CPEN 442 – Introduction to Cybersecurity

Module 2

Cryptography

This material in these slides is largely taken from the "CS458: Computer Security and Privacy" course at the University of Waterloo, and it has been originally designed by Profs. Ian Goldberg and Urs Hengartner, with contributions of other instructors.

Module Outline

Preamble

- Cryptology
- Basic building blocks
- Kerckhoff's principle
- Cryptography attack models
- Confusion and Diffusion
- Symmetric vs Asymmetric crypto

Part 1: Ancient Crypto

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

Part 2: Symmetric crypto

- Definition
- Computational security
- Stream ciphers
  - Examples (A5/1, Salsa/Chacha)
- Block ciphers
  - Modes of operation
  - AES

Part 3: Public-key crypto

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

Part 4: Integrity and Authentication

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

CPEN 442 – Introduction to Cybersecurity

Module 2 – Cryptography

Part 1 – Introduction

Cryptology

- 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.
- The goal of these lectures is to show you what cryptographic tools exist, and information about using these tools in a secure manner.
  - We won't be seeing the details of how the tools work.

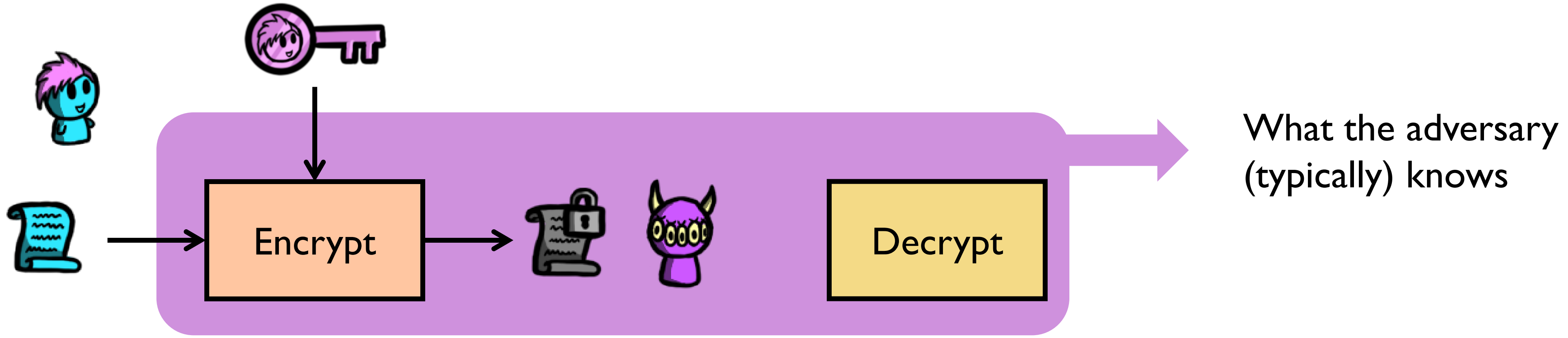
The building blocks of cryptography

- These are the main properties that we want to achieve with cryptography:
  - 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.).

Kerckhoff's principle

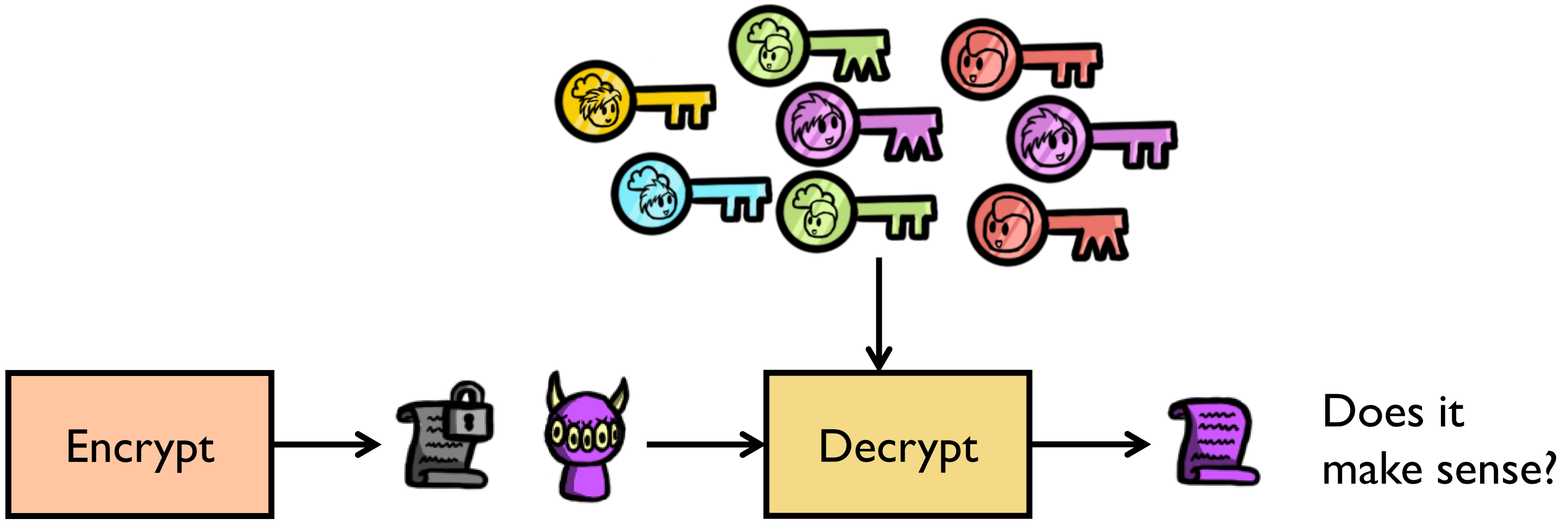
- **Kerckhoff's principle**: a cryptosystem should be secure even if everything about the system, except the key, is public knowledge.
- **Shannon's maxim**: one ought to design systems under the assumption that the enemy will immediately gain full familiarity with them.
- Do not use "secret" encryption 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:

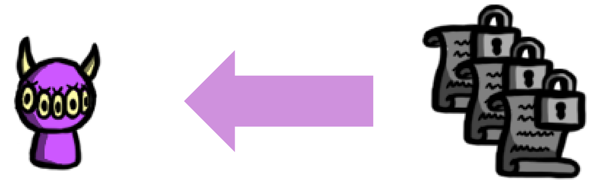
- The system is at most as secure as the total number of possible keys.
- Eve can try every possible key, until she finds the right one.
- Many times, there are shortcuts to finding the key
    - We will see some examples later, in the newspaper cryptogram



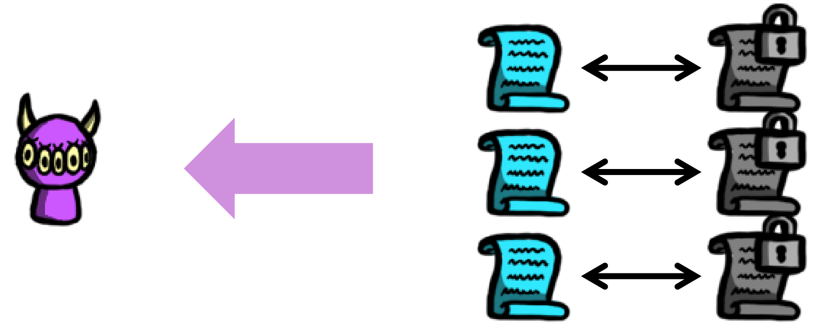
Cryptography Attack Models

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.



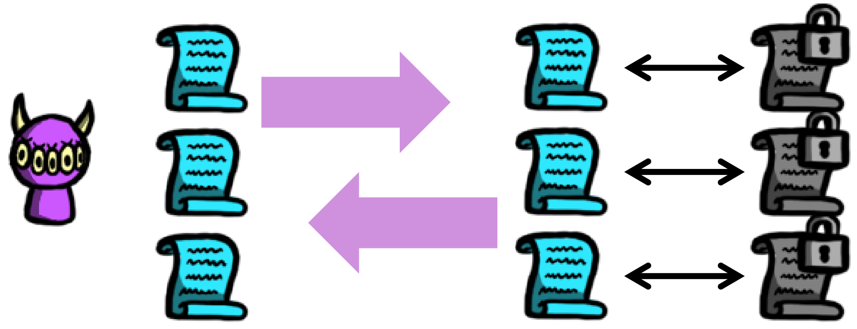
- Know plaintext attack: Eve knows one (or many) plaintext and ciphertext pairs. She tries to guess the key.



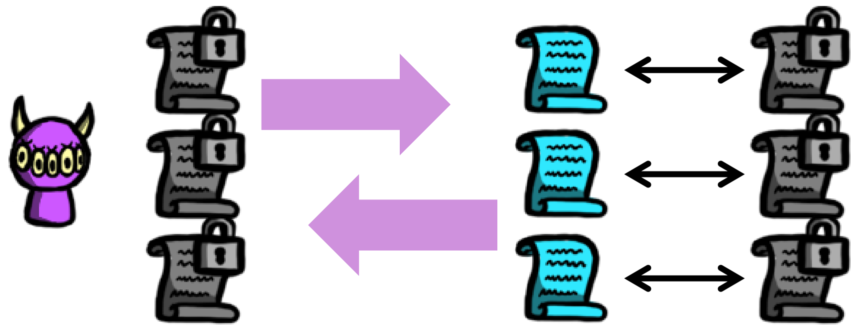
Cryptography Attack Models

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



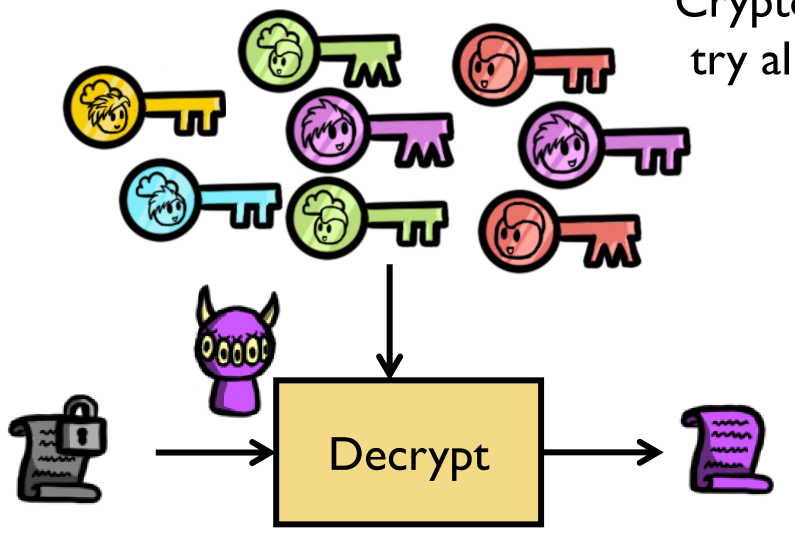
- Chosen ciphertext attack: Even can choose any arbitrary ciphertext and get the plaintext (possibly many times). She wants to guess the key.
  - Can also be *adaptive*.



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: the one-time pad

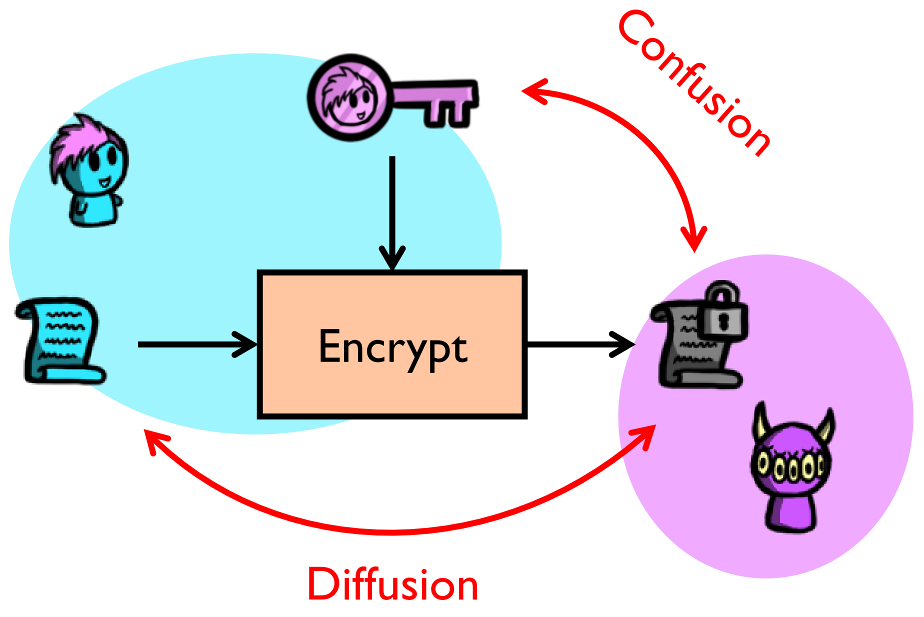


Crypto brute-forcing:
try all keys (usually)

Confusion and diffusion

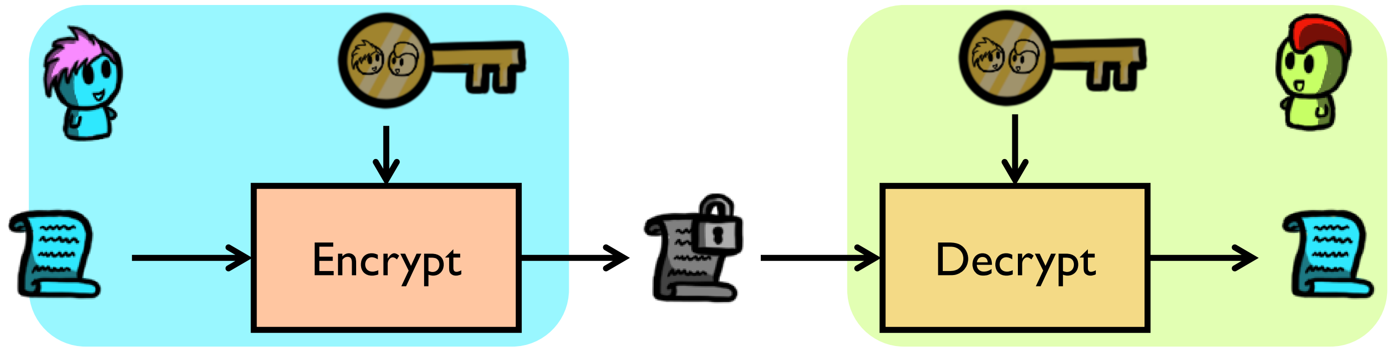
Two important properties of secure cryptosystems:

- **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)



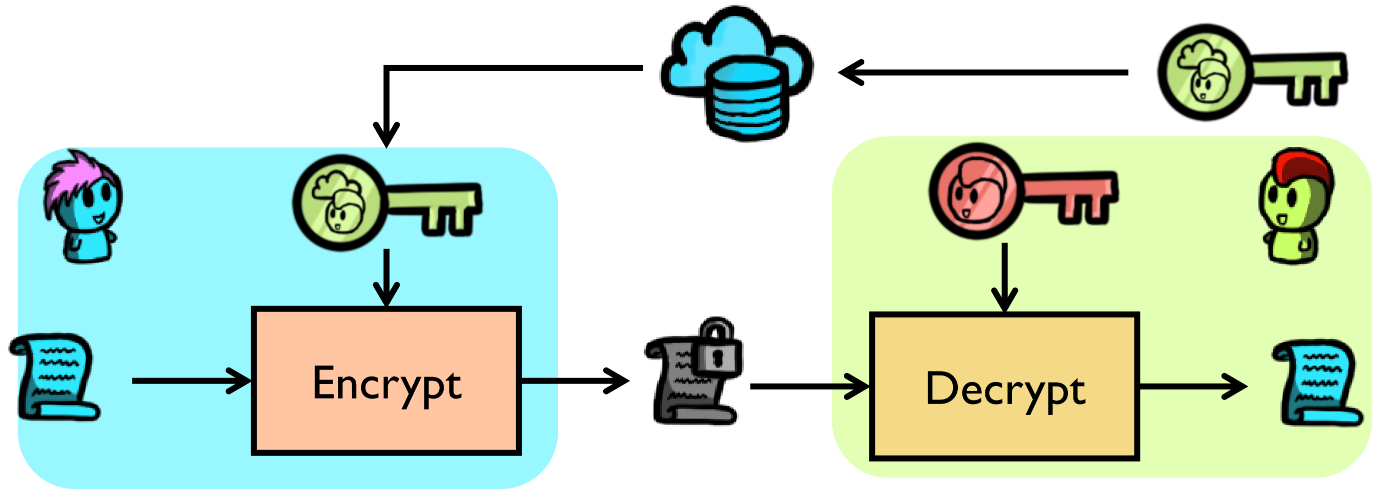
Symmetric Crypto

- 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



Module Outline

Preamble

- Cryptology
- Basic building blocks
- Kerckhoff's principle
- Cryptography attack models
- Confusion and Diffusion
- Symmetric vs Asymmetric crypto

Part 1: Ancient Crypto

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

Part 2: Symmetric crypto

- Definition
- Computational security
- Stream ciphers
    - Examples (A5/1, Salsa/Chacha)
- Block ciphers
    - Modes of operation
    - AES

Part 3: Public-key crypto

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

Part 4: Integrity and Authentication

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

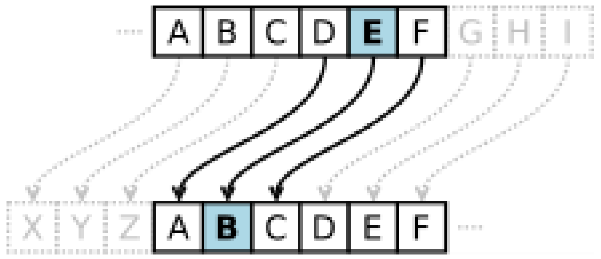
Ancient Cryptography

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

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: XQQXZHQLKFDEQ
```

Why did people think this was secure?

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.

Vigenère cipher

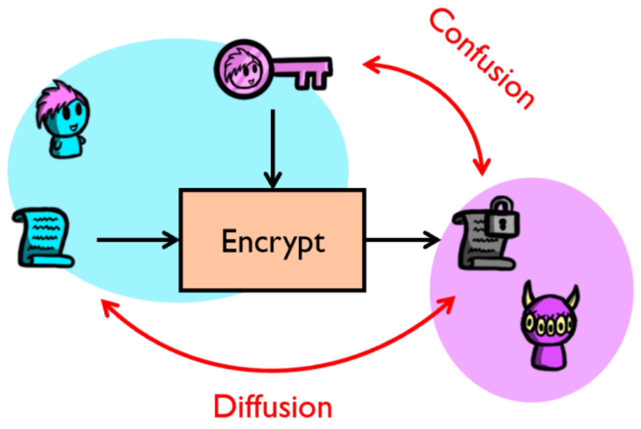
How would you break it?

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

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→ R (17)
   - …

```
Plaintext:  ATTACKTONIGHT
Key:        MYKEYMYKEYMYK
Ciphertext: MRDEAWRYRGSFD
```

How is the confusion and diffusion of this cipher?



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:        MYKEYMYKEYMYK

Ciphertext: MRDEAWRYRGSFD
```

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:         VMRKHXUTEPGLDNYBCFZOJAIWSQ
Ciphertext:  VMRKHXUTEPGLDNYBCFZOJAIWSQ
```

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

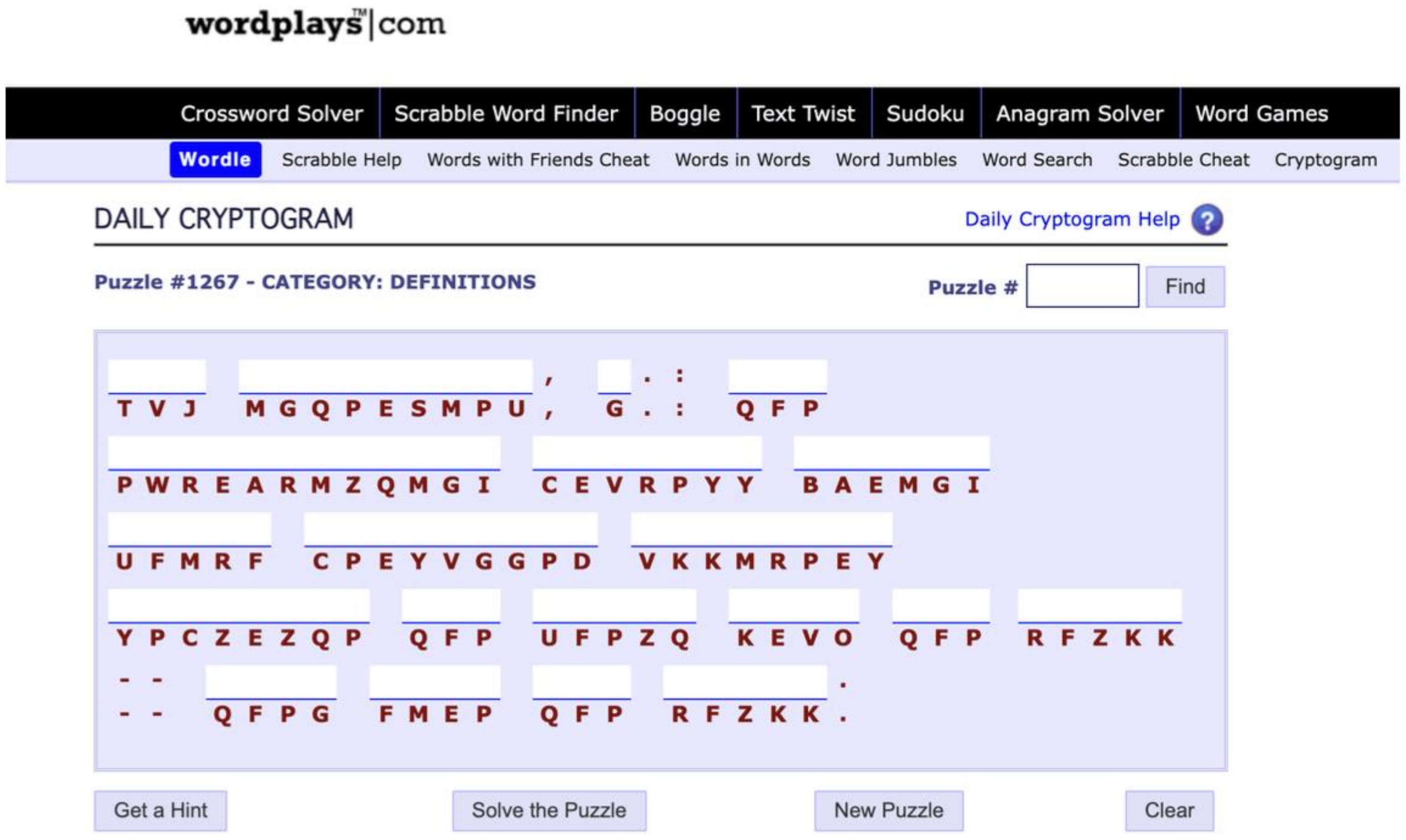
How many possible keys are there in an alphabet of 26 letters?

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

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?

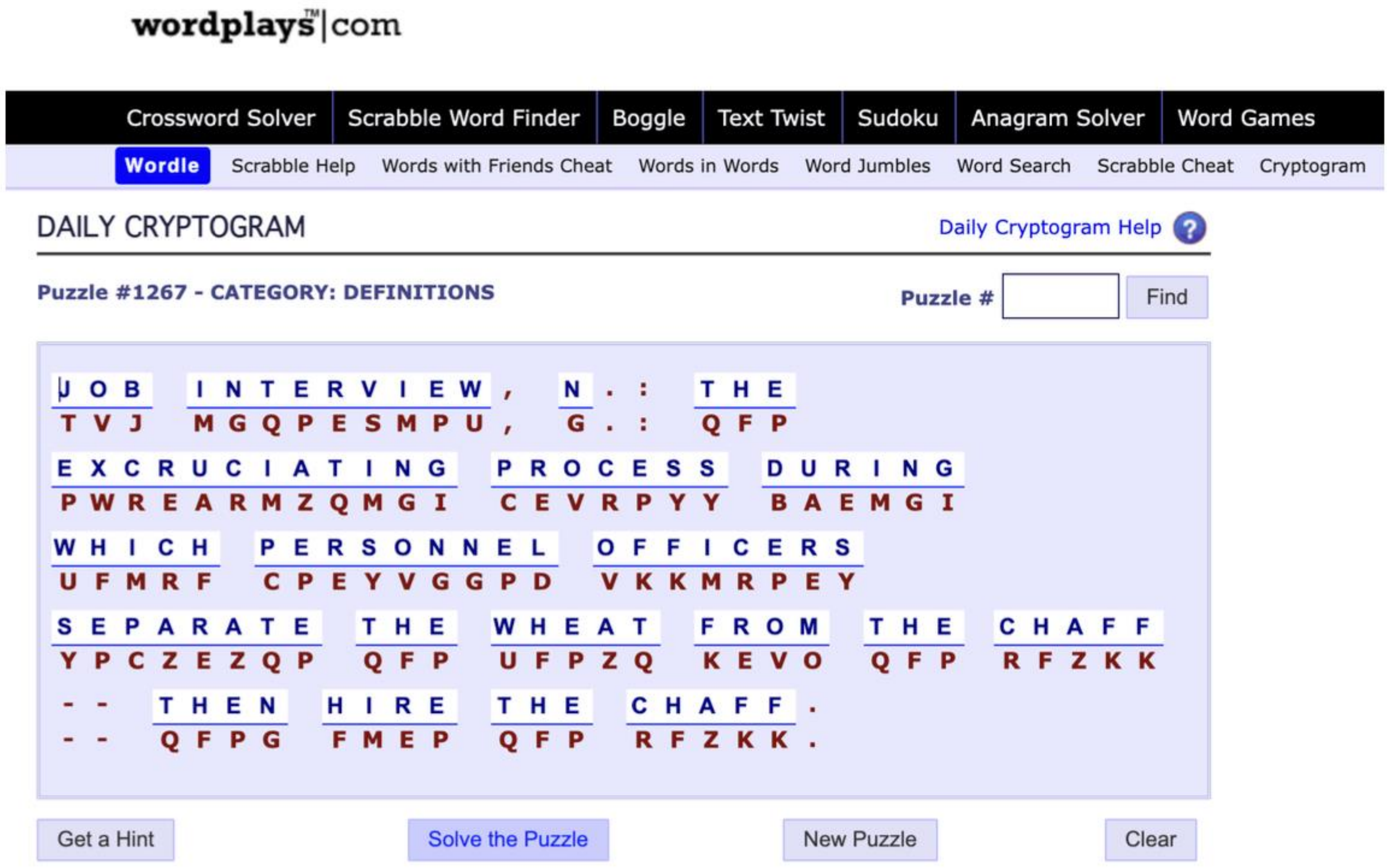


DAILY CRYPTOGRAM

Puzzle #1267 - CATEGORY: DEFINITIONS

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



DAILY CRYPTOGRAM

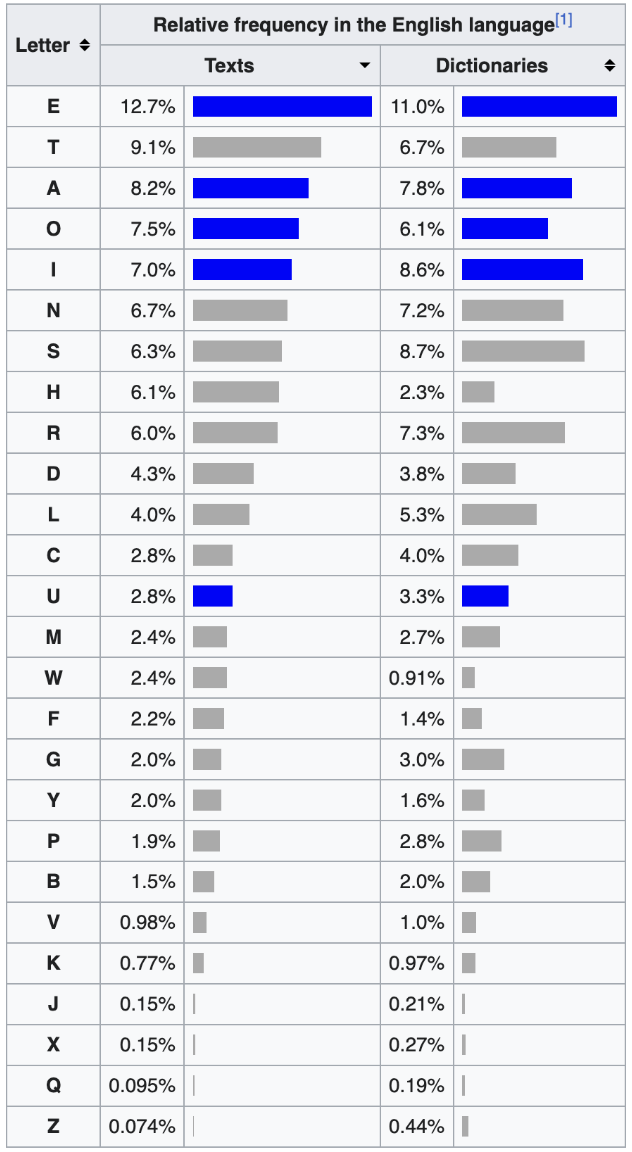
Puzzle #1267 - CATEGORY: DEFINITIONS

```
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 .
```

Breaking weak ciphers

- What do you think of the security of this substitution cipher?
- 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?



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"

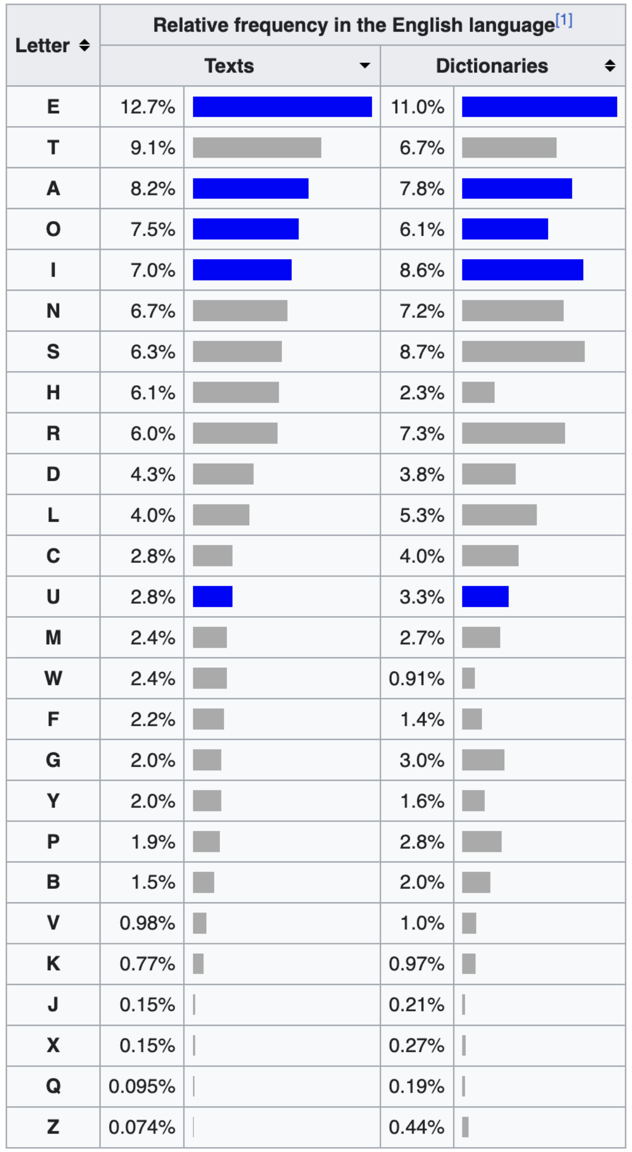
Index of Coincidence

- 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.

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

$c = 26$ 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→0, B→1, C→2,…,Z→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?

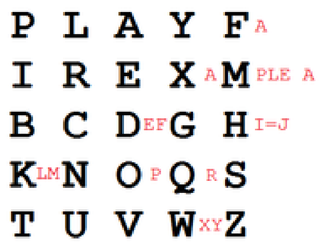
Does this always work?

The Playfair cipher

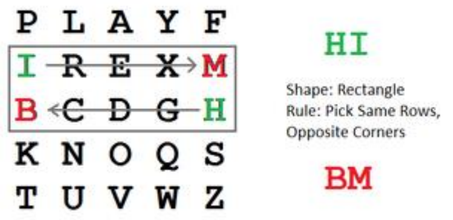
- Invented by Wheatstone (in 1854) for the telegraph, but has the name of Lord Playfair, who made it popular.
- Used later for military purposes (e.g., World War 1, although it was already deemed as insecure back then).
- It is a digram substitution cipher (replaces two letters for two other letters).

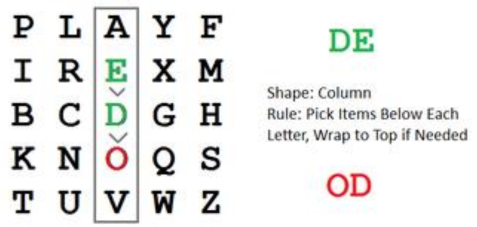
How it works:

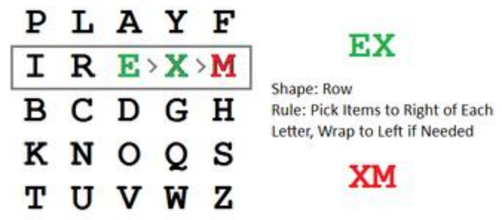
- The alphabet space is 25 characters (e.g., we assume J and I are the same letter, represented as I).
- The key is a permutation of those 25 characters (e.g., random permutation, or a password plus padding).
- Avoid identical consecutive letters (e.g., add an X in between every two repeated letters).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



"hide the gold in the tree stump" →
"HI DE TH EG OL DI NT HE TR EX ES TU MP"







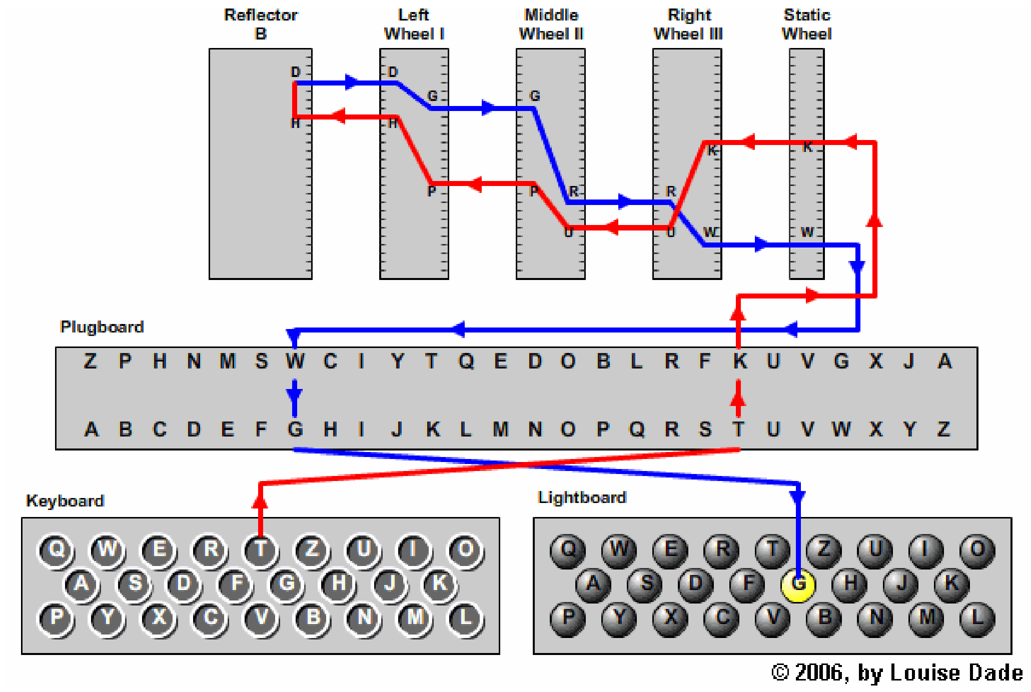
P L A Y F
I R E X M
B C D G H
K N O Q S
T U V W Z
EX
Shape: Row
Rule: Pick Items to Right of Each Letter, Wrap to Left if Needed
XM

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

Example from the Wikipedia.

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.



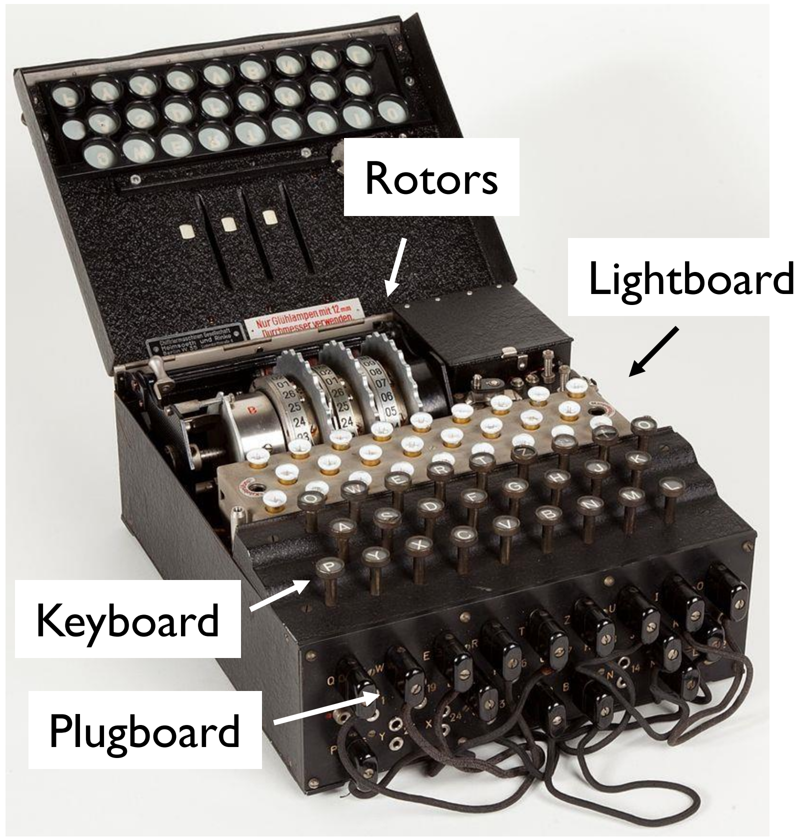
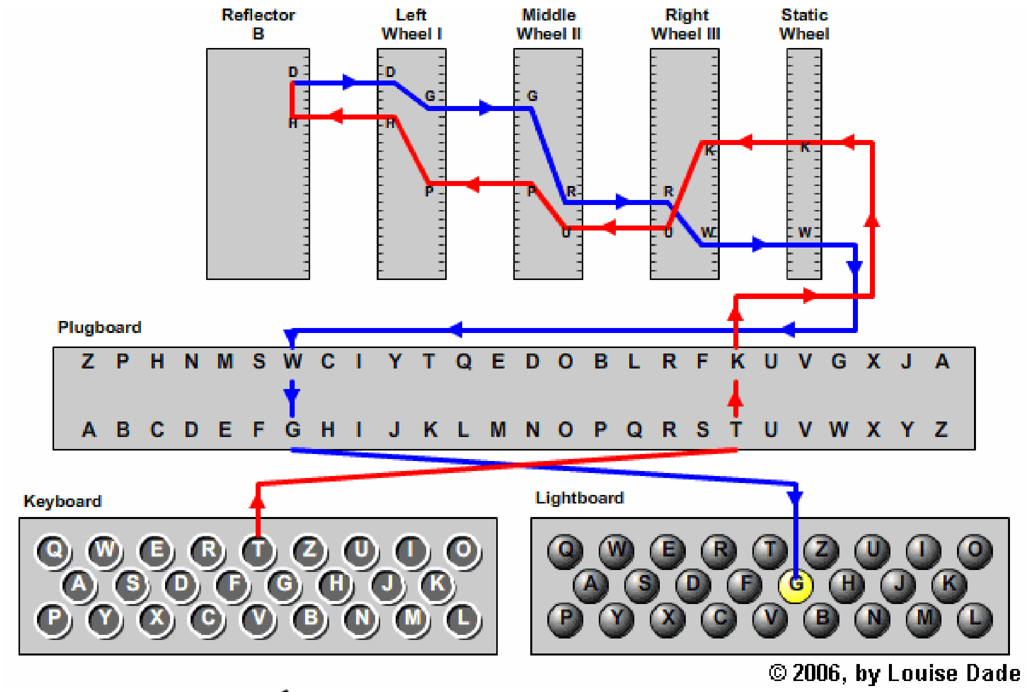


image source: http://enigma.louisedade.co.uk/wiringdiagram.png

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 $150*10^{12}$ combinations.
- The total number of possible keys was around $159*10^{18}$
- **Problem**: a letter could never match itself!! (an encryption of a letter could never be that same letter).



How many rotor permutations?

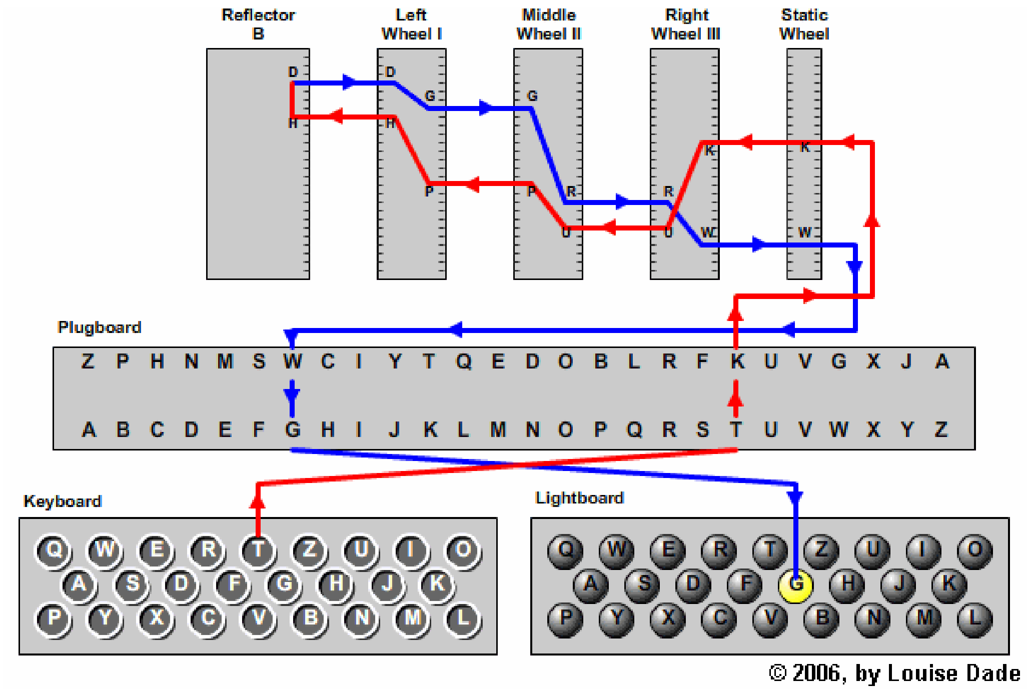
How many initial rotor settings?

The Enigma machine

- **Problem**: a letter could never match itself!! (an encryption of a letter could never be that same letter)
- E.g., ciphertext: `WJEQLDUYBNHJXP`

```
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.



What kind of attack model is this?
- Ciphertext-only attack
- Known plaintext attack
- Chosen plaintext attack
- Chosen ciphertext attack

Vernam Cipher

- Named after Gilbert Standford 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:
  $0 \oplus 0 = 0,\quad 0 \oplus 1 = 1,\quad 1 \oplus 0 = 1,\quad 1 \oplus 1 = 0$

How do we decrypt?

Is this a symmetric or asymmetric cipher?

The One-Time Pad

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

- 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.
- The one-time pad is a very unique cryptosystem: it provides information-theoretic security
- 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
```

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

Shortcomings of the One-Time Pad

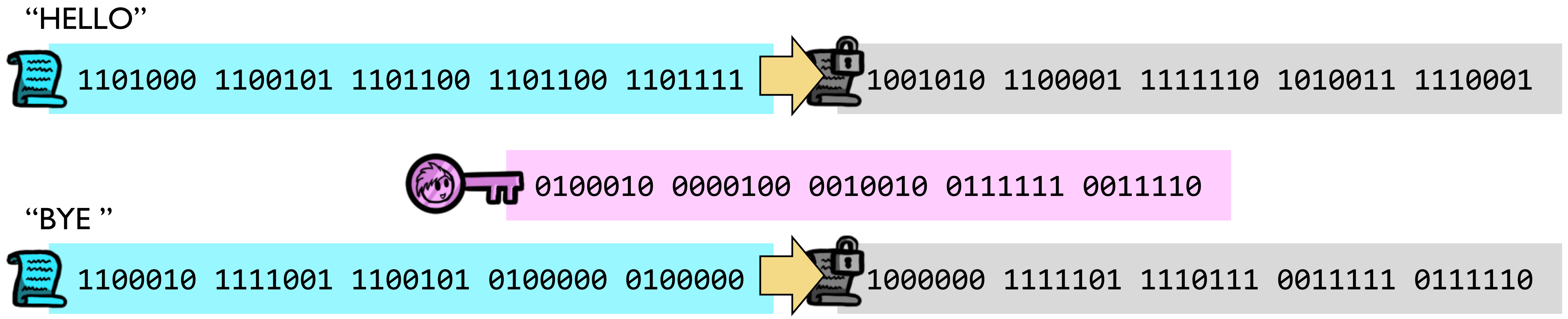
- We have perfect confidentiality! What about integrity?
- Assume your boss has your salary (in binary) encrypted with a one-time pad, and you have access to the ciphertext. What can you do with this?

```
00000000000000010101111110010000
```

```
11111101010011111001011101000101
```

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?

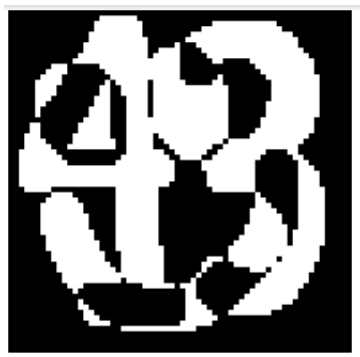
Completely insecure!! But still requires a bit of work to break…

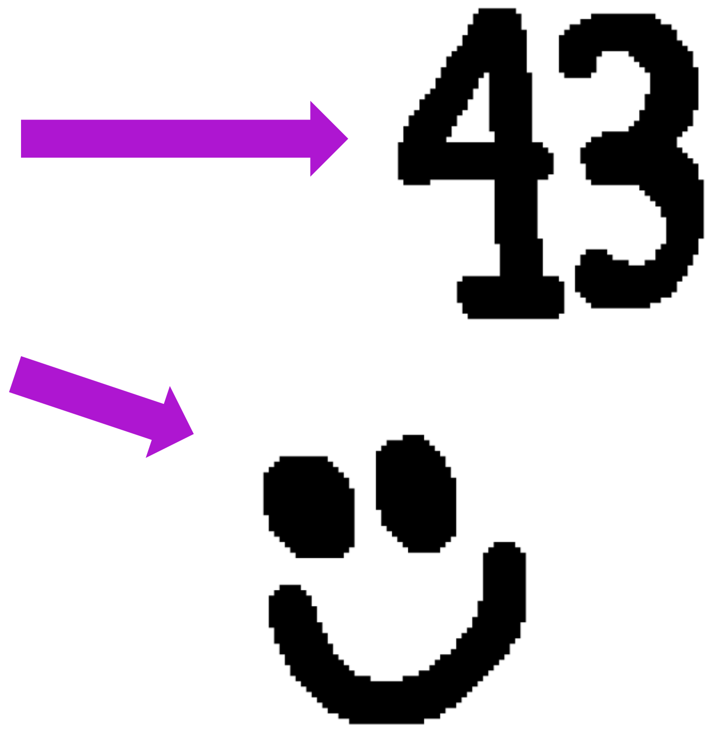
A visual example

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

C_1

C_2


$C_1 \oplus C_2$



- What do we do with them?

CPEN 442 – Introduction to Cybersecurity

Module 2 – Cryptography

Part 2 – Symmetric crypto

Module Outline

Preamble

- Cryptology
- Basic building blocks
- Kerckhoff's principle
- Cryptography attack models
- Confusion and Diffusion
- Symmetric vs Asymmetric crypto

Part 1: Ancient Crypto

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

Part 2: Symmetric crypto

- Definition
- Computational security
- Stream ciphers
  - Examples (A5/1, Salsa/Chacha)
- Block ciphers
  - Modes of operation
  - AES

Part 3: Public-key crypto

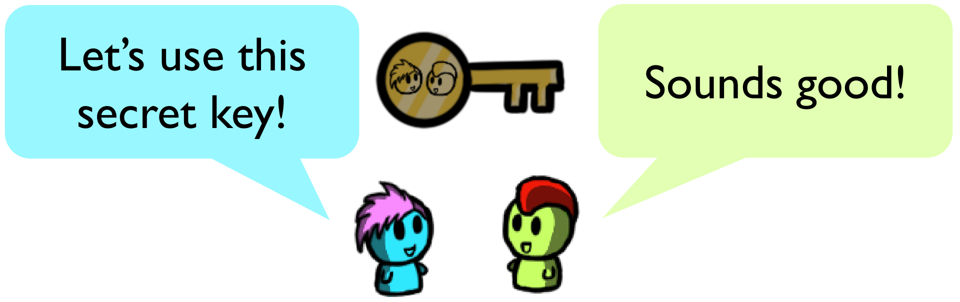
- Definition
- Example: textbook RSA
  - Description
  - Shortcomings
- Key sizes
- Hybrid cryptography

Part 4: Integrity and Authentication

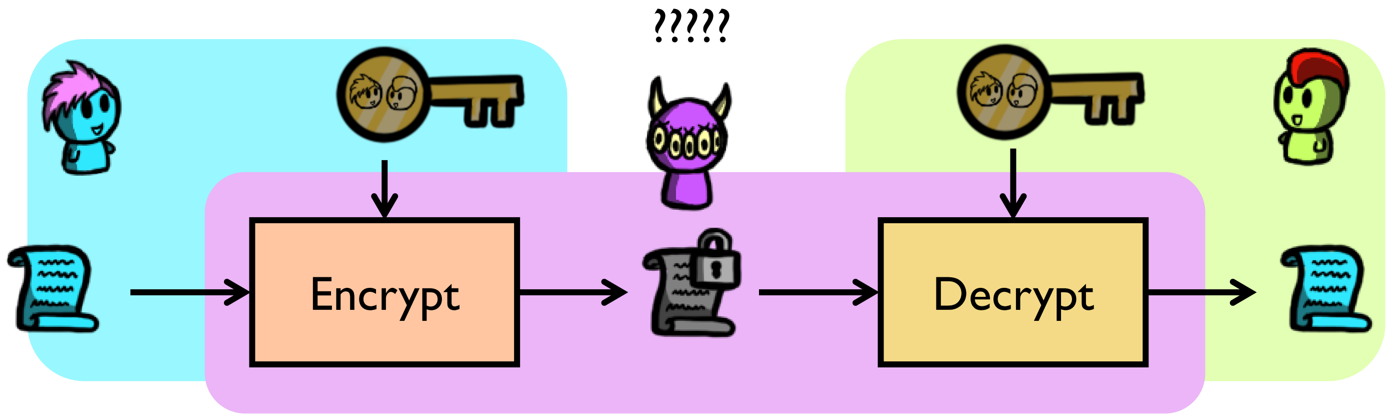
- Cryptographic hash functions
- Authentication
  - Message Authentication Codes (MACs)
  - Digital Signatures
- Repudiation
- Key management and CAs

Symmetric (or Secret-key) cryptography

- We use the same key to encrypt and to decrypt



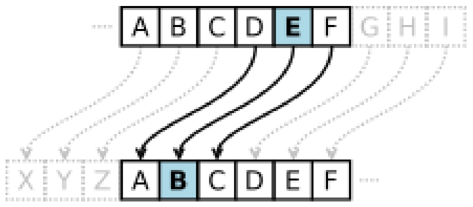
This "key exchange" is easy to do in person, but harder to do over the Internet. Eve could be watching! (we'll see how to fix this later)



Note: 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

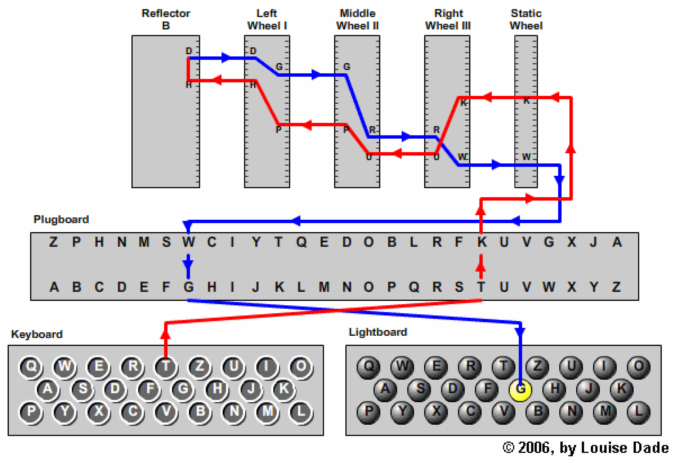
Ancient crypto: which ones are symmetric?

Caesar cipher



Vigenère cipher

```
Plaintext:  ATTACKTONIGHT
Key:        MYKEYMYKEYMYK
Ciphertext: MRDEAWRYRGSFD
```



Affine cipher

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

Vernam cipher

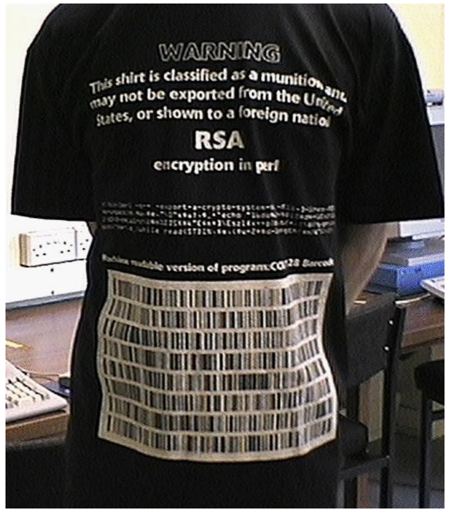
$$C = M \oplus K$$

Computational security

- In contrast to the One-Time Pad "perfect" or "information-theoretic" security, most cryptosystems have "computational" security.
- This means they can be broken if Eve does enough work.
- How much is "enough"?

- At worst, Eve tries every key:
  - How long it takes depends on how many possible keys there are
  - Usually, there are "shortcuts" and Eve does not have to try every key

Computational security: some numbers


WARNING
This shirt is classified as a munition and may not be exported from the United States, or shown to a foreign nation
RSA
encryption in perl

- 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 ($\approx 4 \cdot 10^{20}$ keys/second)**
40-bit	18 hours	11 minutes	2.7 ns
56-bit	134 years	16 months	0.18 ms
128-bit	$6.3 \cdot 10^{23}$ years	$6.3 \cdot 10^{21}$ years	$2.7 \cdot 10^{10}$ years
256-bit	…	…	…

- $2.7 \cdot 10^{10}$ years: around 2 times larger than the age of the universe, around 2.7 times larger than the expected lifetime of the sun.

Computational security: some numbers

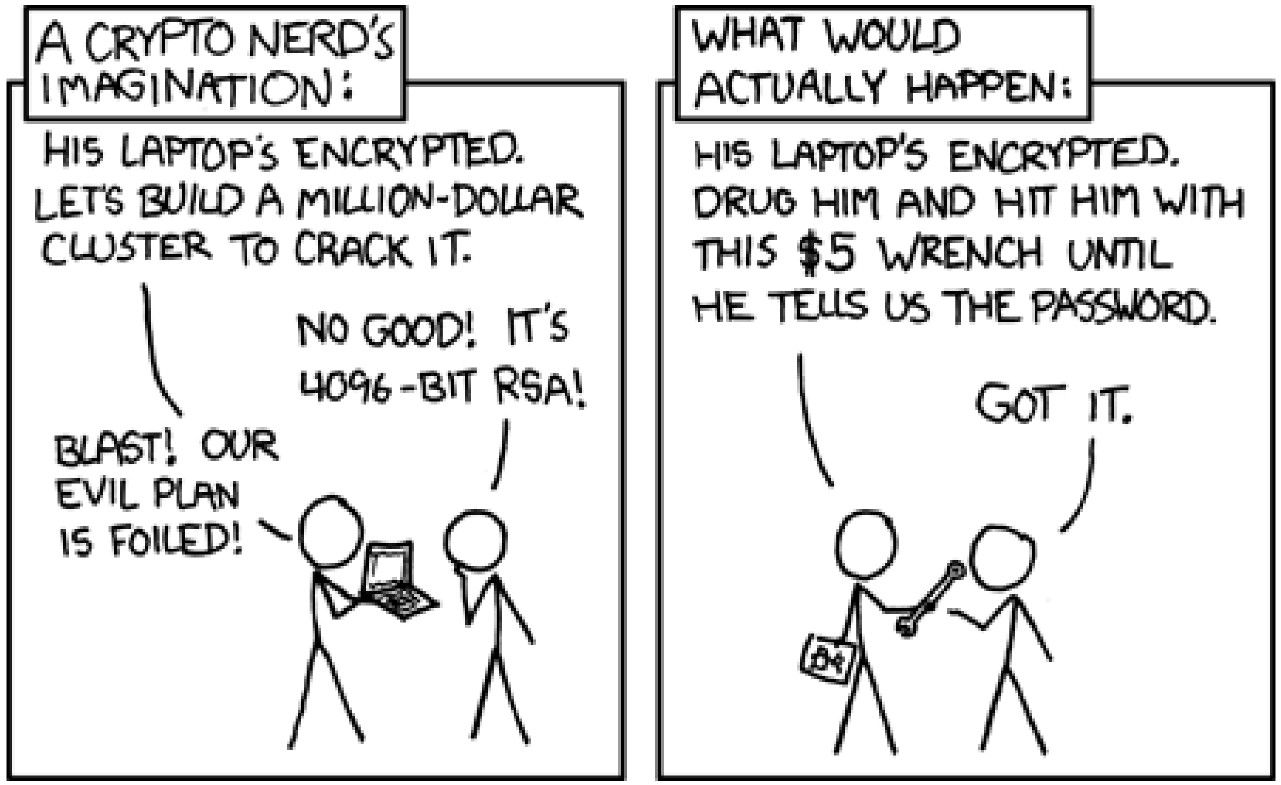
- Wait, but computers get faster over time!
- Moore's law: "the number of transistors on a microchip doubles about every two years".
- We can just wait a few hundreds of years, then crack the key in less than one hour!

Do we believe this?

What about quantum computers?

- Actually, there is a much better strategy…

A better strategy

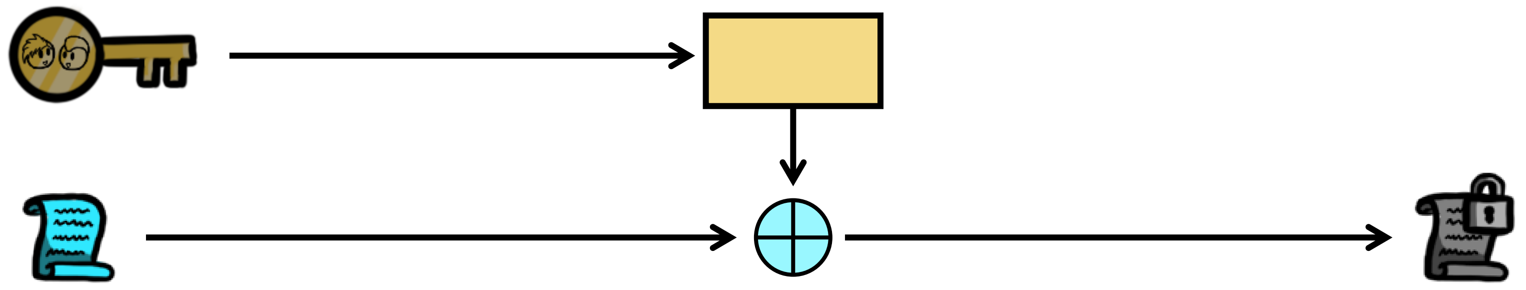

A CRYPTO NERD'S IMAGINATION:
HIS LAPTOP'S ENCRYPTED. LET'S BUILD A MILLION-DOLLAR CLUSTER TO CRACK IT.
NO GOOD! IT'S 4096-BIT RSA!
BLAST! OUR EVIL PLAN IS FOILED!
WHAT WOULD ACTUALLY HAPPEN:
HIS LAPTOP'S ENCRYPTED. DRUG HIM AND HIT HIM WITH THIS $5 WRENCH UNTIL HE TELLS US THE PASSWORD.
GOT IT.

https://xkcd.com/538/

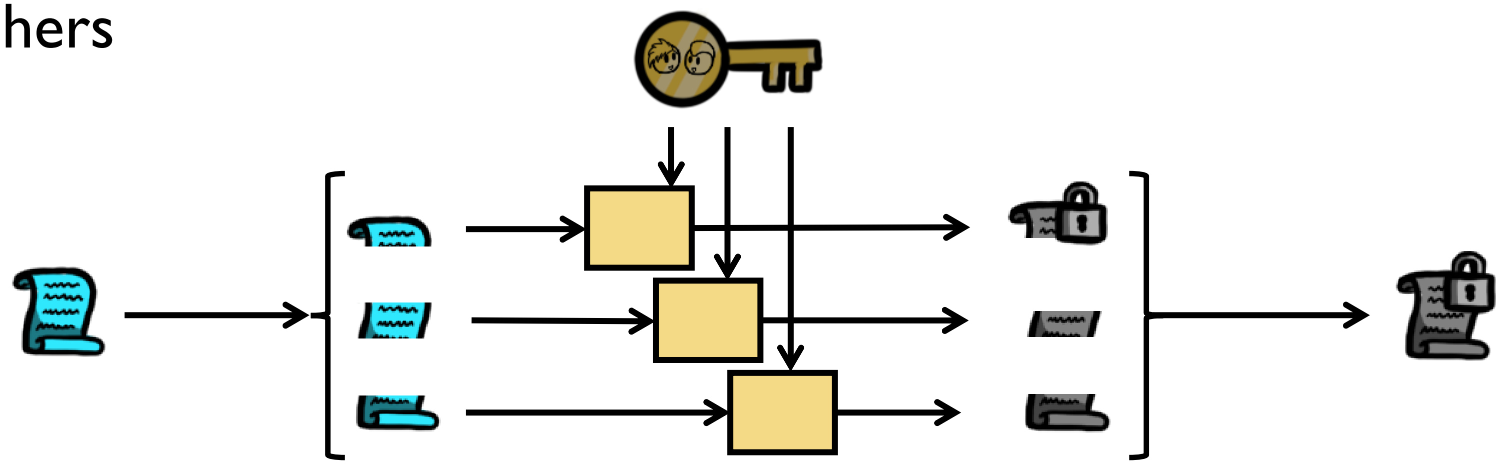
Types of symmetric cryptosystems

Secret-key cryptosystems come in two major classes:

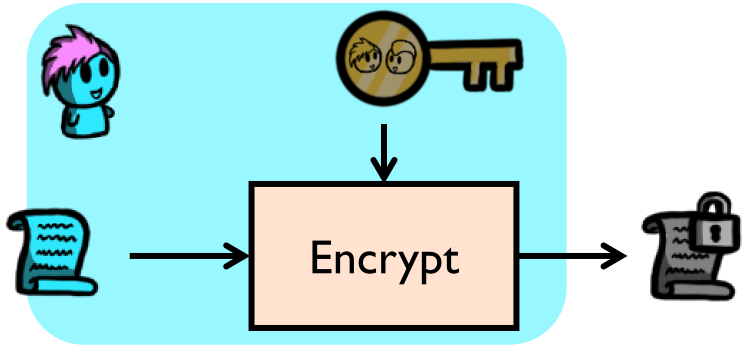
- Stream ciphers



- Block ciphers

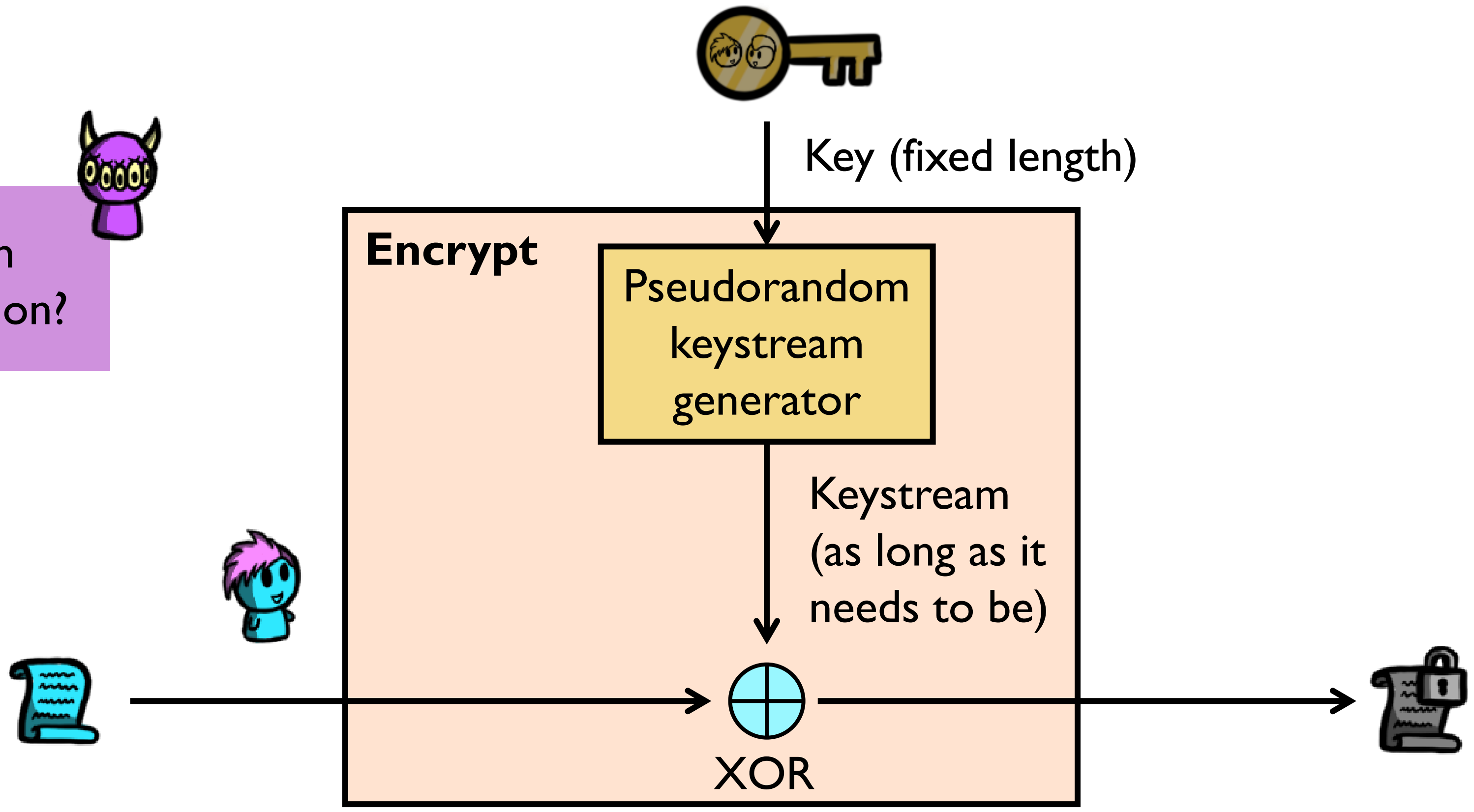


Stream ciphers



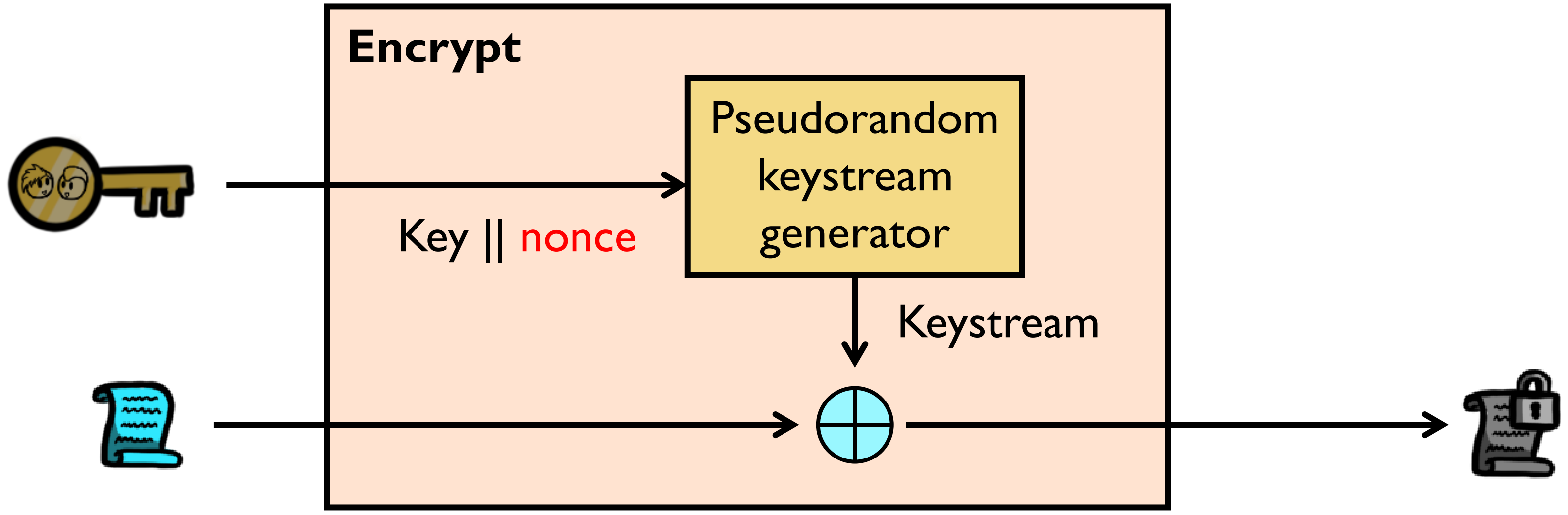
- A stream cipher operates one bit at a time.
- Basically, like the one-time pad, but using a pseudorandom keystream, instead of a truly random one.

Do these rely on confusion or diffusion?


Key (fixed length)
Encrypt
Pseudorandom keystream generator
Keystream (as long as it needs to be)
XOR

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 like the "salt")



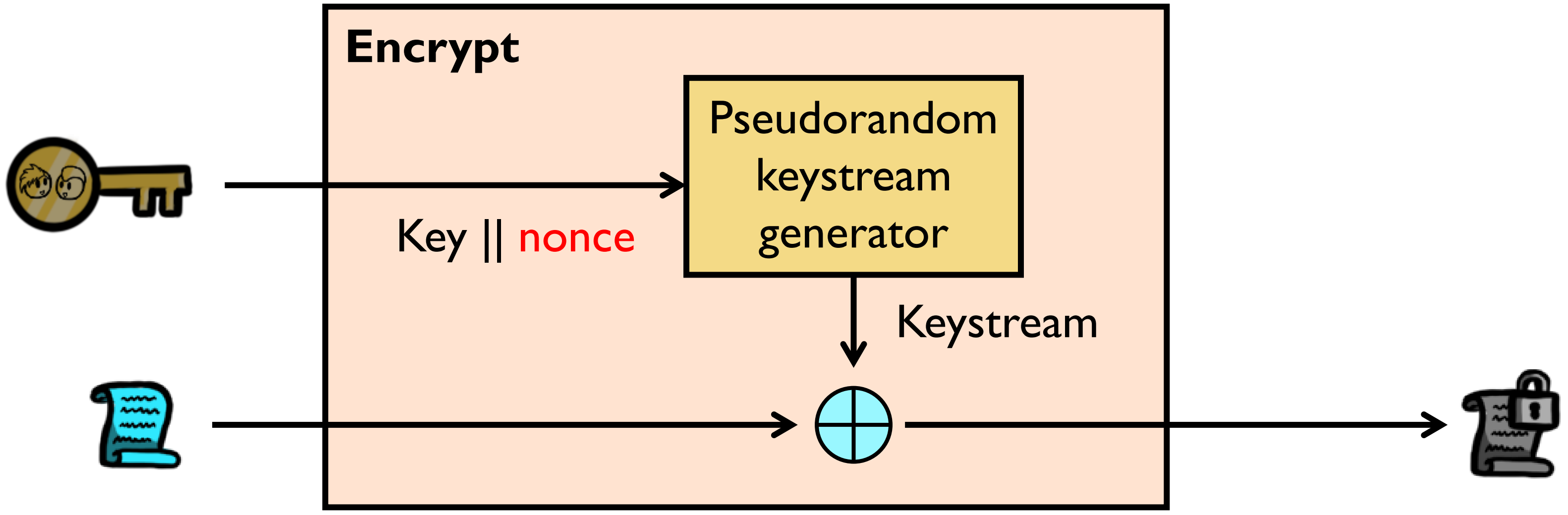
Stream ciphers: preventing keystream re-use

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

What do we gain by making the key longer?

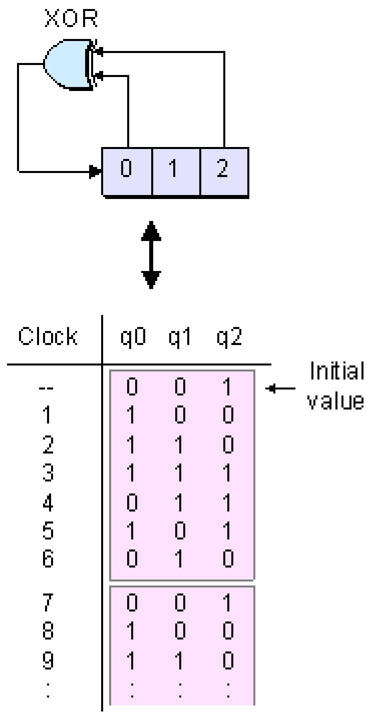
What do we gain by making the nonce longer?

What do we gain by making the nonce secret?

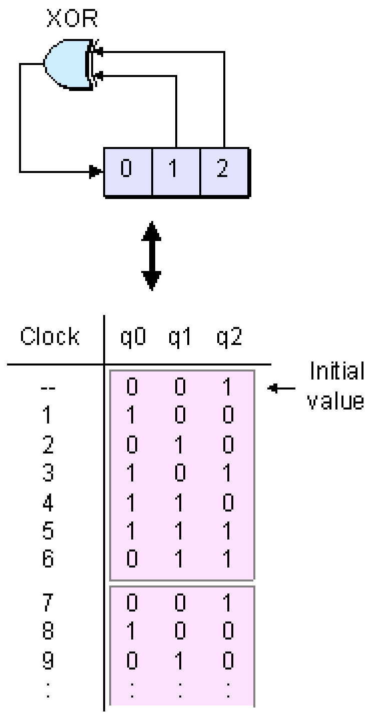


Pseudo-random number generators

- 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:



Clock	q0	q1	q2
--	0	0	1
1	1	0	0
2	1	1	0
3	1	1	1
4	0	1	1
5	1	0	1
6	0	1	0
7	0	0	1
8	1	0	0
9	1	1	0
:	:	:	:



Clock	q0	q1	q2
--	0	0	1
1	1	0	0
2	0	1	0
3	1	0	1
4	1	1	0
5	1	1	1
6	0	1	1
7	0	0	1
8	1	0	0
9	0	1	0
:	:	:	:

Notice the sequence has a period of 7

Example from: https://www.eetimes.com/tutorial-linear-feedback-shift-registers-lfsrs-part-1/

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 LSFRs:
  - The registers are of size 19, 22, and 23.
  - The 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…)

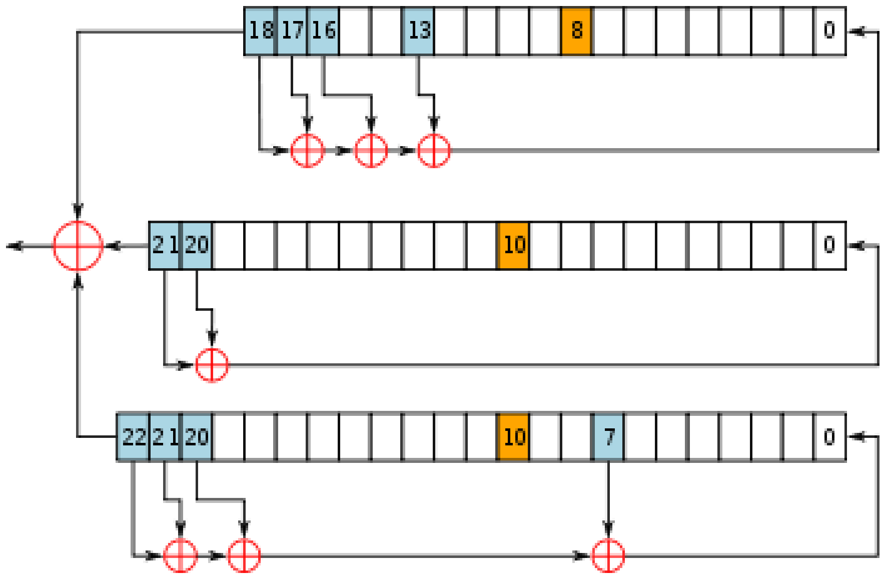
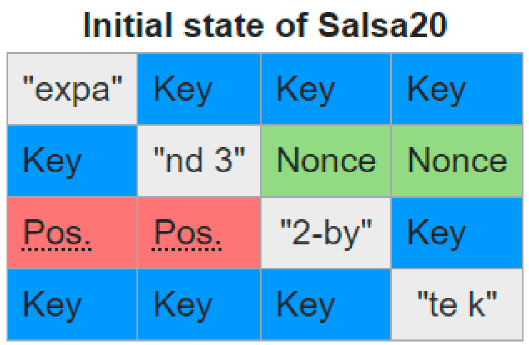


Image from: https://en.wikipedia.org/wiki/A5/1

Salsa20 and ChaCha20

- ChaCha20 is a variation of Salsa20 which is increasingly popular (Chrome, Android).
- Again: *details are not important* here; this is to give you an idea of how stream ciphers work.
- Key: 256 bits.

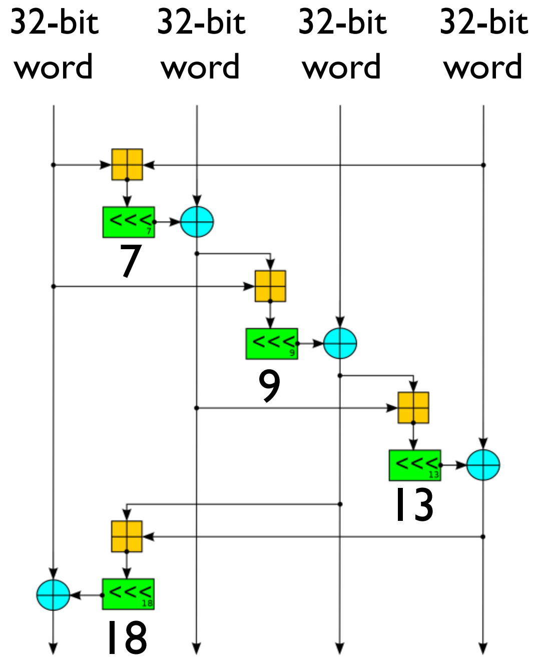
The internal state has sixteen 32-bit words arranged in a 4x4 square



Initial state of Salsa20

"expa"	Key	Key	Key
Key	"nd 3"	Nonce	Nonce
Pos.	Pos.	"2-by"	Key
Key	Key	Key	"te k"

Quarter-round: applied to rows and columns of that matrix



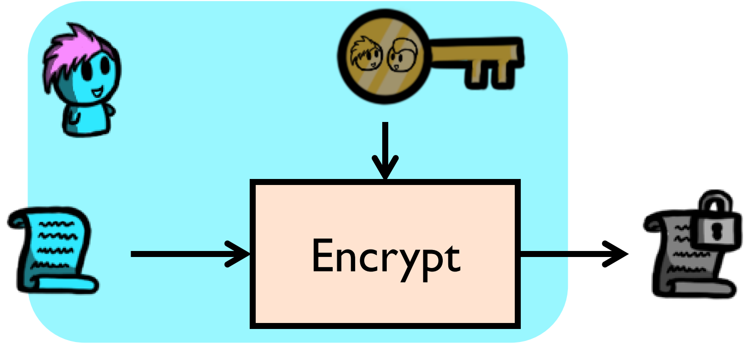
- Salsa20 does 20 rounds of mixing.
- Many technicalities omitted here…
- ChaCha20 is similar, with a different quarter-round and initial internal state

All of this from: https://en.wikipedia.org/wiki/Salsa20

Relevant Stream Ciphers

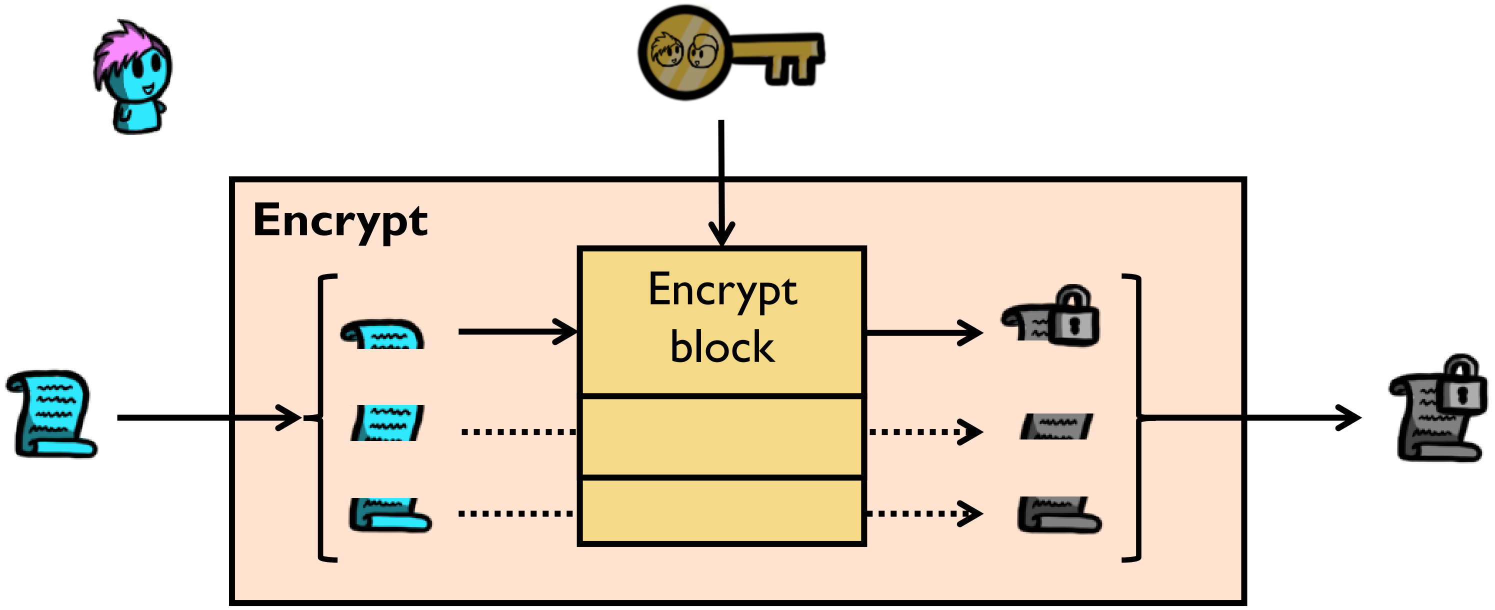
- WEP and PPTP are great examples of how not to use stream ciphers.
- RC4 was the most common stream cipher on the Internet but is now deprecated.
- ChaCha is increasingly popular (Chrome and Android)
- SNOW3G in mobile phone networks

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.



Modes of operation

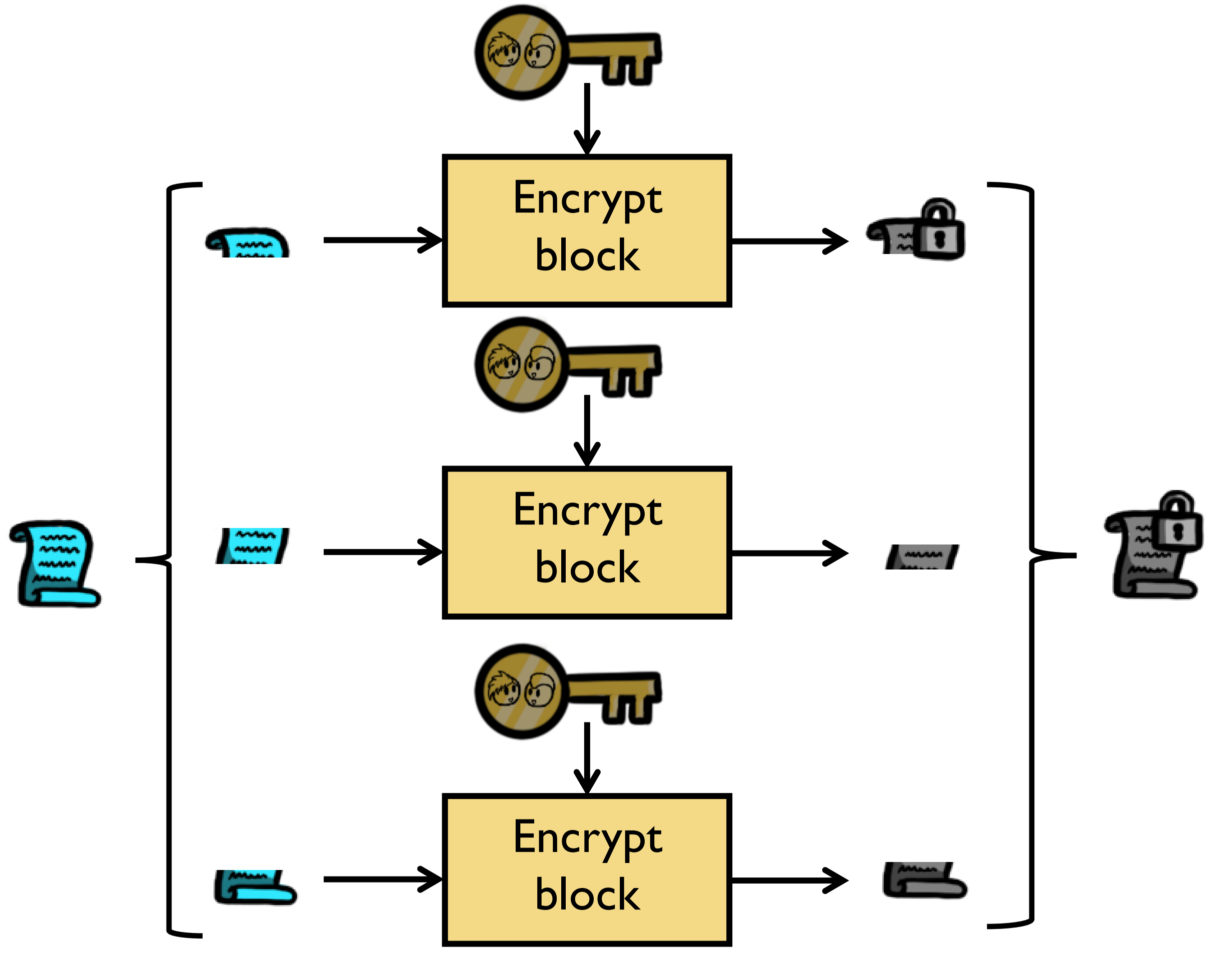
We will see the following, but there are more…

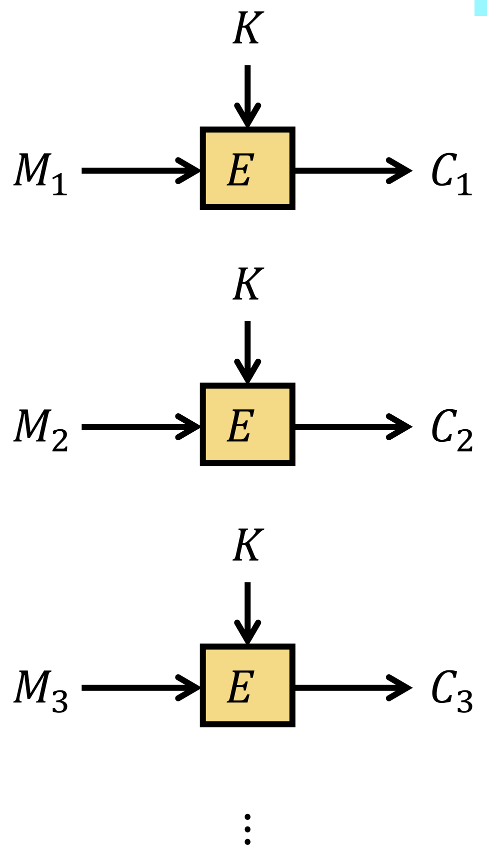
- Electronic Code Book (ECB)
  - Do not use this!!
- Cipher Block Chaining (CBC)
  - You use this with authentication (MAC) only!
- Counter Mode (CTR)
  - Interesting but you use GCM instead
- Galois Counter Mode (GCM)
  - This mode is similar to CTR, but also adds authentication
  - We might see this during the authentication/integrity part

Electronic Code Book (ECB) mode

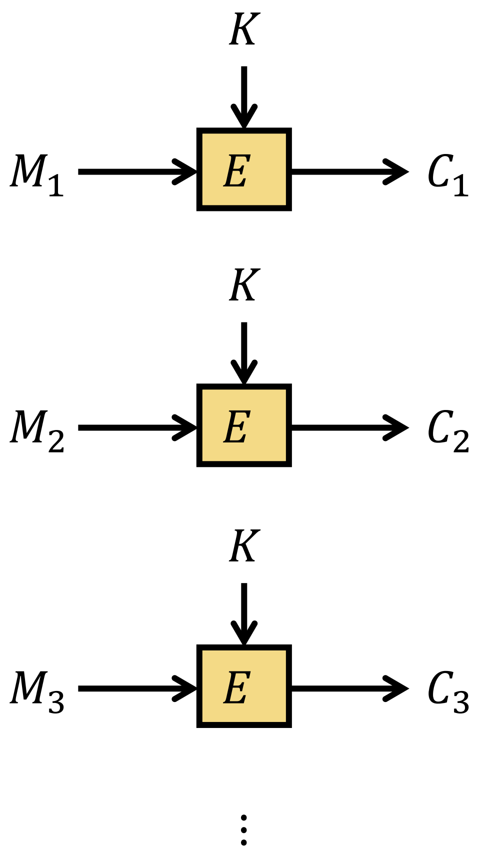
- ECB: encrypt each successive block separately

Enough drawings, cause we need space now…

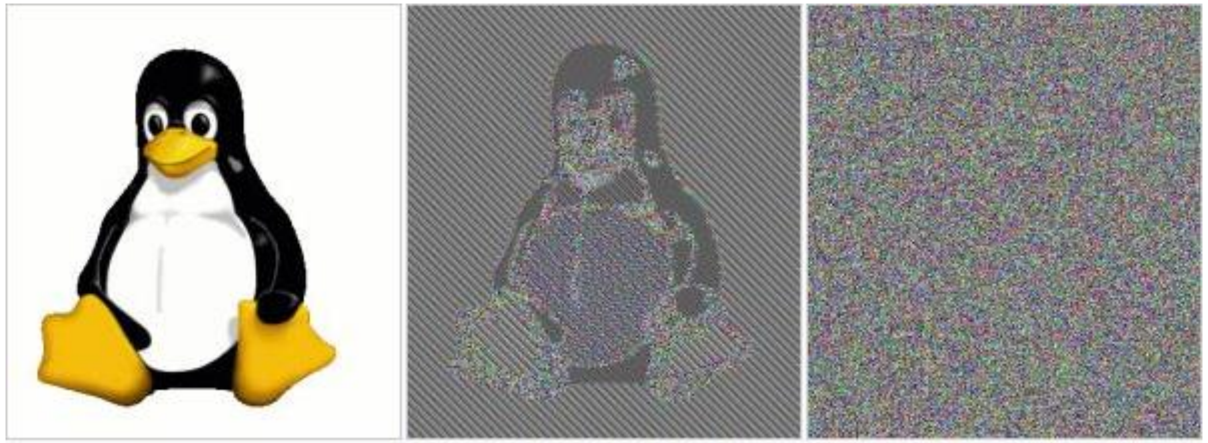




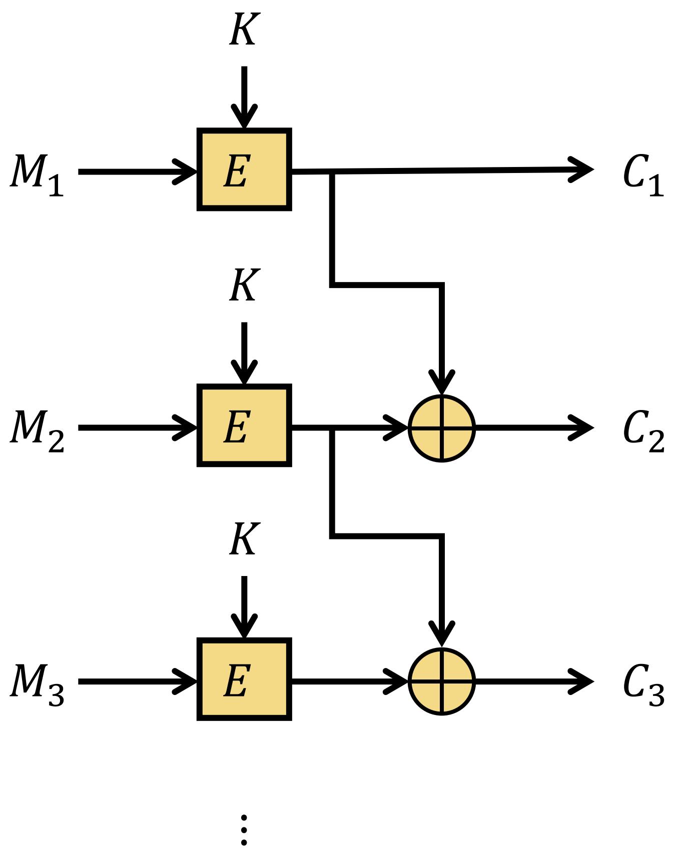
Electronic Code Book (ECB) mode



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



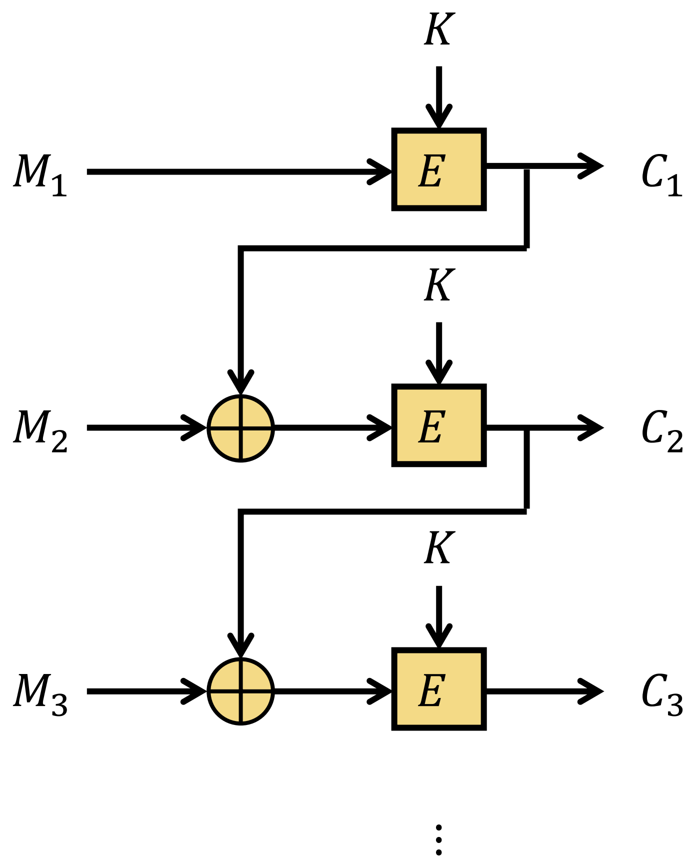
Improving ECB



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

Does this avoid repeating patterns?
Any issues here?

Improving ECB

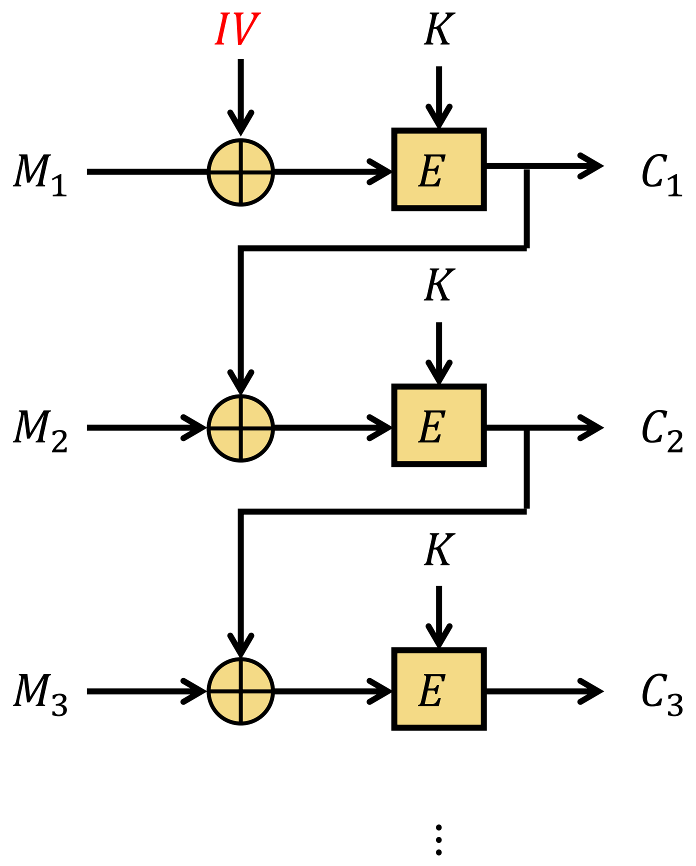


- Another way of providing "feedback"

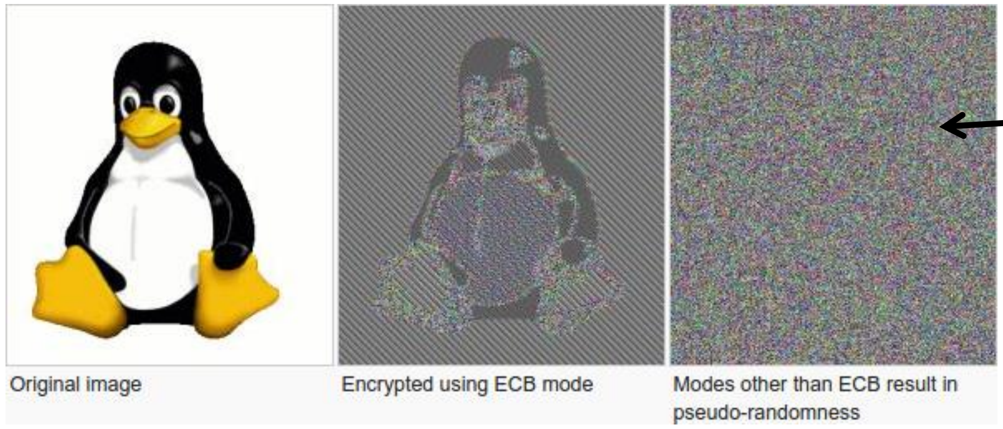
Does this avoid repeating patterns?
Any issues here?

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

Cipher Block Chaining (CBC) mode



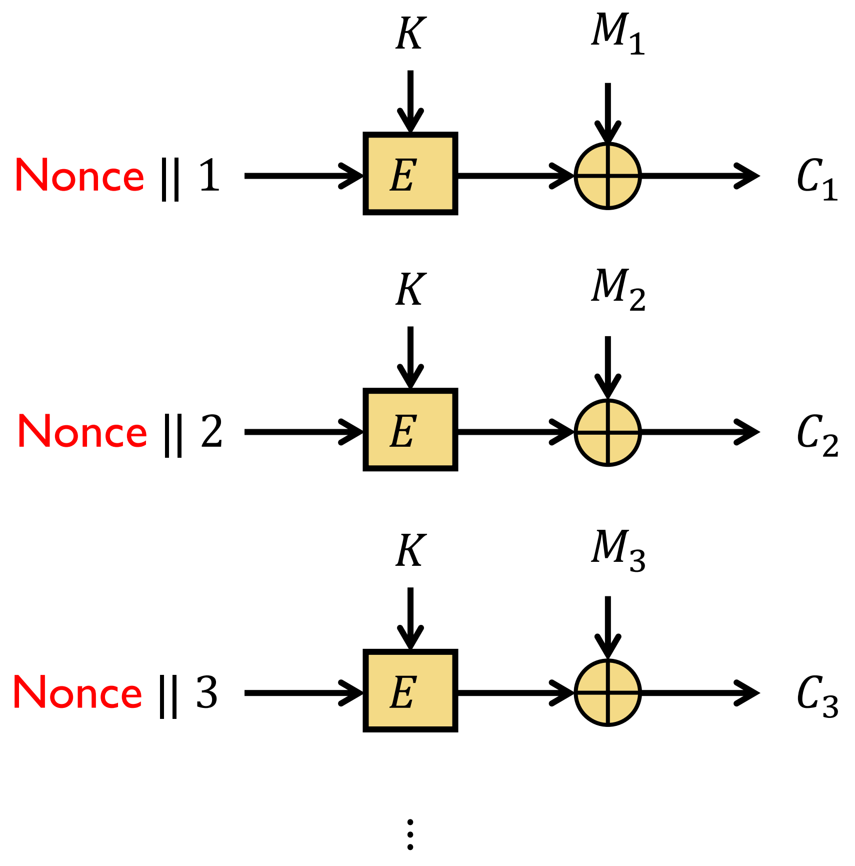
- This solves the issue of repeating patterns in blocks or the whole plaintext.
- The IV is the Initialization Vector.
  - Also called nonce.
  - Also called salt.
- We also share the IV in the clear!
  - Remember it is "conceptually different" than the key
  - Do not re-use it!



This could be CBC

Counter mode (CTR)

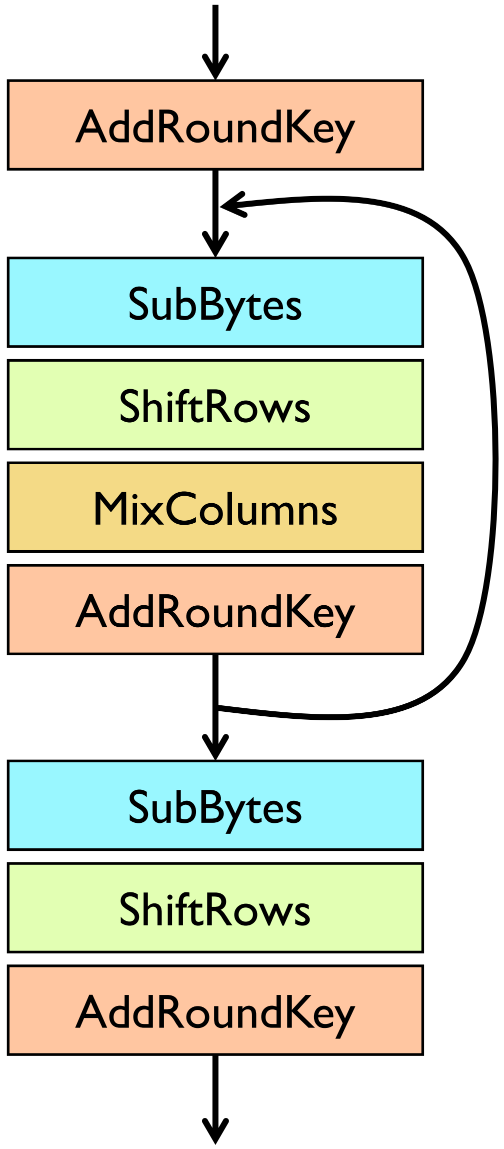
- Counter mode turns a block cipher into a stream cipher
- GCM also adds authentication, we *might* see it later…



What about the “Encrypt block” box?

- These are modes of operation.
- The actual block cipher is the “Encrypt block” box $\boxed{E}$.
- 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 (Rinjdael).
- Today: if you are going to use a block cipher, you should always use AES (Rinjdael), unless you have a very, very good reason.

Advanced Encryption Standard (AES)

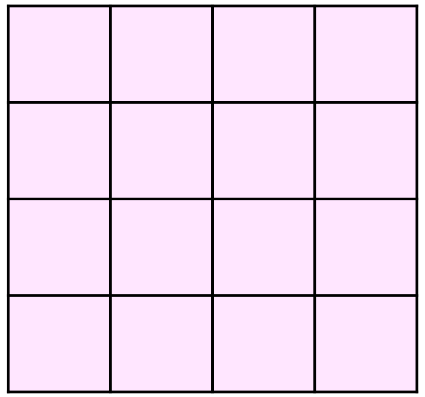


Initial round

Rounds:
- 9 rounds for 128-bit key
- 11 rounds for 192-bit key
- 14 rounds for 256-bit keys

Final round

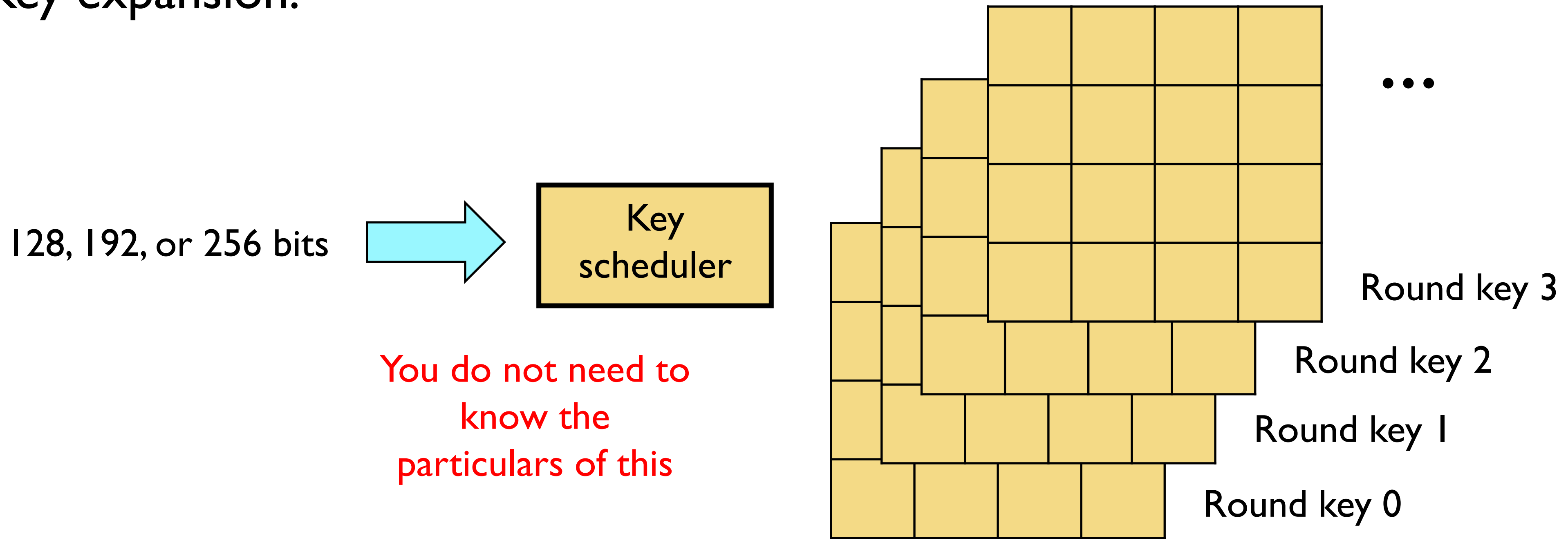
- Based on substitution-permutation networks
- Block size of 128 bits (fixed)
- Key sizes of 128, 192, or 256 bits
- AES operates on a 4x4 matrix where each element is a byte (128 bits total)



AES Key Schedule

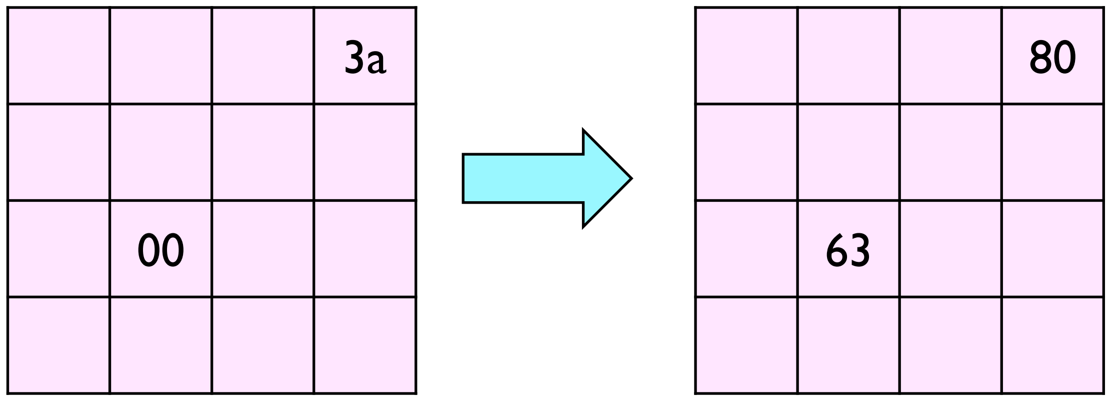
- The AES key gets "expanded" into different sub-keys to use in the AddRoundKey steps of the AES algorithm.

- Key expansion:

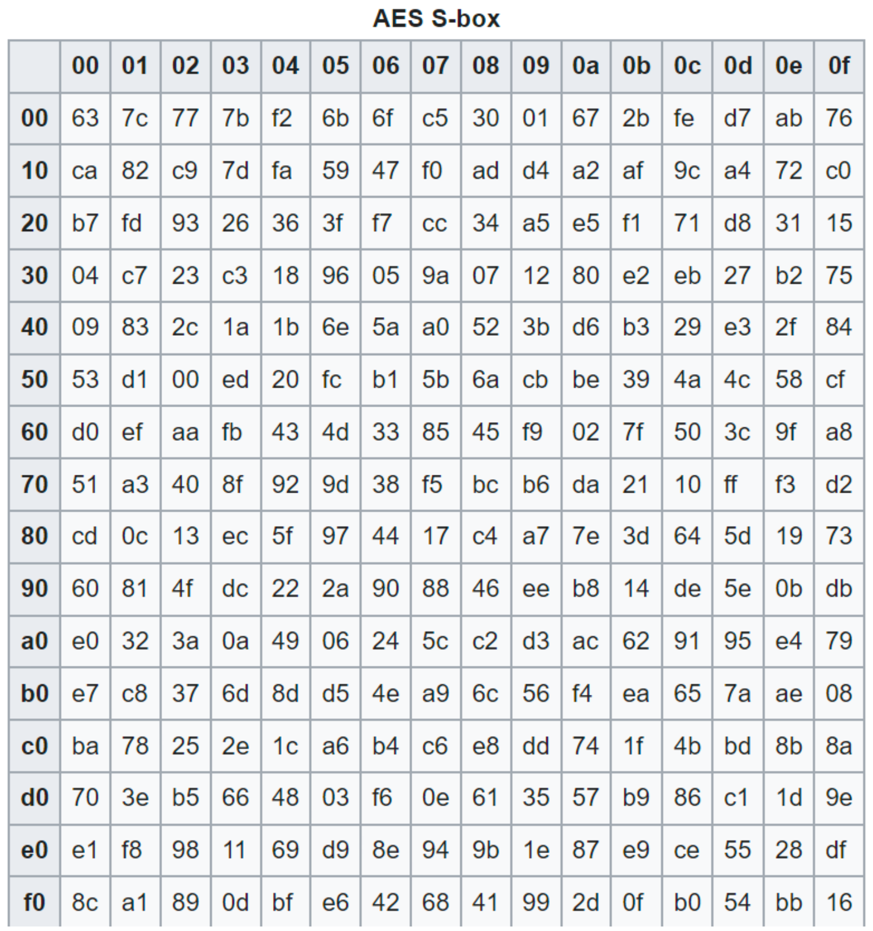


Step 1: SubBytes

- 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)



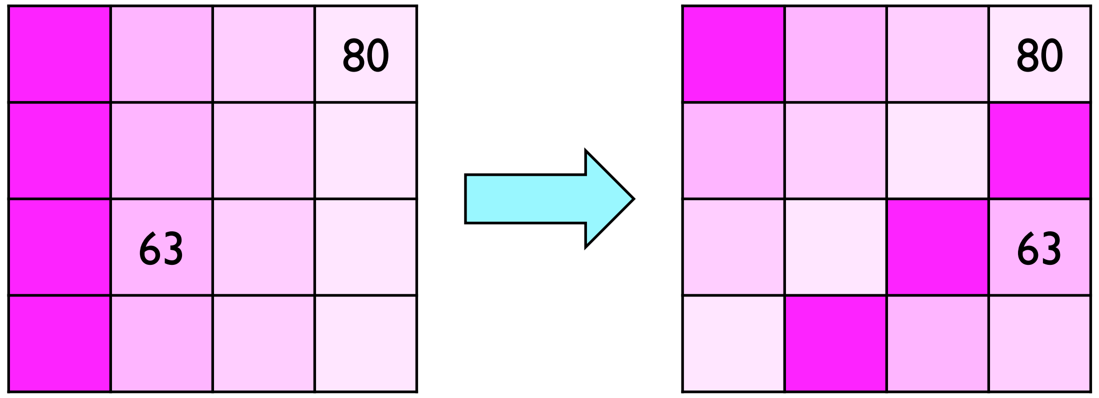
- This adds confusion (very non-linear, destroys patterns)


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

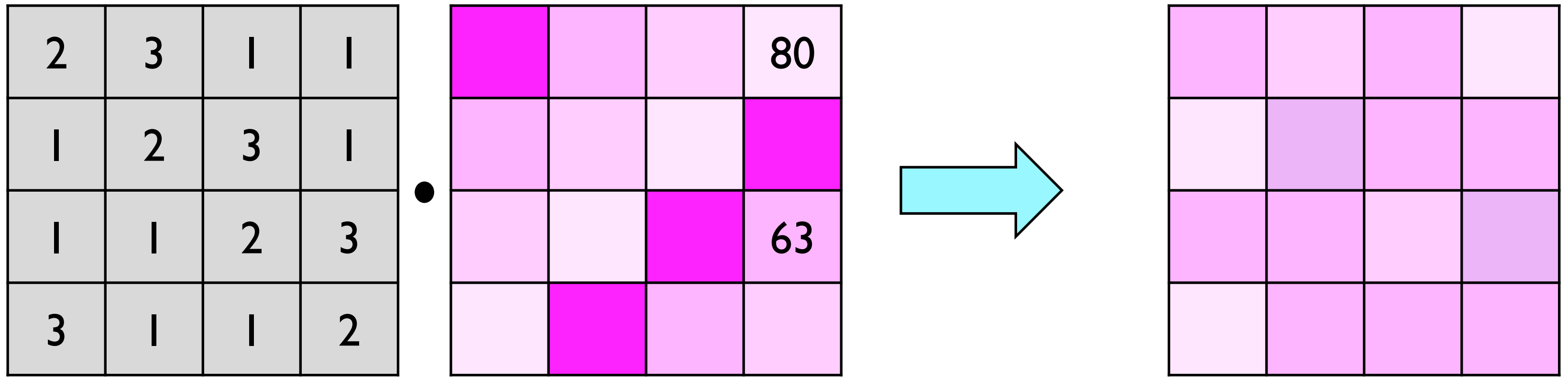
- Shift the ith row ($i \in [0,1,2,3]$) i positions to the left



- 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

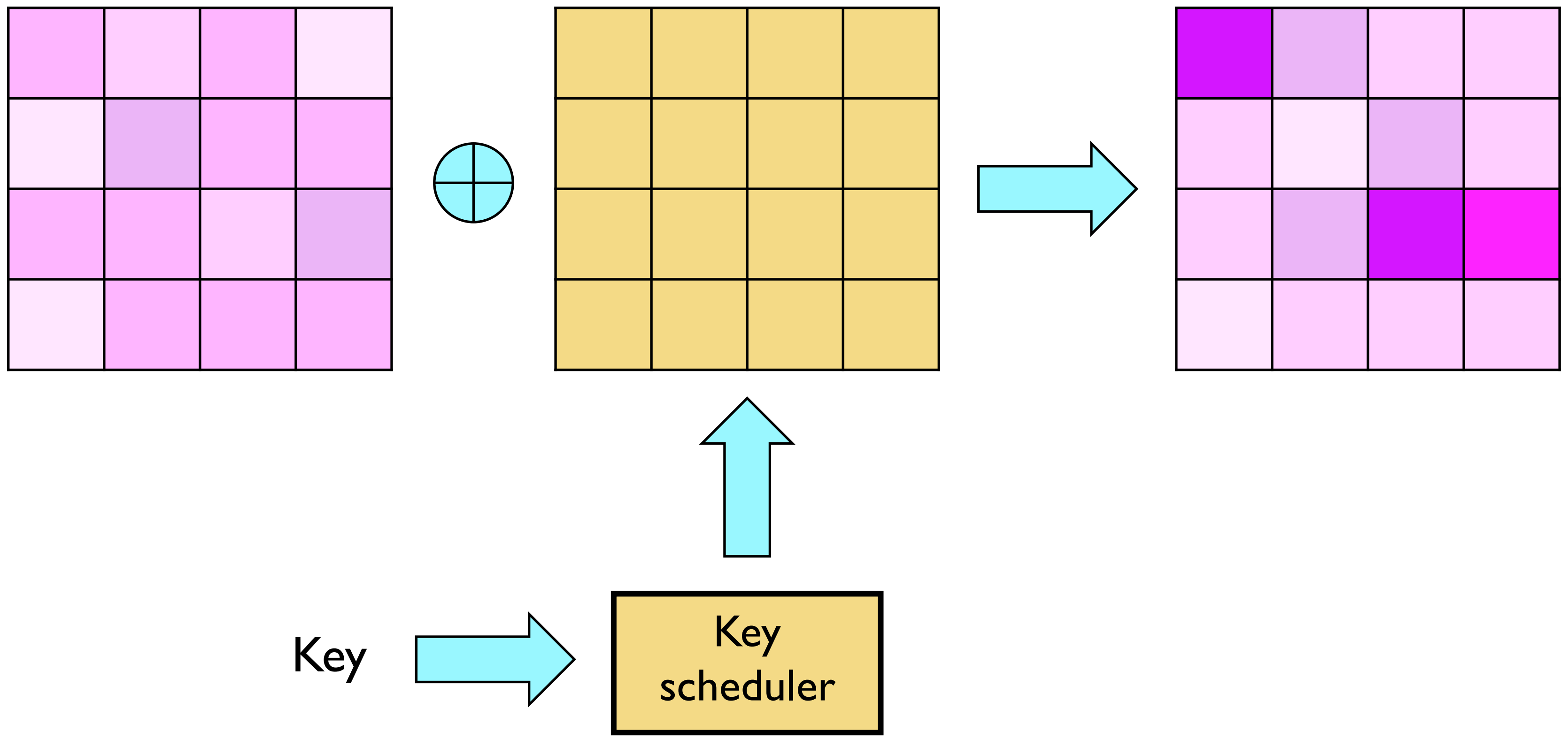
- Multiply each column using a linear transformation (matrix multiplication)



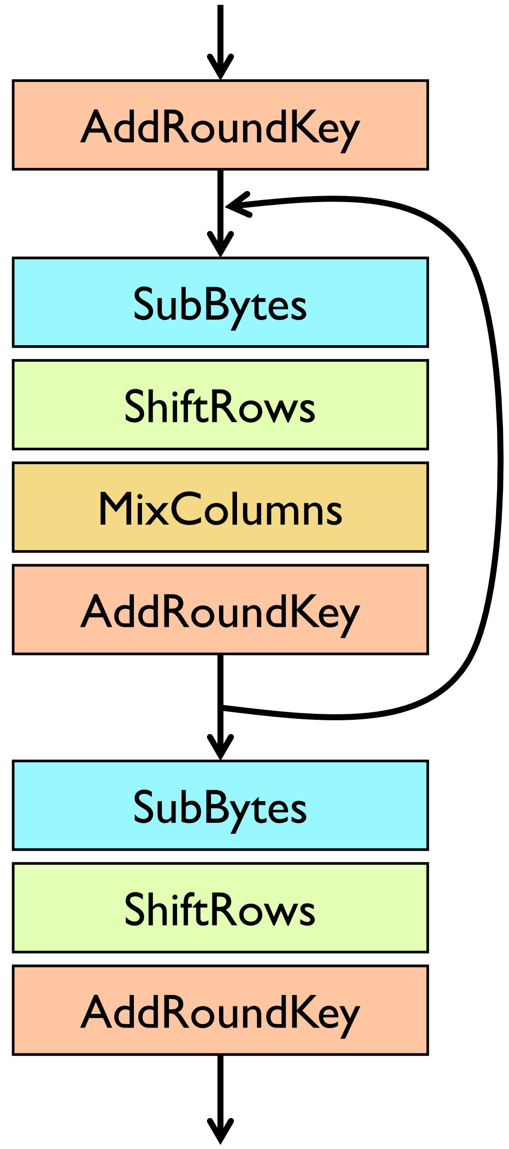
- AES arithmetic is in Galois Field using polynomials… but you don't need to know anything about this, other than it's adding some more diffusion, and it's an invertible operation.

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)



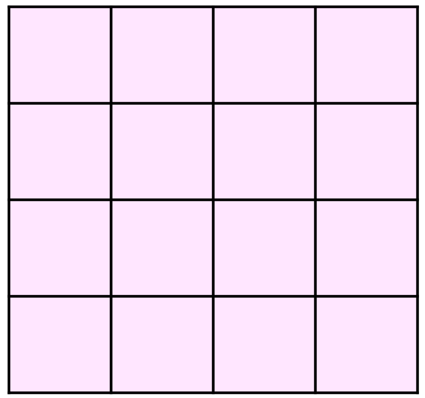
Initial round

Rounds:

- 9 rounds for 128-bit key
- 11 rounds for 192-bit key
- 14 rounds for 256-bit keys

Final round

- 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)



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… (Part 3)

- (we still don't have integrity and authentication, that's in Part 4)

CPEN 442 – Introduction to Cybersecurity

Module 2 – Cryptography

Part 3 – Public-key crypto

Module Outline

Preamble

- Cryptology
- Basic building blocks
- Kerckhoff's principle
- Cryptography attack models
- Confusion and Diffusion
- Symmetric vs Asymmetric crypto

Part 1: Ancient Crypto

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

Part 2: Symmetric crypto

- Definition
- Computational security
- Stream ciphers
    - Examples (A5/1, Salsa/Chacha)
- Block ciphers
    - Modes of operation
    - AES

Part 3: Public-key crypto

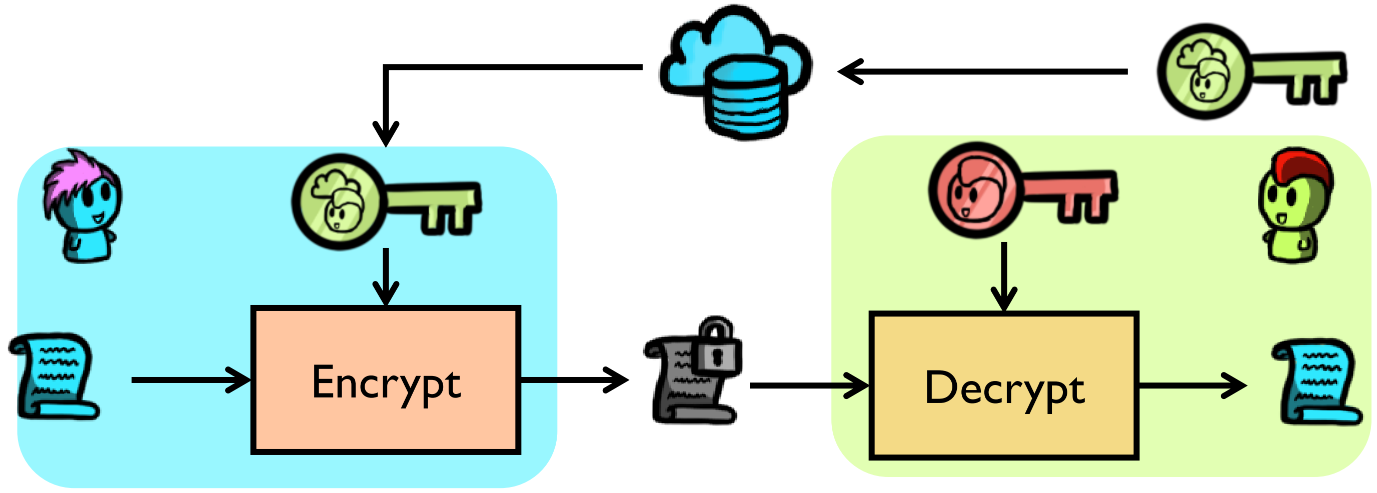
- Definition
- Example: textbook RSA
    - Description
    - Shortcomings
- Key sizes
- Hybrid cryptography

Part 4: Integrity and Authentication

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

Public-key (or Asymmetric) cryptography

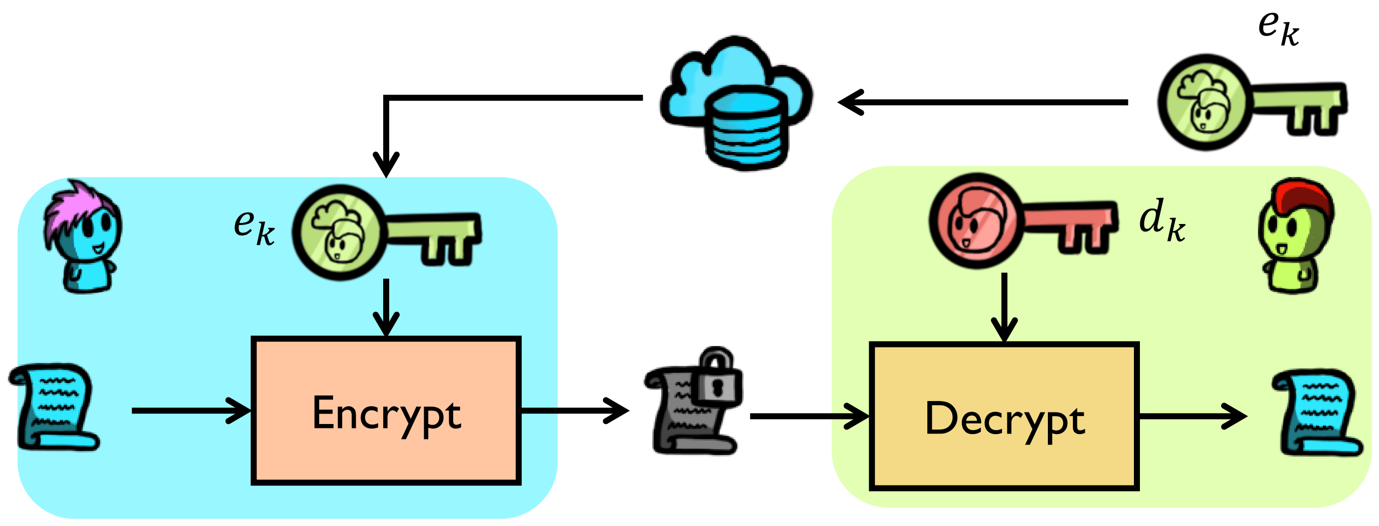
- Invented (in public) in the 1970's.
- We use a different key to encrypt and decrypt.
- The encryption key is public, and the decryption key is private



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

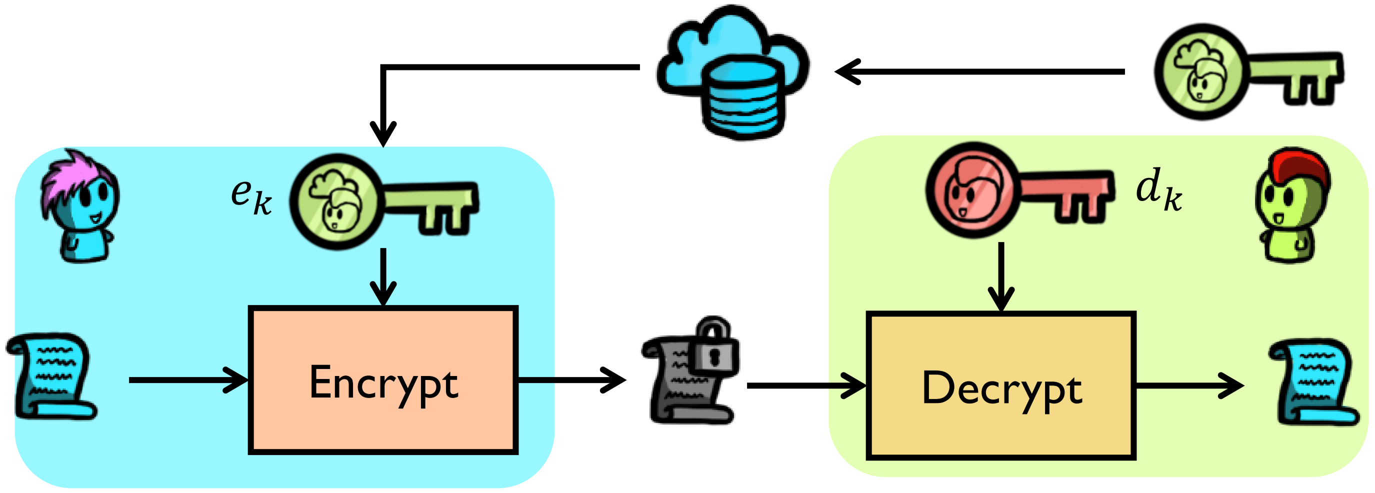
How does it work?

- 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"



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 are working with modular arithmetic: integer numbers that "wrap around".
- 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.

Some discrete math background (simplified)

- The multiplicative inverse of a modulo n (denoted a^{-1}) only exists if a and n are co-prime: $\gcd(a, n) = 1$ (greatest common divisor).
  - We mentioned this when we talked about affine ciphers
- We can find the multiplicative inverse with the extended Euclidean algorithm.
  - This algorithm finds x and y such that:

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

$$\begin{tikzcd}
a, n \arrow[d] \\
\boxed{\text{ext. Euc. alg.}} \arrow[d] \\
x, y
\end{tikzcd}$$

- What is the inverse of a mod n in this equation?
  - If we write this mod n, we get $a \cdot x \equiv 1 \pmod{n}$,
  - So x is a^{-1}.
- We will not see the details of this algorithm, but we'll assume we can compute an inverse of a mod n if we know a and n, and if $\gcd(a, n) = 1$.

Some discrete math background (simplified)

- What about the multiplicative inverse in the "exponents"?
- E.g., what is the x such that:

$$M^{a \cdot x} \equiv M \pmod{n}$$

- Turns out, when $n = p \cdot q$ for p and q prime, these "exponents" are in a field mod $\varphi(n) = (p-1)(q-1)$.
- For example: $M^{a+\varphi(n)} \equiv M^a \pmod{n}$.

When does the x above exist?

How do we compute the x above? What do we need?

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 $\gcd\big(e, \varphi(n)\big) = 1$.
- 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: $C \equiv M^e \pmod{n}$.
- Decryption: $M \equiv C^d \pmod{n}$.

- **Note:** the decryption indeed works.
  $$(M^e)^d \equiv M \pmod{n}$$

- **Note:** factoring n breaks this! Why?

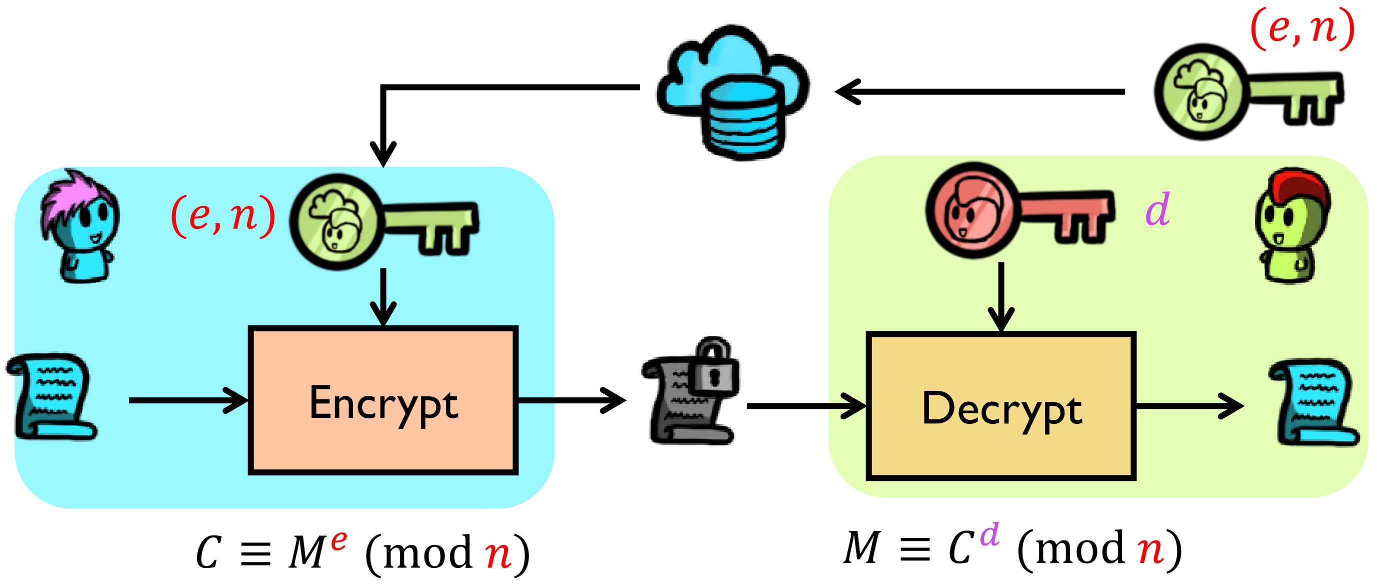
- This is textbook RSA, never do this!! (we'll see one of the reasons next).

Recall

- We want a mathematical operation that is very easy to do in one direction using e_k.
- And very hard to "un-do" given $e_k = (\mathrm{e}, \mathrm{n})$
- But easy to "un-do" given $d_k = d$.
- These are called "trapdoor functions"

Modular exponentiation given (e, n) is easy: $C \equiv M^e \pmod{n}$

Modular exponentiation is very hard to un-do without d.



Example of Textbook RSA

<u>Textbook RSA cheat sheet</u>

- 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: $M \equiv C^d \pmod{n}$.

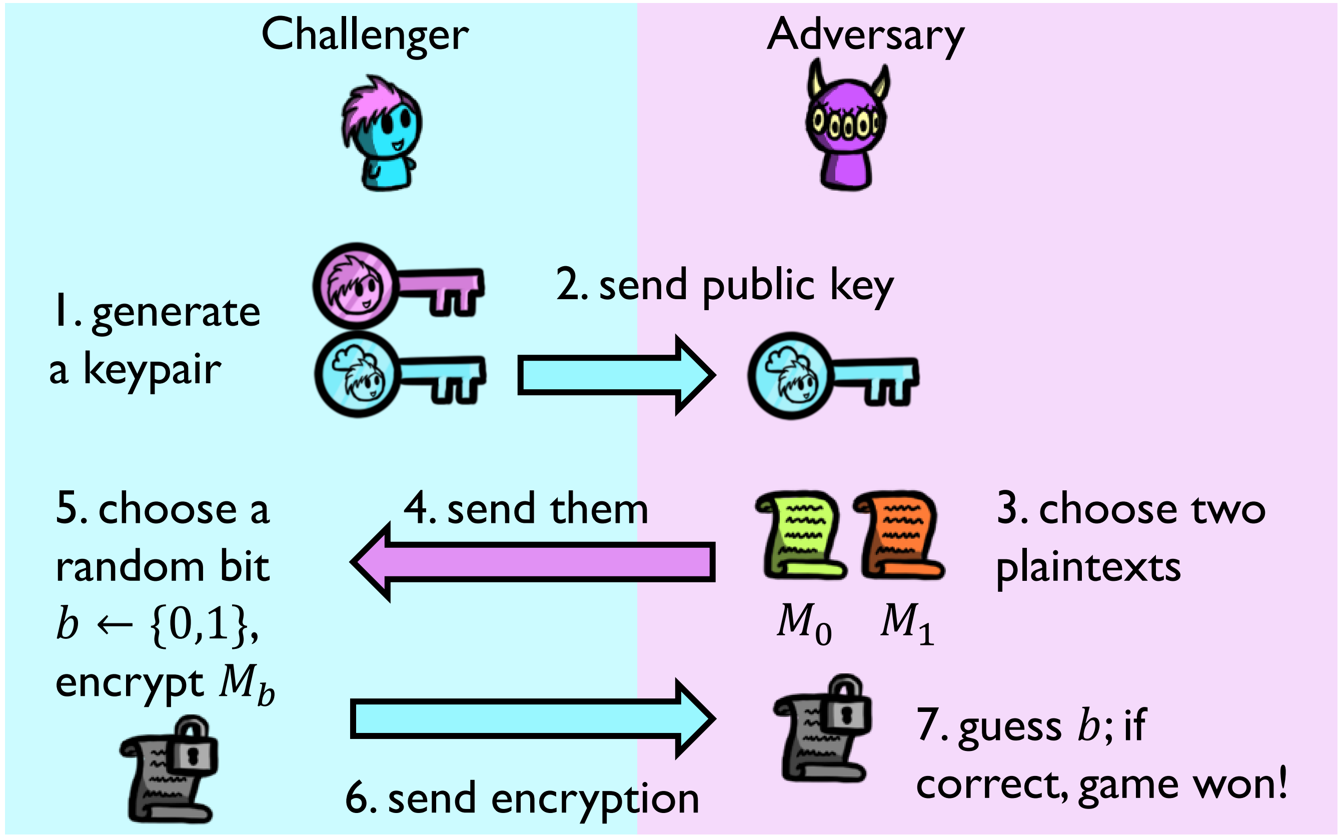
Assume $p = 53, q = 101, e = 139, d = 1459$.

1. Compute n. $n = 53 \cdot 101 = 5353$
2. Compute $C_1 = E_e(1011)$. Verify decryption works. $C_1 = 1011^{139} = 5253$
3. Compute $C_2 = E_e(4)$. Verify decryption works. $C_2 = 4^{139} = 324$
4. Compute $D_d(C_1 \cdot C_2)$. What is happening? Why? $(5253 \cdot 324)^{1459} = 4044$

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

<u>Textbook RSA cheat sheet</u>

- 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 keys:

e_A, e_B

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

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 4 – Integrity and Authentication

Module Outline

Preamble

- Cryptology
- Basic building blocks
- Kerckhoff's principle
- Cryptography attack models
- Confusion and Diffusion
- Symmetric vs Asymmetric crypto

Part 1: Ancient Crypto

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

Part 2: Symmetric crypto

- Definition
- Computational security
- Stream ciphers
  - Examples (A5/1, Salsa/Chacha)
- Block ciphers
  - Modes of operation
  - AES

Part 3: Public-key crypto

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

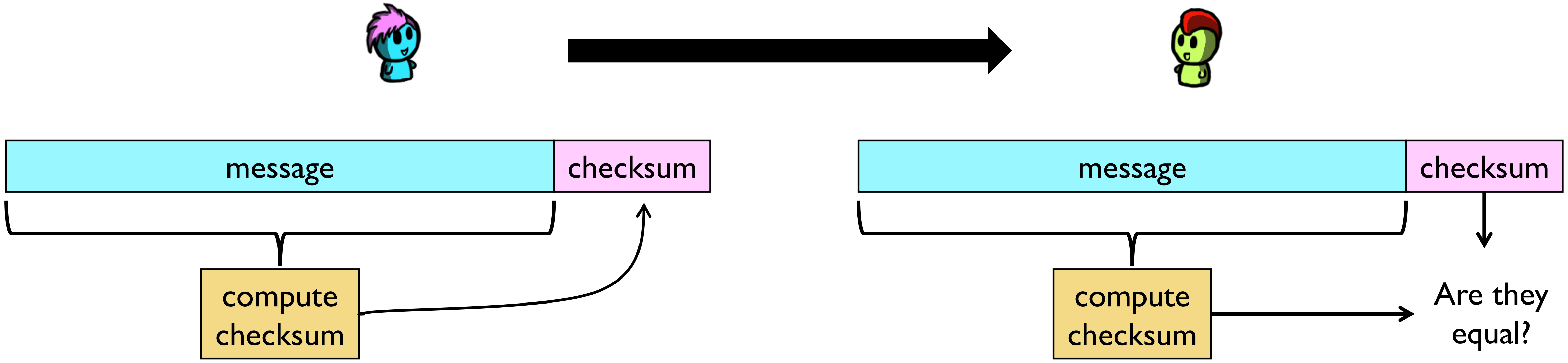
Part 4: Integrity and Authentication

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

Integrity in non-malicious settings

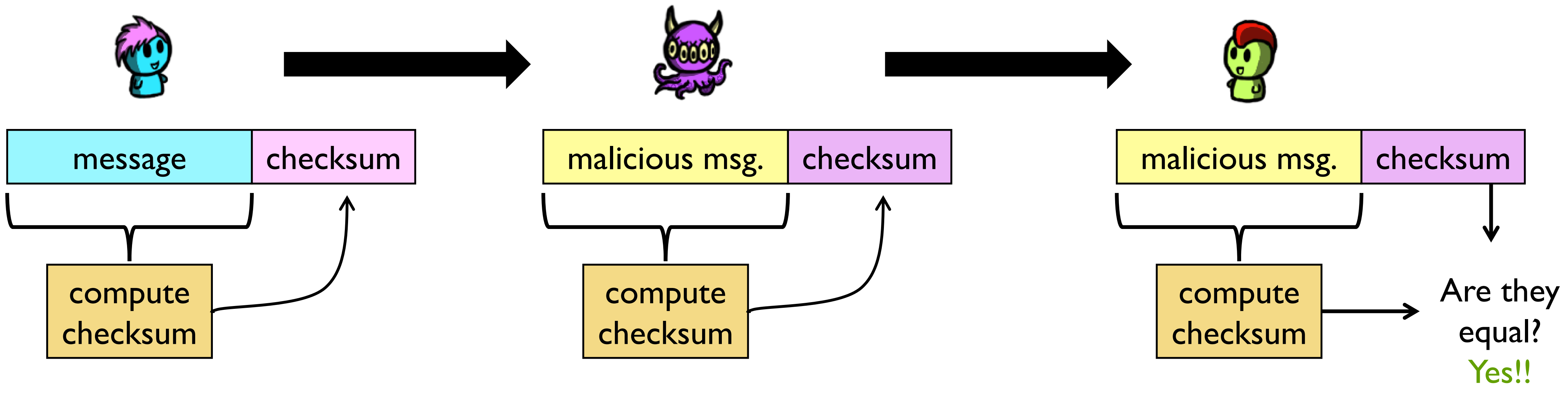
- How do we tell if a message has changed in transit?

- 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

$$\begin{tikzcd} x \arrow[r] & \boxed{h} \arrow[r] & y \end{tikzcd}$$

- 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)
  - …

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 x)

$$\begin{tikzcd} x \arrow[r] & \boxed{h} \arrow[r] & y \end{tikzcd}$$

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'.

$$\begin{tikzcd} x \arrow[r] & \boxed{h} \arrow[dr] & \\ & & y \\ x' \arrow[r] & \boxed{h} \arrow[ur] & \end{tikzcd}$$

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'.

$$\begin{tikzcd} x \arrow[r] & \boxed{h} \arrow[dr] & \\ & & y \\ x' \arrow[r] & \boxed{h} \arrow[ur] & \end{tikzcd}$$

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.

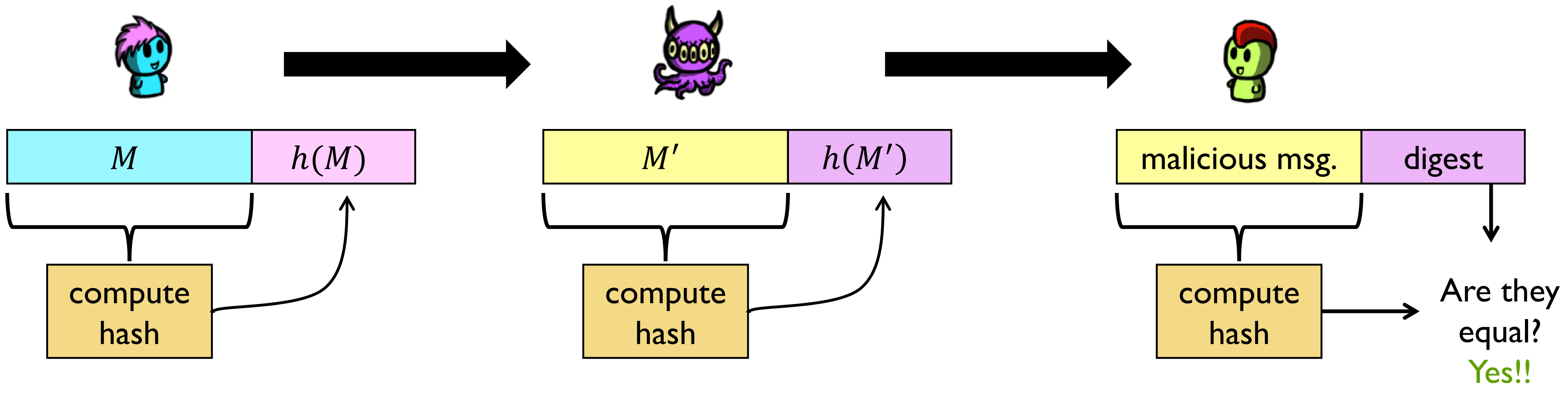
The birthday paradox

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*

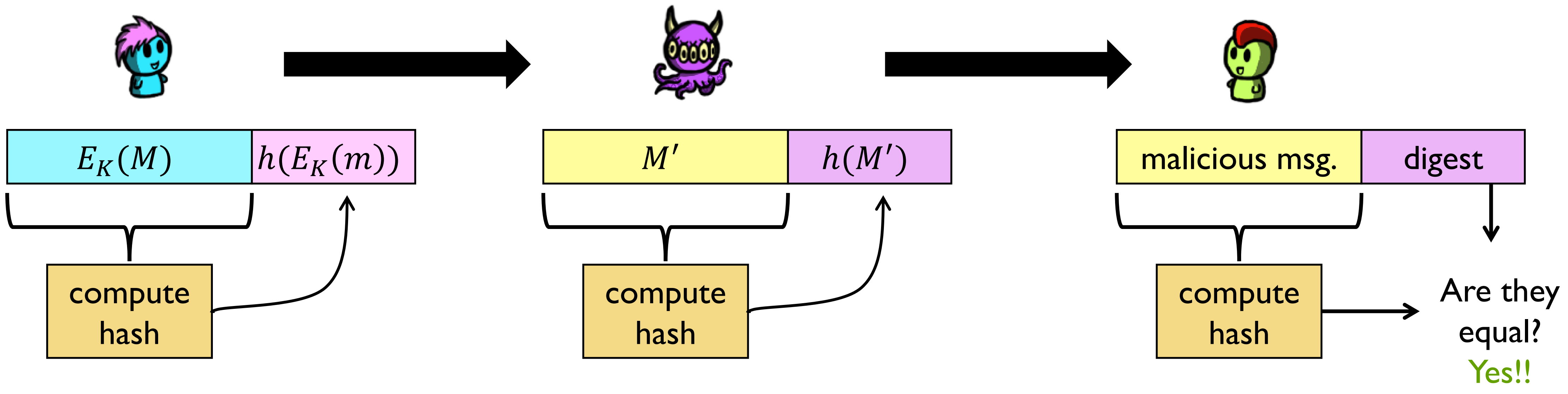
- For 23 people, the probability is larger than 50%
- For 40 people, it's almost 90%
- For 50 people, around 97%
- For 60 people, more than 99%

Let's use a hash function!



- Mallory can break integrity…

Let's use a hash function!



- 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"!

Authentication

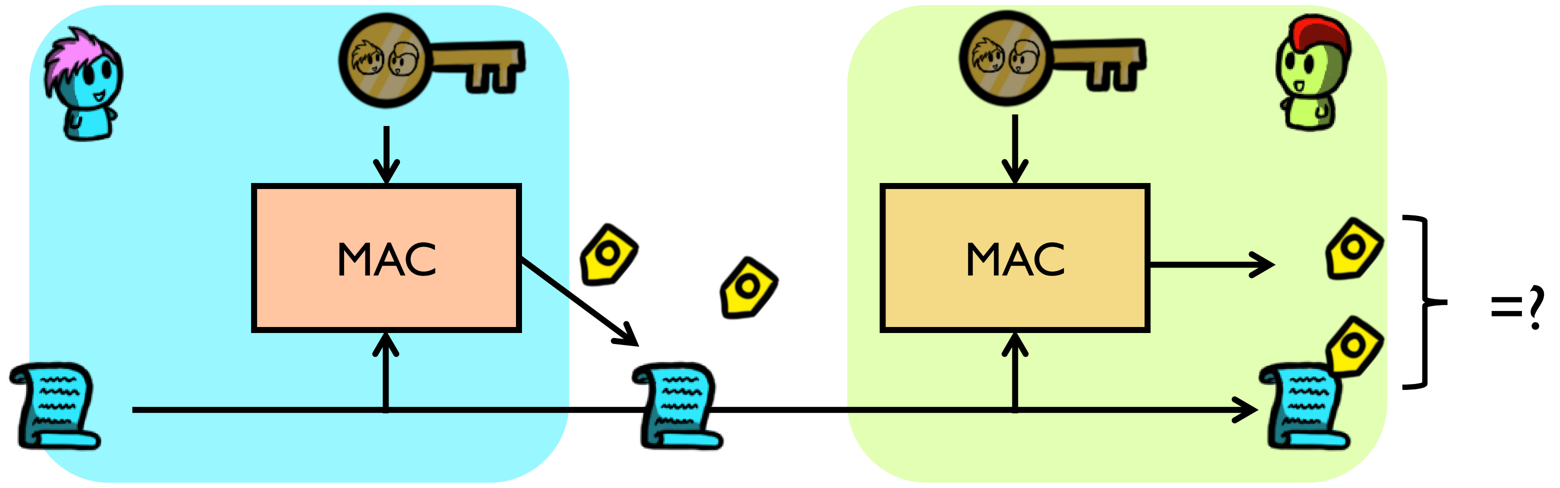
- Authentication and integrity go together in cryptography: we either provide both, or none.
  - If we cannot verify the "authenticity" of a sender, then that sender could be Mallory, and she could have modified the plaintext message…
- Authentication/integrity should not rely on how "reasonable" the 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

- MACs 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 $\mathrm{HMAC}_K(M)$ to denote the HMAC with key K for message M.
- What HMAC does (simplified):

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

- 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 $\left(E_K(\cdot), D_K(\cdot)\right)$, and a secret key K' for their MAC $(MAC_{K'}(\cdot))$. Concatenation is ∥. 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.

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

- 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.

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

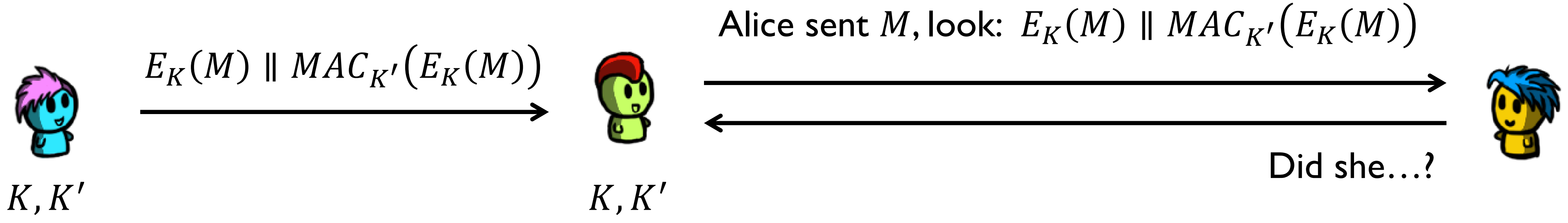
What is the right order?

- 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)

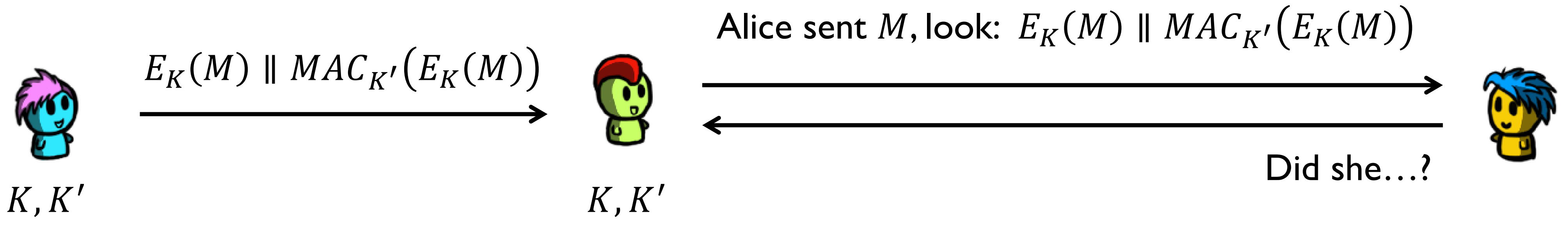
Repudiation


$E_K(M) \parallel MAC_{K'}(E_K(M))$
K, K'
K, K'
Alice sent M, look: $E_K(M) \parallel MAC_{K'}(E_K(M))$
Did she…?

- 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


$E_K(M) \parallel MAC_{K'}(E_K(M))$
Alice sent M, look: $E_K(M) \parallel MAC_{K'}(E_K(M))$
Did she…?
K, K'
K, K'

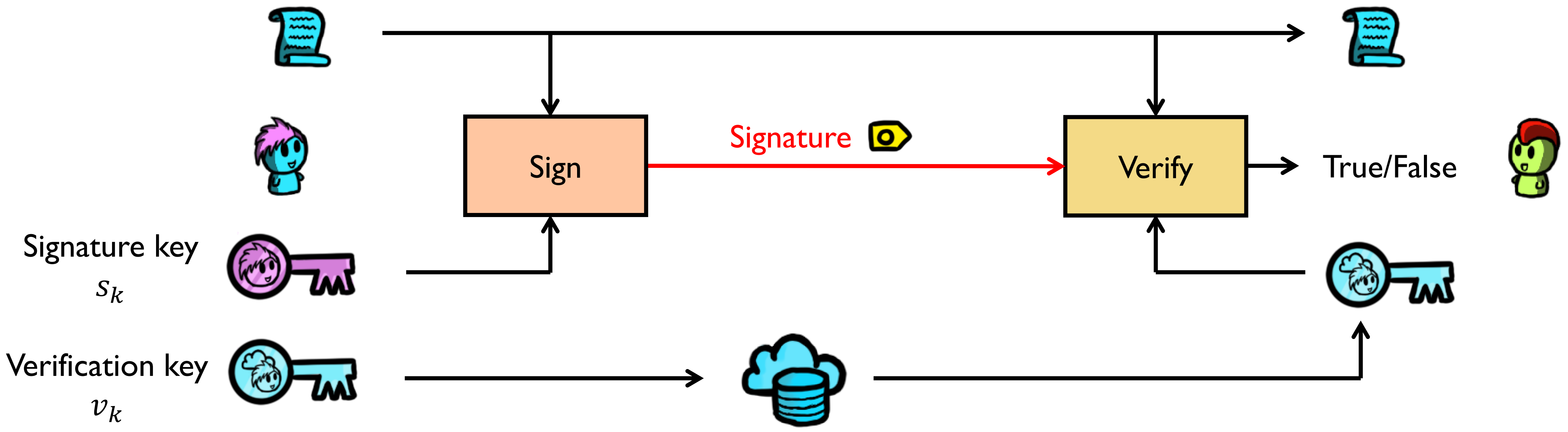
- Alice can just claim that Bob made up the message M, and calculated the MAC himself.
- This is called repudiation, and we sometimes want to avoid it.
- Some interactions should be repudiable.
  - Private conversations
- Some interactions should be non-repudiable.
  - Electronic commerce

Digital signatures

- MACs are the "symmetric" version of authentication.
- Digital signatures are the "asymmetric" version of authentication.

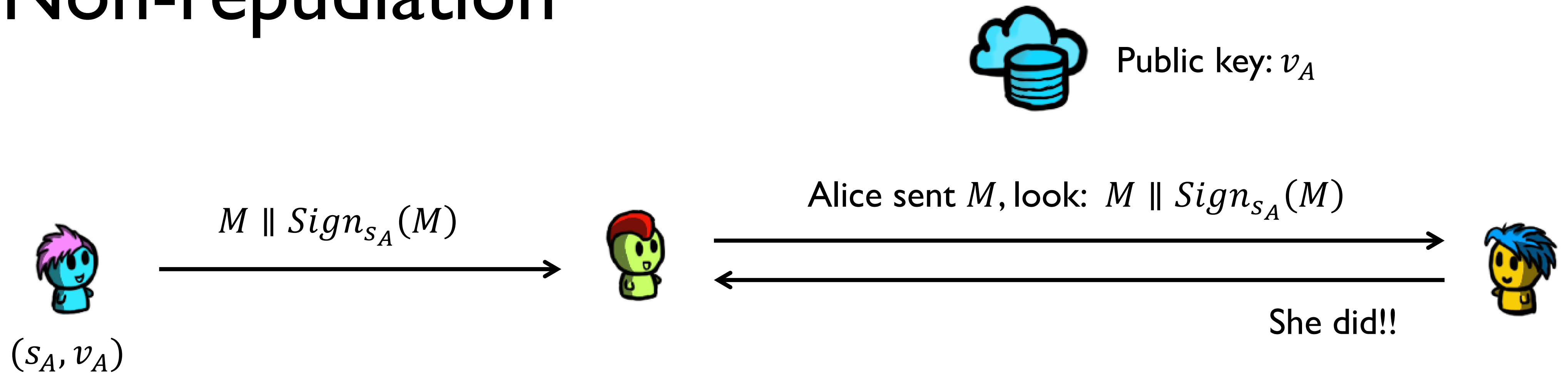
- Remember, to encrypt/decrypt in public-key cryptography
  - Alice *encrypts* with Bob's *public encryption* key.
  - Bob *decrypts* with his *private decryption* key.
- To sign and verify in public-key cryptography:
  - Alice *signs* with her *private signature* key.
  - Bob *verifies* the message with Alice's *public verification* key.

Digital Signatures



- If you need encryption, you also need to "do" encryption.

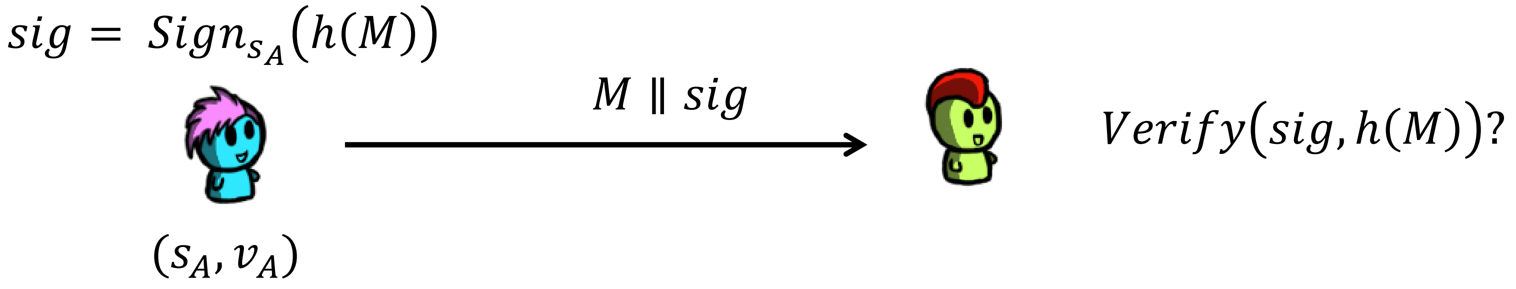
Non-repudiation


Public key: v_A
$M \parallel Sign_{s_A}(M)$
Alice sent M, look: $M \parallel Sign_{s_A}(M)$
She did!!
(s_A, v_A)

- 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.



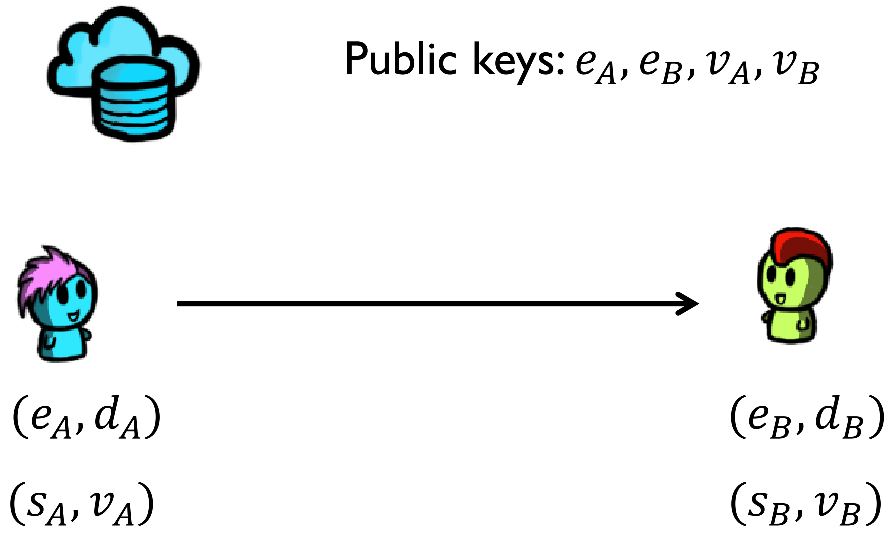
- Remember that authenticity and confidentiality are separate; if we want both, we have to do both.

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?



Encryption/decryption functions (symmetric or public-key)

$E_{key}(\cdot)$ $D_{key}(\cdot)$

Authentication functions:

$MAC_{key}(\cdot)$ $Sign_{key}(\cdot)$ $Verify_{key}(\cdot, tag)$

Assume these two hash before sign/verify

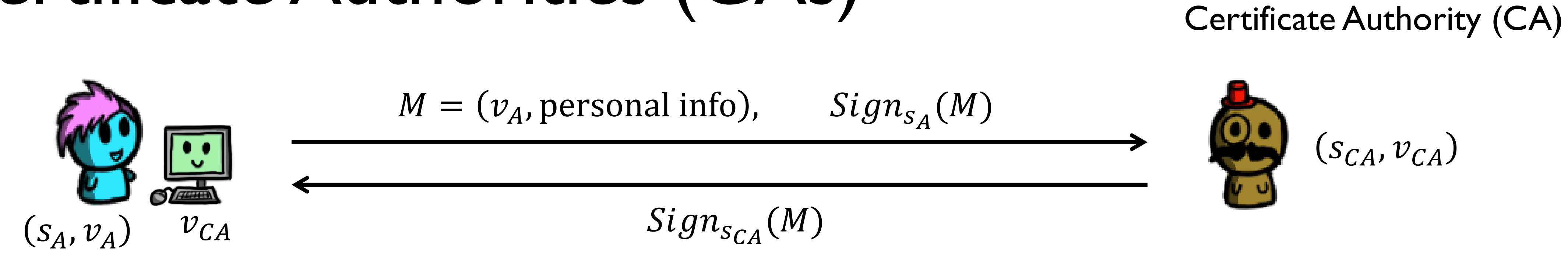
Relationship between key pairs

- 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 k.

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.

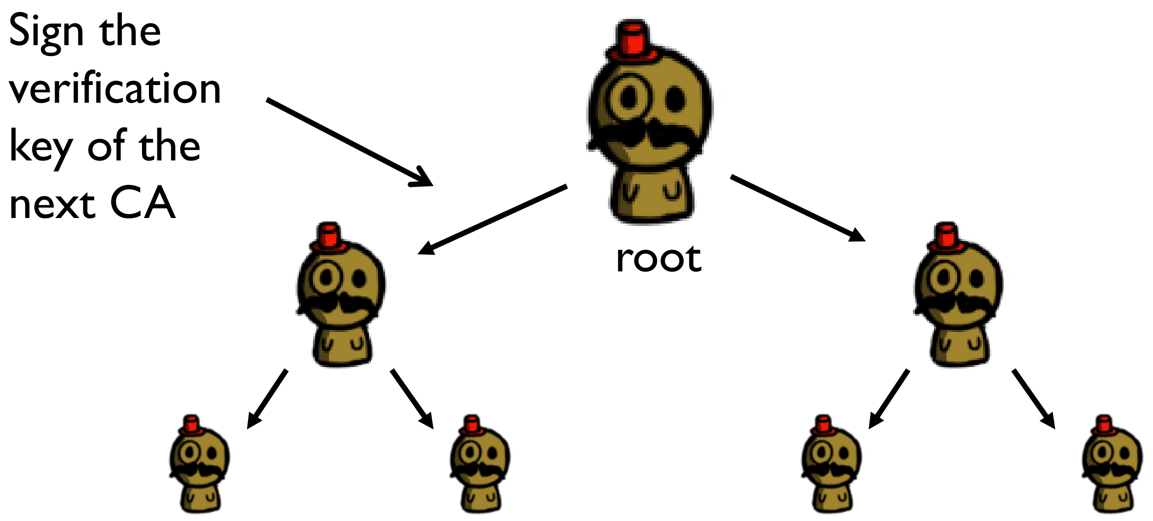
Certificate Authorities (CAs)


Certificate Authority (CA)
$M = (v_A, \text{personal info}), \quad Sign_{s_A}(M)$
$Sign_{s_{CA}}(M)$
(s_A, v_A)
v_{CA}
(s_{CA}, v_{CA})

- A CA is a trusted third party who keeps a directory of people's (and organizations') verification keys.
- Alice generates a (s_A, v_A) key pair, and sends the verification key and personal information, both signed with Alice's signature key, to the CA.
- The CA ensures that the personal information and Alice's signature are correct.
- The CA generates a certificate consisting of Alice's personal information, as well as her verification key. The entire certificate is signed with the CA's signature key.
- https://letsencrypt.org has changed the game. Most web traffic is now encrypted.

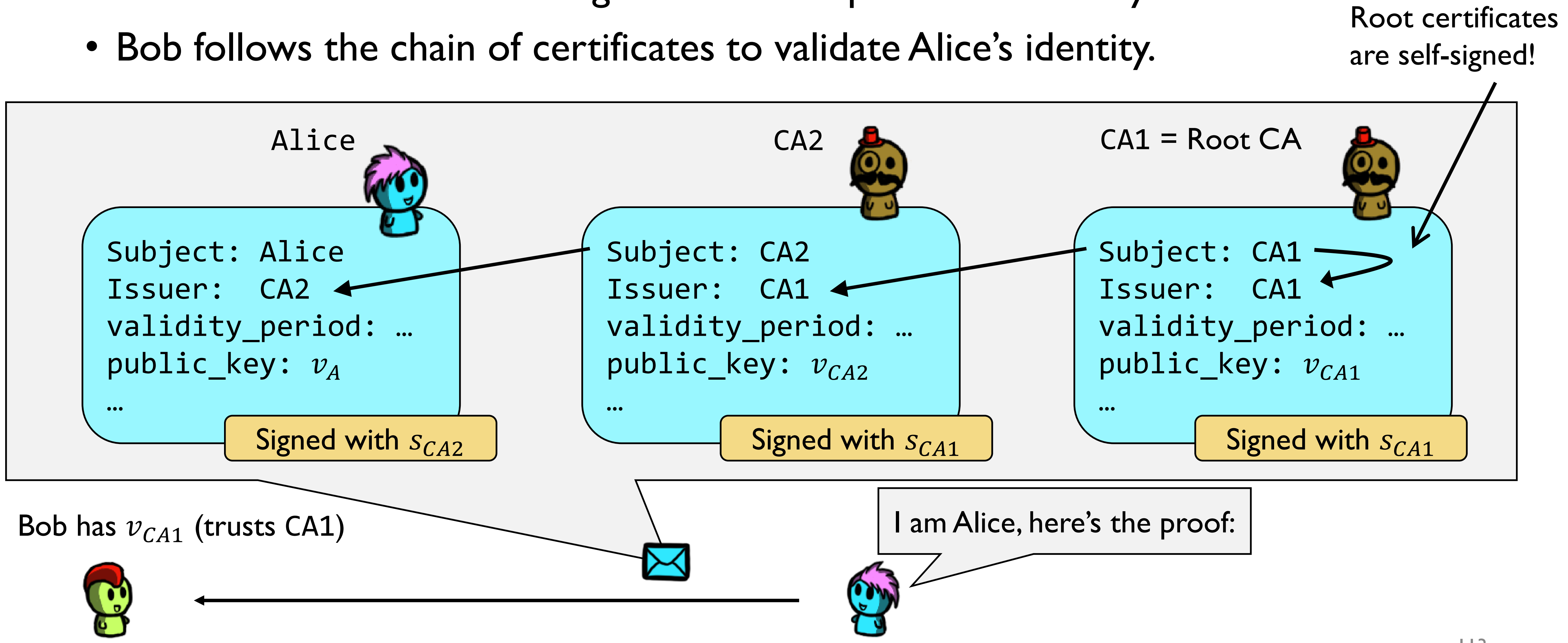
Certificate Authorities (CAs)

- Everyone is assumed to have a copy of the CA's verification key (v_{CA}), so they can verify the signature on the certificate.
- There can be multiple levels of certificate authorities; level n CA issues certificates for level $n + 1$ CAs. This is the Public-Key Infrastructure (PKI).
- We only need the verification key of the root CA to verify the certificate chain!



Chain of certificates

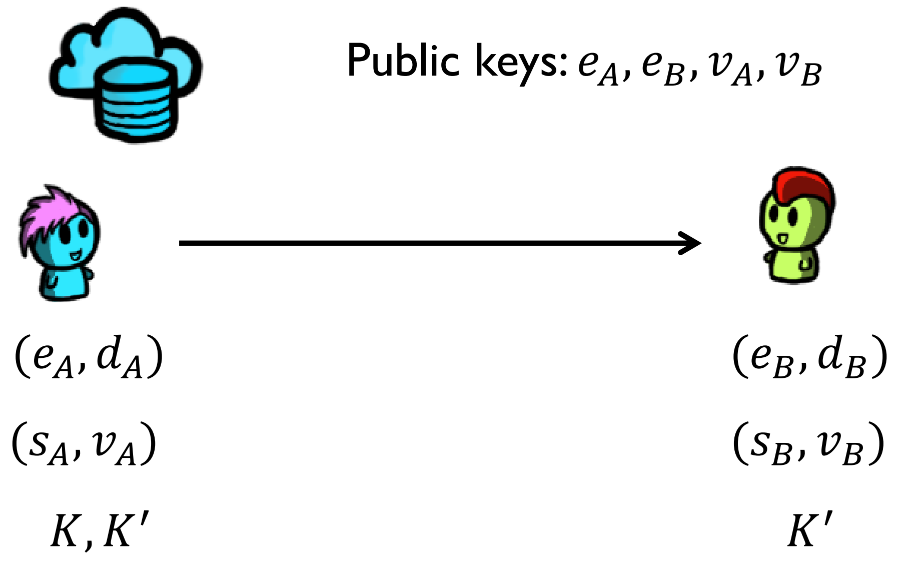
- Alice sends Bob the following certificate to prove her identity.
- Bob follows the chain of certificates to validate Alice's identity.



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!

Knowledge check III:



For each of these messages from Alice to Bob, where the data M is meant to be secret. For each message, indicate:

1. Can Bob learn/decrypt M if the message is not modified in transit?
2. Does it provide confidentiality of M?
3. Does it provide authentication? (then, if yes)
    1. does it provide repudiation?
    2. can Bob verify authenticity first?

For each case: assume Bob and Eve/Mallory know the "format" of the message.

Message	Decrypt?	Conf?	Auth?	Repudiation?	Verify first?
$E_{e_A}(M) \parallel Sign_{s_A}(M)$					
$E_K(M) \parallel E_{e_B}(K)$					
$E_K(M) \parallel E_{e_B}(K) \parallel MAC_K(M)$					
$E_K(M) \parallel K \parallel MAC_K(M)$					
$E_{K'}(M) \parallel MAC_{K'}(M)$					

Knowledge check III:

(e_A, d_A)
(s_A, v_A)
K, K'

(e_B, d_B)
(s_B, v_B)
K'

Message	**Decrypt?**	**Conf?**	**Auth?**	**Repudiation?**	**Verify first?**
$E_{e_B}(M) \parallel Sign_{s_A}\left(E_{e_B}(M)\right)$					
$E_{e_B}\left(h(M)\right)$					
$M \parallel Sign_{s_A}(M)$					
$M \parallel E_{e_B}(K) \parallel MAC_K(M)$					

Now you must design a message such that…

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

Module 2: recap

- Remember Kerckhoff's principle!
- Confusion and diffusion.
- Ancient crypto: learn why they seem to work well, but how they can be broken
- Symmetric crypto: same key to encrypt/decrypt, requires shared secret, it is fast, secure with 128-bit key
  - Stream ciphers: they XOR a keystream with the plaintext, bit by bit, only confusion
    - Understand why we add a nonce and why it is not encrypted
  - Block ciphers: they split the plaintext in blocks, encrypt each block (confusion and diffusion), connect the blocks (understand why ECB is bad, and why CBC and CTR are better).
    - Understand the IV, why we don't encrypt it

Module 2: recap

- Public-key crypto: different key to encrypt/decrypt, does not require shared secret, it is slow, requires larger key sizes than symmetric crypto
  - Textbook RSA as an example: understand how it works, the toy example we saw in the classroom, and the shortcomings that we saw.
- Hybrid crypto: this is the common approach, understand it!
- Hash functions: understand the 3 properties, which ones are easier to break, which ones are better to provide.
- Authentication:
  - MACs: symmetric, repudiable
  - Signatures: asymmetric, non-repudiable
- The key management problem: we need a “trusted” verification key to at least get some authenticity/integrity over the Internet.
  - CAs and chain of certificates