-key params: (e_A, d_A)
Secret-key params: K



Public keys:

e_A, e_B

Public-key params:
 (e_B, d_B)
Secret-key params: ?



- Encrypt/decrypt functions: $E_{key}(\cdot), D_{key}(\cdot)$.
- Concatenation: $||$
- Alice wants to send a **very large** message M to Bob.



How does Alice
build the message?



How does Bob recover
the message?

Is that all?

- We know how to send secret messages, and Eve cannot do anything about it.
- However, Mallory could modify our encrypted messages in transit!
- Mallory won't necessarily know what the message says, but can still change it in an undetectable way
 - e.g., the bit-flipping attacks on stream ciphers
- This is counterintuitive, but often forgotten...
- How do we make sure that Bob gets the same message that Alice sent?

CPEN 442 – Introduction to Cybersecurity

Module 2 – Cryptography



Part 5 – Integrity and Authentication

Part 1:
Preamble

Part 2:
Ancient Crypto

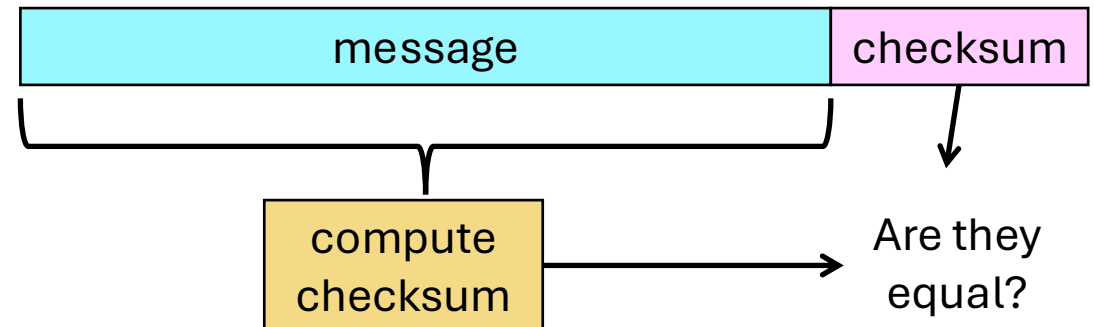
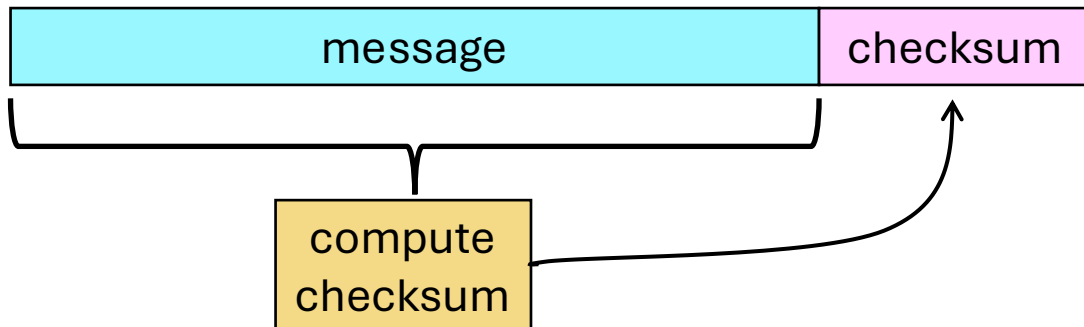
Part 3:
Symmetric
Crypto

Part 4:
Public-key
Crypto

- Cryptographic hash functions
- Authentication
 - Message Authentication Codes (MACs)
 - Digital Signatures
- Repudiation
- Key management and CAs

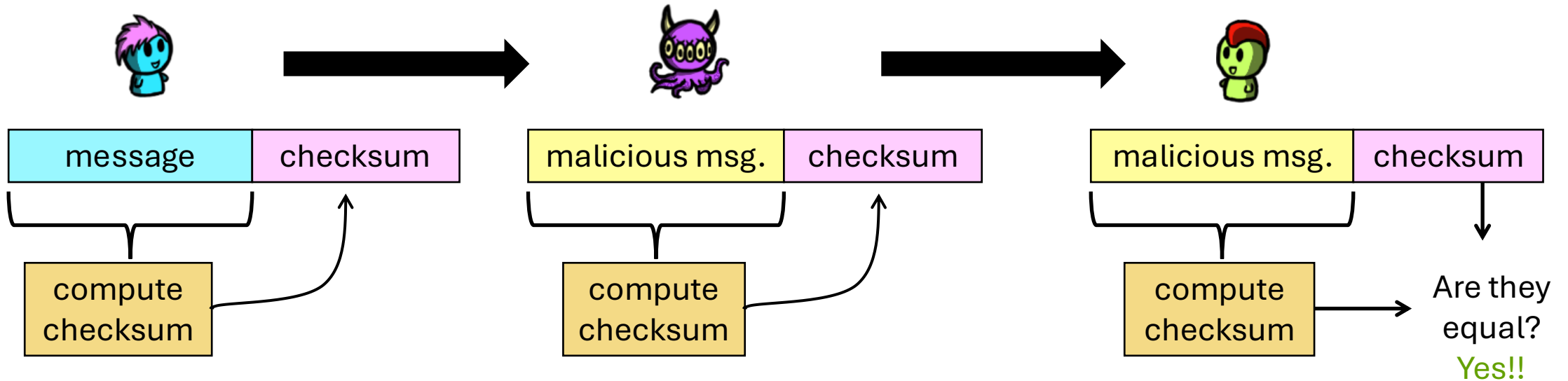
Integrity in non-malicious settings

- How do we tell if a message has changed in transit?
- Simplest answer: use a checksum
 - For example, add up all the bytes of a message.
 - The last digits of serial numbers (credit card, ISBN, etc.) are usually checksums.



Integrity in malicious settings

- Checksum does not work!

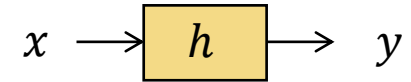


- We need something else...

If the message || checksum was encrypted with a stream cipher: can Mallory still do this?

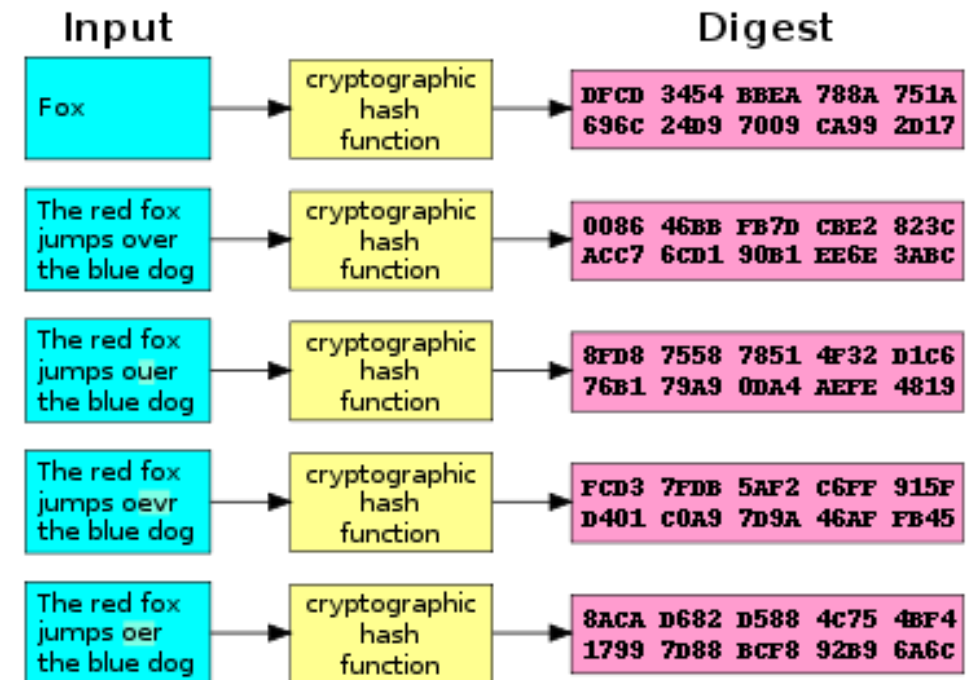


Cryptographic Hash Functions



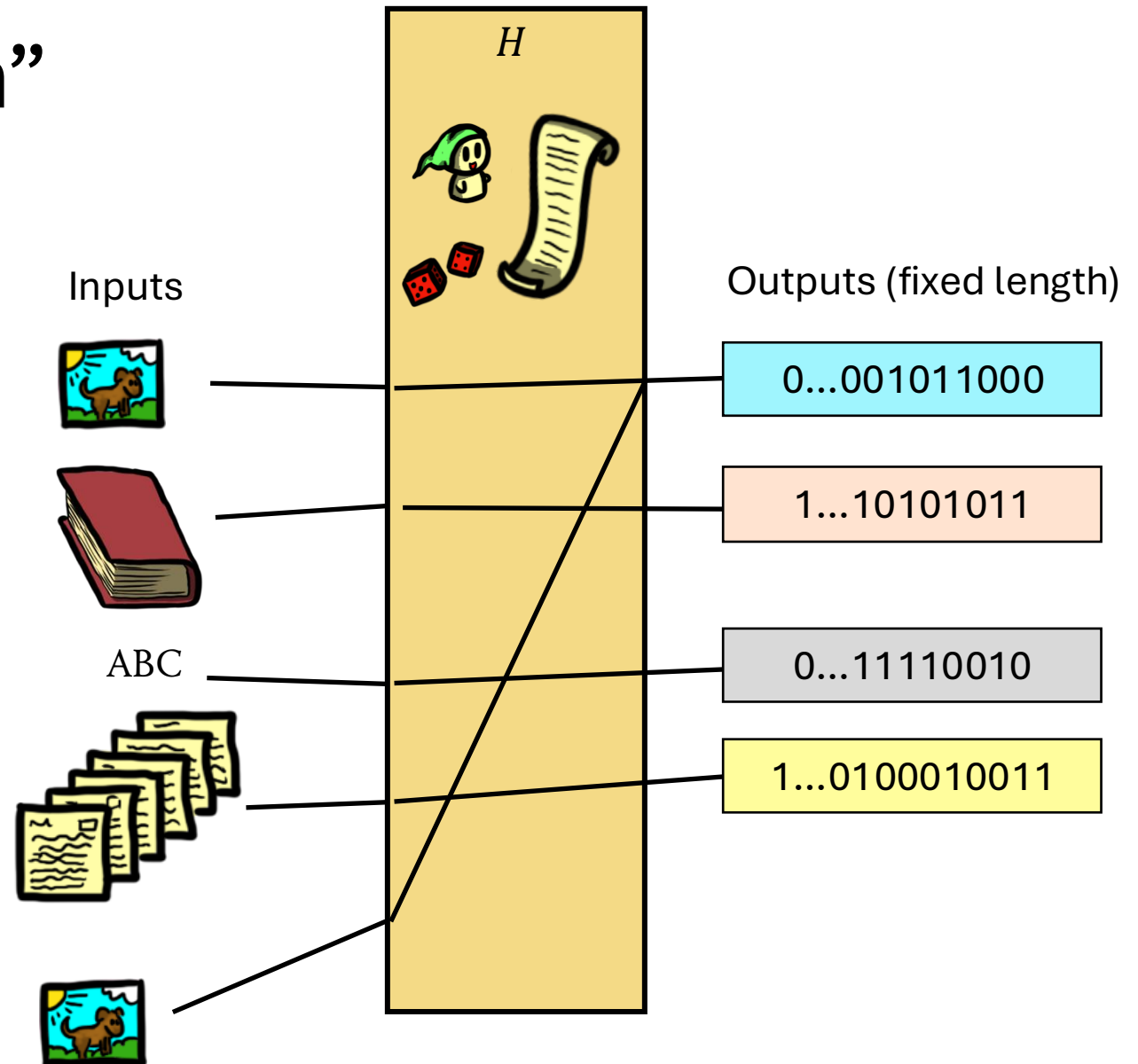
- A **hash function** h takes an arbitrary length string x and computes a fixed-length string $y = h(x)$ called a **message digest** (or hash, or fingerprint, or tag).
- Common examples:
 - MD5
 - SHA-1
 - SHA-2
 - SHA-256
 - SHA-512
 - SHA-3 (AKA Keccak, from 2012 on)
 - ...

Changing the input should re-randomize the output



The ideal “Hash function”

- The *ideal* hash function takes an input of any size and produces a fixed-size output in a way that behaves like a **random function** while being deterministic.
- The ideal hash function is modelled as a **random oracle**. This is a black box with:
 - An infinite **memory** (think of it as a dictionary mapping inputs to outputs)
 - A pure source of **randomness**.
- If you give it an input:
 - If that input is in the dictionary, grab the output from the dictionary and return that.
 - If it is not, then generate an output at random, add the input and output to the dictionary, and return the output.





Cryptographic hash functions: properties

Cryptographic hash functions should have three properties:

1. Preimage resistance:

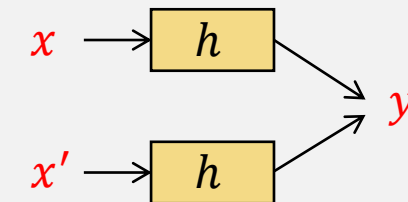
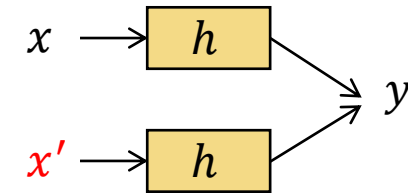
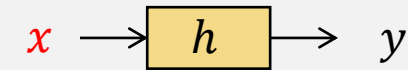
Given y , it is hard to **find** x such that $h(x) = y$ (i.e., it is hard to find a “preimage” of $h(x)$)

2. Second preimage resistance:

Given x , it is hard to **find** $x' \neq x$ such that $h(x) = h(x')$ (i.e., a “second preimage” of $h(x)$). Note that x is fixed, and we have to find x' .

3. Collision resistance:

It is hard to **find** any two distinct values x, x' such that $h(x) = h(x')$ (i.e., a “collision”). Note that we have free choice of x and x' .



h has a property if it is hard for the adversary to find the thing in **red** (given the things in black).



Easiest-to-hardest properties to break?
Weakest-to-strongest properties?

It must be “hard”

- We said that it must be “hard” for the adversary to break those properties?
- What is “hard”?
- For SHA-1, for example, it takes 2^{160} work to find a preimage or second preimage, and 2^{80} work to find a collision using a brute-force search.
 - However, there are faster ways than brute force search to find collisions in SHA-1 or MD5.
- Collisions are always easier to find than preimages or second preimages due to the well-known birthday paradox.



The birthday paradox

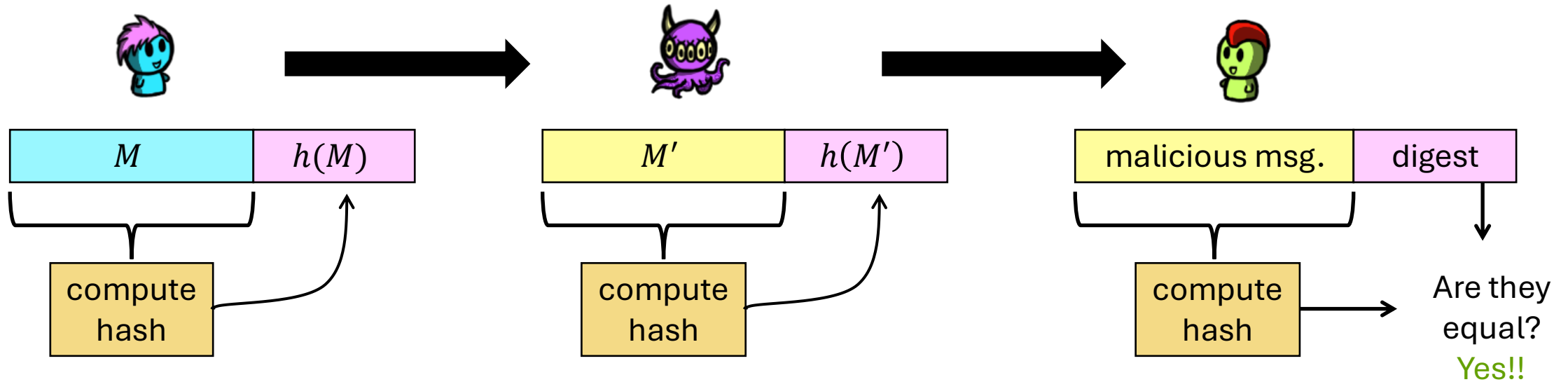
If there are n people in a room, what is the probability that at least two people have the same birthday?

**assuming the birthday distribution is uniform and we're all iid samples*

- For 23 people, the probability is larger than 50%
- For 40 people, it's almost 90%
- For 50 people, around 97%
- For 60 people, more than 99%



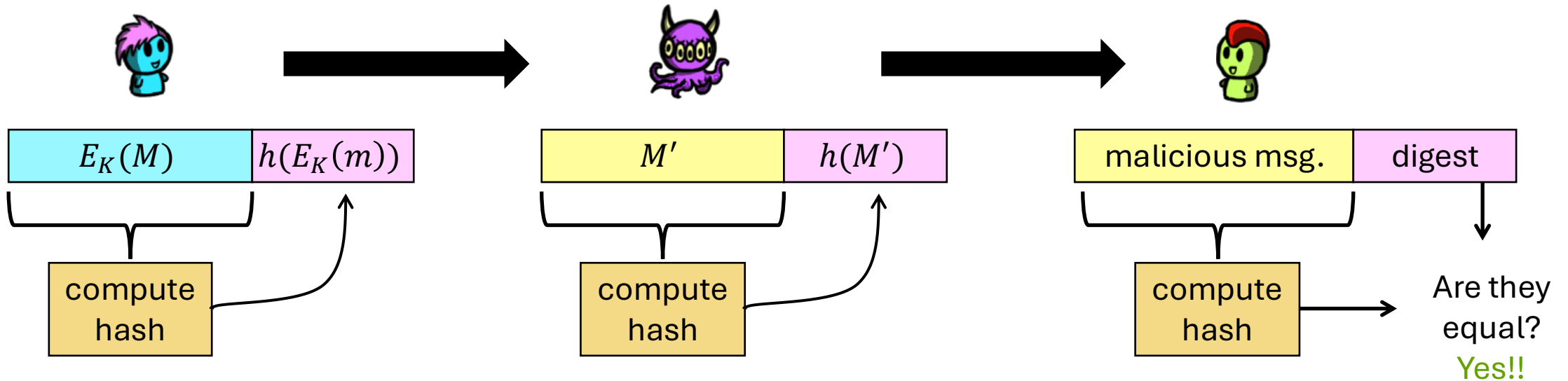
Let's use a hash function!



- Mallory can break integrity...



Let's use a hash function with encryption!



- Mallory can still break integrity!
 - The integrity should not depend on the decrypted message “making sense”.
 - The integrity check should work even if M was a message of randomly sampled bits; we still want Alice to be able to send that to Bob.

Cryptographic hash functions

- Hash functions provide integrity guarantees only when there is a **secure way** of sending and/or storing the message digest.
 - For example, Bob can publish a hash of his public key (i.e., a message digest) on his business card.
 - Putting the whole key on there would be too big.
 - But Alice can download Bob's key from the Internet, hash it herself, and verify that the result matches the message digest on Bob's card.
- What if there is no external channel to do this?
 - For example, you're using the Internet to communicate...
 - We can use "keyed hash functions"!
 - We will see these next

Authentication

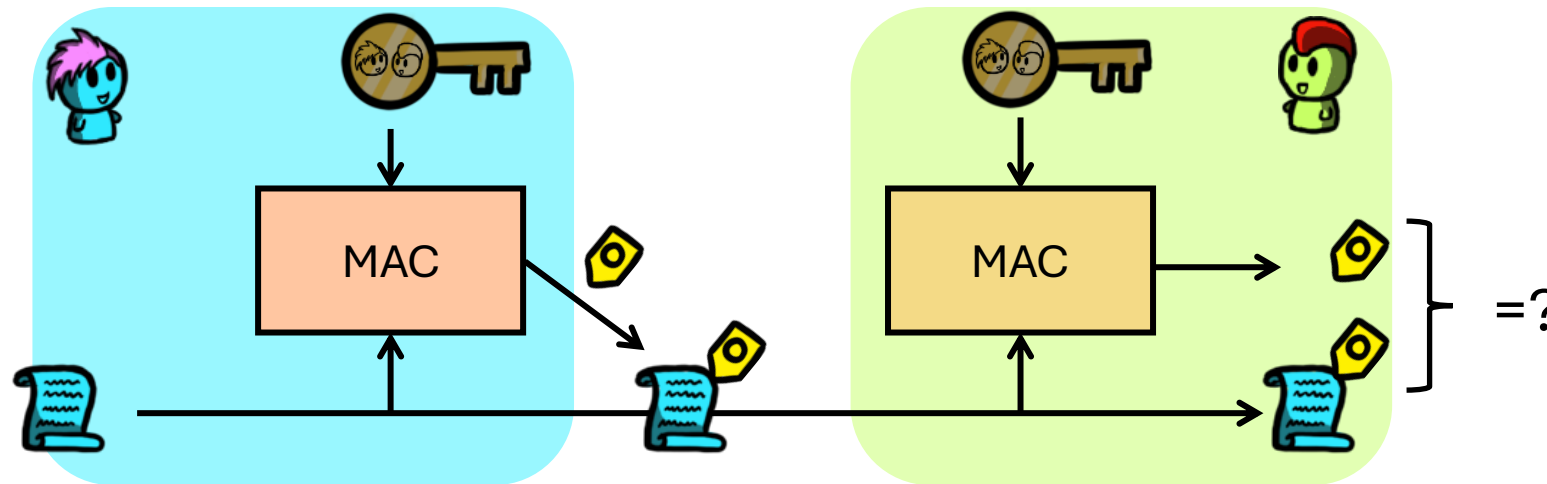
- **Authentication** and **integrity** go together in cryptography: we either provide both, or none.
 - If we cannot verify the “authenticity” of a sender, then that sender could be Mallory, and she could have modified the plaintext message...
- Authentication/integrity should not rely on how “reasonable” the decrypted plaintext M looks.
 - We need a separate tool to decide whether a message comes from the intended sender, and whether the message has been modified in transit or not.
 - The authentication check should work even when M is a totally random message.
- Two main ways of providing authentication:
 - MACs (Message Authentication Codes)
 - Digital Signatures

Message Authentication Codes (MACs)

Common examples of MACs:

- SHA-1-HMAC
- SHA-256-HMAC
- CBC-MAC

- **MAC**s are basically “keyed hash functions”.
- Only those who know the secret key can generate, or even check, the computed hash value (sometimes called a **tag**).



- Can Mallory forge a message with a valid tag? What does she need to do so?

HMAC

- HMAC is a way of building a MAC from a hash function.
- Let's use $\text{HMAC}_K(M)$ to denote the HMAC with key K for message M .
- What HMAC does (simplified):

$$\text{HMAC}_K(M) = h(K \parallel h(K \parallel M))$$

- The “double hashing” is needed to prevent some “length extension attacks” that are out-of-scope for this course.

Combining ciphers and MACs

- In practice, we often need **both** confidentiality and message integrity.
- There are multiple strategies to combine a cipher and a MAC when processing a message.
 - MAC-then-Encrypt
 - Encrypt-and-MAC
 - Encrypt-then-MAC
- Ideally, your crypto library already provides an **authenticated encryption mode** that securely combines the two operations, so you don't have to worry about getting it right.
 - Some examples:
 - GCM
 - CCM (used in WPA2, see later)
 - OCB mode



Combining ciphers and MACs, let's try it!

Alice and Bob have a secret key K for a secret-key cryptosystem $(E_K(\cdot), D_K(\cdot))$, and a secret key K' for their MAC $(MAC_{K'}(\cdot))$. Concatenation is $\parallel$. How does Alice build a message for Bob in the following scenarios?

- **MAC-then-Encrypt**: compute the MAC on the message, then encrypt the message and MAC together, and send that ciphertext.

$$E_K(M \parallel MAC_{K'}(M))$$

- **Encrypt-and-MAC**: compute the MAC on the message, compute the encryption on the message, and send both.

$$E_K(M) \parallel MAC_{K'}(M)$$

- **Encrypt-then-MAC**: encrypt the message, compute the MAC on the encryption, send encrypted message and MAC.

$$E_K(M) \parallel MAC_{K'}(E_K(M))$$



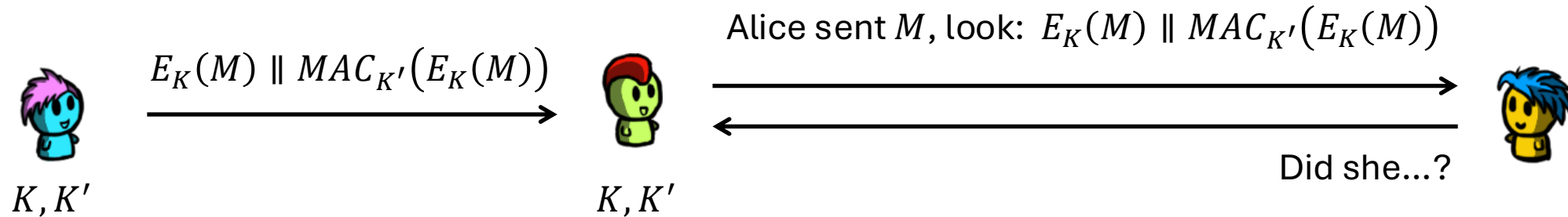
What is the right order?

- Usually, we want the receiver to **verify the MAC first**. What is the recommended strategy, then?
- The recommended strategy is **usually** Encrypt-then-MAC:

$$E_K(M) \parallel MAC_{K'}(E_K(M))$$

- There is a [nice blog post](#) that calls this the “Doom principle”: if you have to perform *any* cryptographic operation before verifying the MAC on a message you’ve received, it will *somehow* lead inevitably to doom.
 - It explains two simple attacks that can happen if the Doom principle is violated.
- However, the others might have some uses... (we might see some examples of what can go wrong with this)

Repudiation

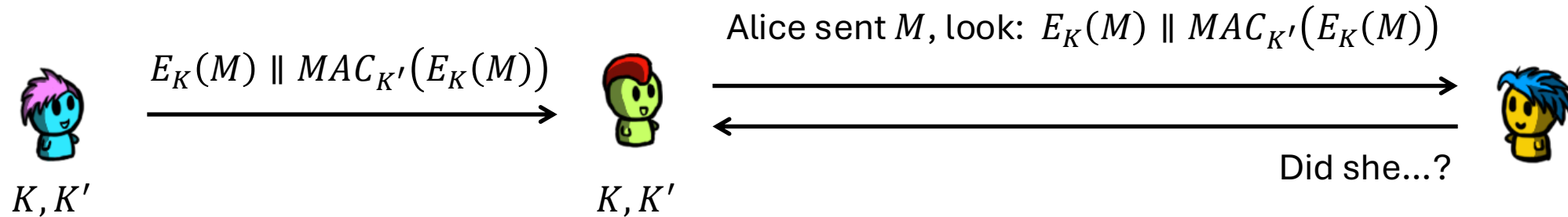


- Bob can be assured that Alice is the one who sent M and that the message has not been modified since she sent it.
- We have confidentiality, integrity, and authentication!
- This is like a “signature” on the message... but not quite the same!
- Bob **cannot** prove to Carol that Alice sent M : this is called **repudiation**.

Why not?



Repudiation

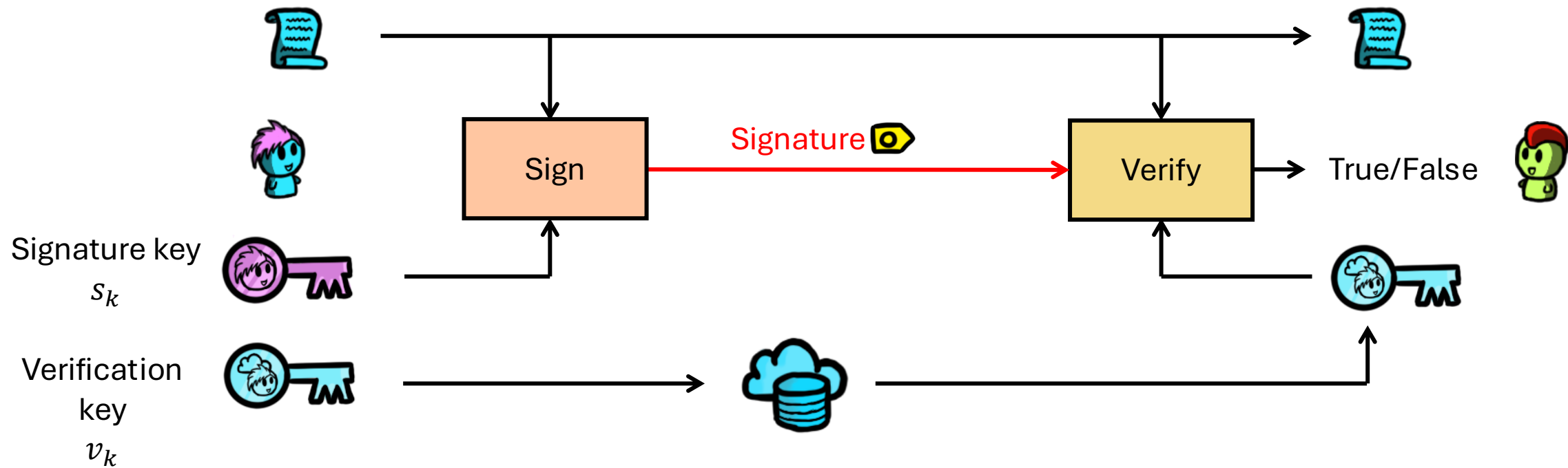


- Alice can just claim that Bob made up the message M , and calculated the MAC himself.
- This is called **repudiation**, and we sometimes want to avoid it.
- Some interactions should be repudiable.
 - Private conversations
- Some interactions should be non-repudiable.
 - Electronic commerce

Digital signatures

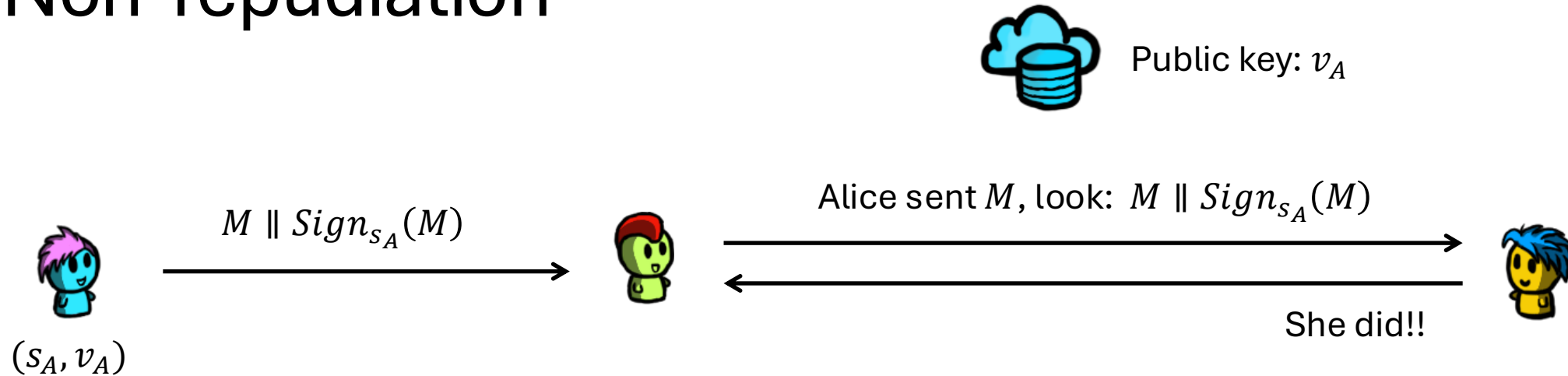
- MACs are the “**symmetric**” version of authentication.
- Digital signatures are the “**asymmetric**” version of authentication.
- Remember, to encrypt/decrypt in public-key cryptography
 - Alice *encrypts* with Bob’s *public encryption* key.
 - Bob *decrypts* with his *private decryption* key.
- To sign and verify in public-key cryptography:
 - Alice *signs* with her *private signature* key.
 - Bob *verifies* the message with Alice’s *public verification* key.

Digital Signatures



- If you need encryption, you also need to “do” encryption.

Non-repudiation

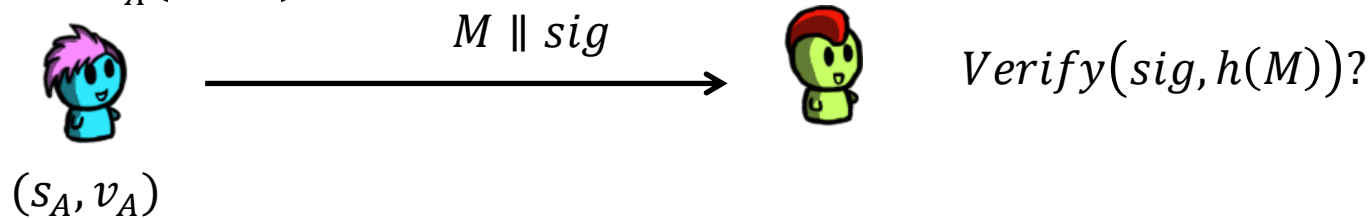


- Digital signatures provide **non-repudiation**.
- If Bob receives a message with Alice's *digital signature* on it, then:
 - Alice, and not an impersonator, sent the message (like a MAC).
 - The message has not been altered since it was sent (like a MAC).
 - Bob can prove these facts to a third party (additional property not satisfied by a MAC).

Faster signatures

- Just like encryption in public-key crypto, signing large messages is slow (MACs are faster!).
- We can also “hybridize” signatures to make them faster:
 - Alice sends the (unsigned) message, and also a **signature on a hash** of the message.
 - The hash is much smaller than the message, and so it is faster to sign and verify.

$$sig = \text{Sign}_{s_A}(h(M))$$



- We *always* want to hash before signing!
- From now on, when we write $\text{Sign}(x)$, we are first hashing x , and then signing it!



Knowledge check II:

Alice wants to send a **large** message M to Bob. She wants CIA and **non-repudiation**, and we want Bob to verify the integrity/authentication **first**.

What does she send?



What does Bob do after receiving the message?



Public keys: e_A, e_B, v_A, v_B



(e_A, d_A)

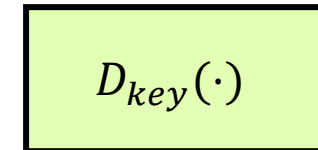
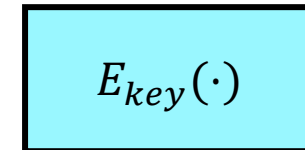
(s_A, v_A)



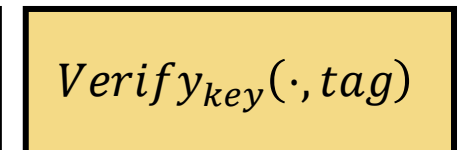
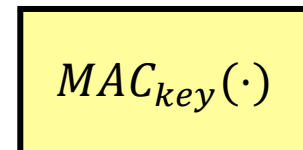
(e_B, d_B)

(s_B, v_B)

Encryption/decryption functions (symmetric or public-key)



Authentication functions:



Assume these two hash before sign/verify

Relationship between key pairs

- Usually, Alice's (signature, verification) key pair is long-lived, whereas her (encryption, decryption) key pair is short-lived.
 - Gives forward secrecy (see later).
- When creating a new (encryption, decryption) key pair, Alice uses her signing key to sign her new encryption key and Bob Alice's verification key to verify the signature on this new key.

The key management problem



- How can Alice and Bob be sure they're talking to each other, and not Mallory?
 - By having each other's verification key!
- Finding this verification key is a very hard problem.
- Some possible solutions for Bob to get Alice's verification key:
 - He can know it **personally** (manual keying).
 - SSH does this.
 - He can trust a **friend** to tell him (web of trust).
 - PGP does this.
 - He can trust some **third party** to tell him (CAs).
 - TLS/SSL does this.

We will see how to solve the key management problem this in a couple of weeks...

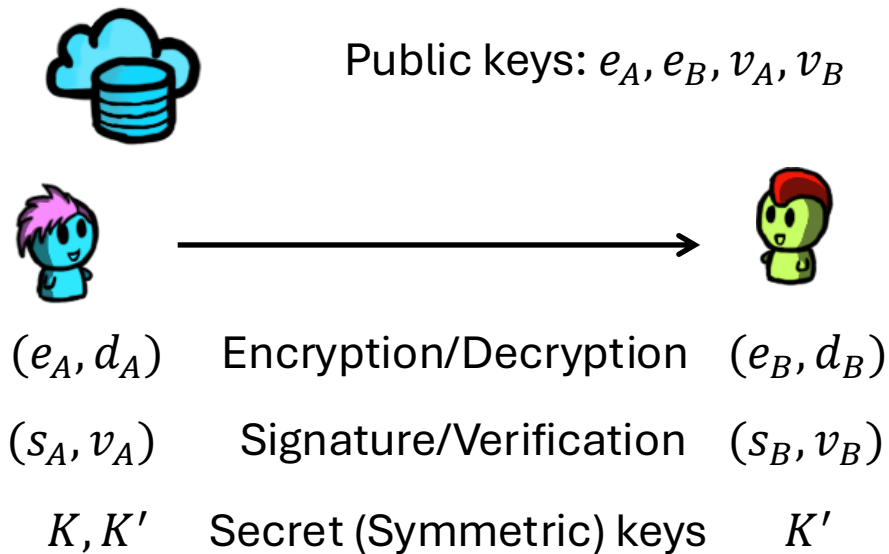
Putting it all together

- We have all these blocks: now what?
- We put them together to build **protocols**.
- This is HARD. Just because the pieces work, it does not mean that building something with them will. You have to use the pieces correctly.
- Common mistakes include:
 - Using the same stream cipher keystream for two messages.
 - Assuming encryption also provides integrity.
 - Falling for replay attacks (adversary sends a previously-captured packet).
 - Falling for reaction attacks (adversary observes someone's reaction to a packet).
 - E.g., did it trigger a decryption error? an integrity error?
 - LOTS more!

Exam question: Message formats

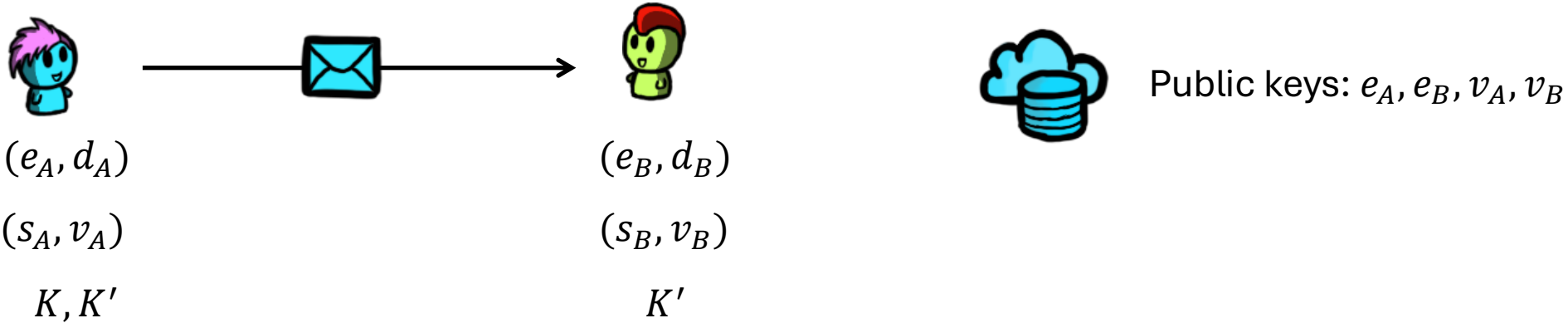
Alice and Bob agree to use a specific message format to send the plaintext content M to the other party. (The message format is public knowledge.)

This is the key information they have:



Alice sends a message to Bob.

- Can Bob read the plaintext?
 - If yes:
 - Does the message provide confidentiality?
 - Does the message provide integrity/authentication?
 - If yes:
 - Does the message provide repudiation?
 - Can Bob verify the authenticity before decrypting?

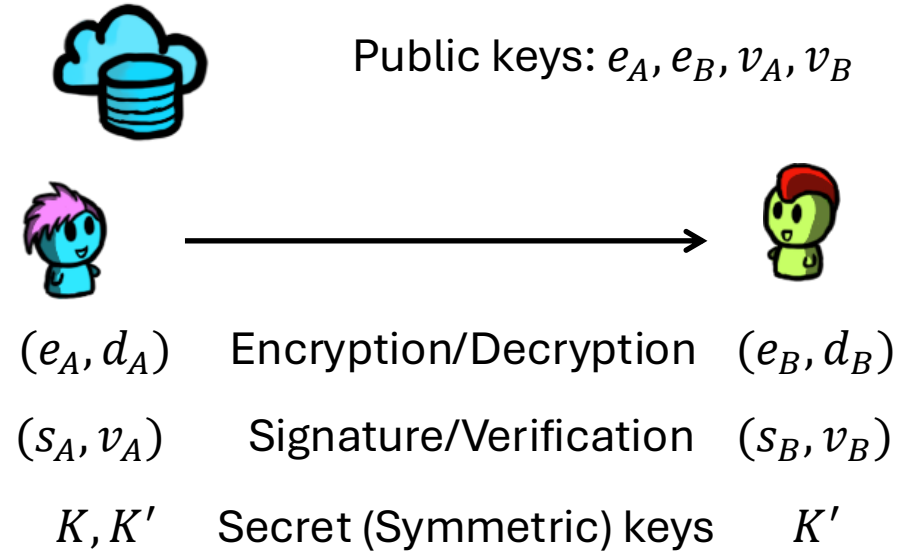


Message	Read M ?	Conf?	Auth?	Repudiation?	Verify first?
$E_{e_A}(M) \parallel \text{Sign}_{s_A}(M)$					
$E_K(M) \parallel E_{e_B}(K)$					
$E_K(M) \parallel E_{e_B}(K) \parallel \text{MAC}_K(M)$					
$E_K(M) \parallel K \parallel \text{MAC}_K(M)$					
$E_{K'}(M) \parallel \text{MAC}_{K'}(M)$					
$E_{e_B}(M) \parallel \text{Sign}_{s_A}(E_{e_B}(M))$					
$E_{e_B}(h(M))$					
$M \parallel \text{Sign}_{s_A}(M)$					
$M \parallel E_{e_B}(K) \parallel \text{MAC}_K(M)$					

Question variations

There will be a message format question in the midterm (this knowledge will also be evaluated in the final, almost surely). There are many variations:

- Assess the crypto properties of a message format.
- Propose a fix to some property (e.g., how can you modify the message format to provide repudiation, under some restrictions...)
- Create a message format that gives certain properties (see below).



Message	Decrypt?	Conf?	Auth?	Repudiation?	Verify first?
	Yes	Yes	No	-	-
	Yes	Yes	Yes	Yes	No
	Yes	Yes	Yes	Yes	Yes
	Yes	Yes	Yes	No	Yes

- I will upload the related exercises as practice to Canvas.