



# Data Security

# Logistics!

- In-person class resumes next week
  - Live Zoom lecture should still be available, along with posted recordings and slides
- Schedule: now in the syllabus on Canvas
- First recitation is on Wednesday
- Quizzes start this week
  - Assigned start of class Tuesday, due one week later
  - Answers may not be covered until Thursday
  - Includes material from both lecture and recitation

# Logistics!

- ◎ Waitlist: There are still a lot of people waiting
  - If you are planning on dropping, waitlisted folks need to know by Wednesday

# Jupyter notebooks?

1. Do you all know how to use Jupyter notebooks?

# Jupyter notebooks?

1. Do you all know how to use Jupyter notebooks?
2. What is the best way of distributing a Jupyter notebook?

*(I can upload them to Canvas and have you download and run them on coding.csel.io, does that work?)*

# What digital data do we use?



# What digital data do we use?



<https://www.theguardian.com/world/2018/jan/28/fitness-tracking-app-gives-away-location-of-secret-us-army-bases>

# What digital data do we use?



# What digital data do we use?



<https://mercedescct.com/blog/how-to-use-the-mercedes-benz-brake-hold-function/>

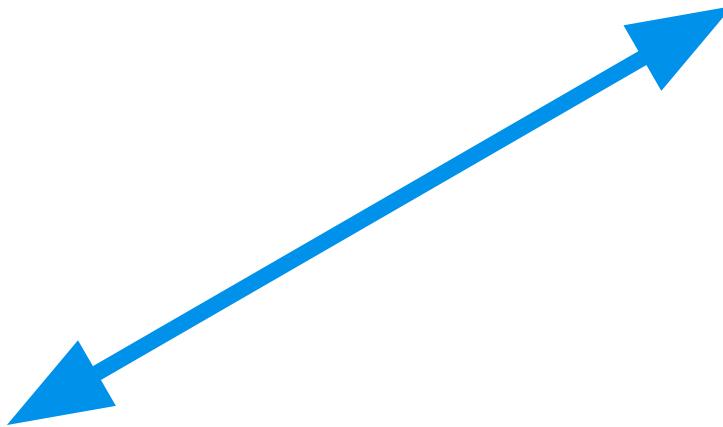
# What digital data do we use?



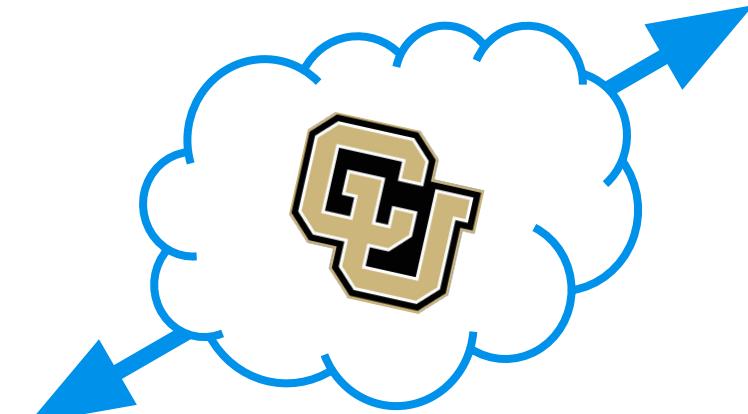
<https://mercedescct.com/blog/how-to-use-the-mercedes-benz-brake-hold-function/>

# Data security

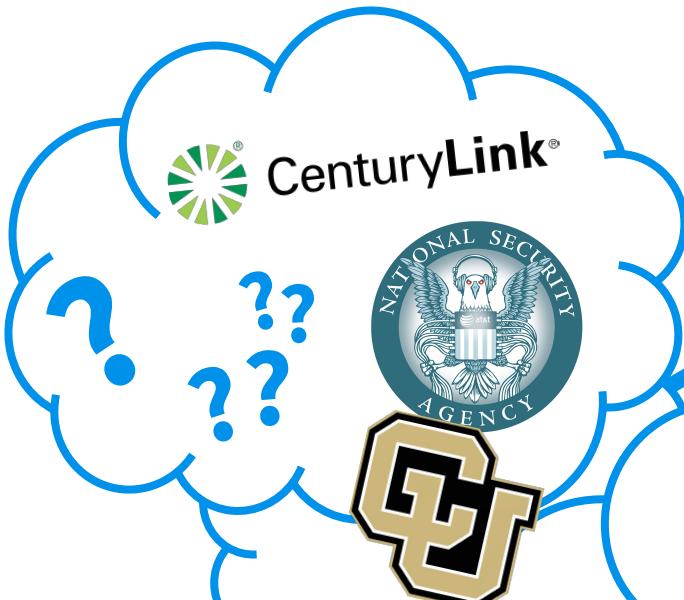
- ◎ **Data security:** Protecting data from attackers, while ensuring it still remains available
- ◎ Foundation of most other security topics

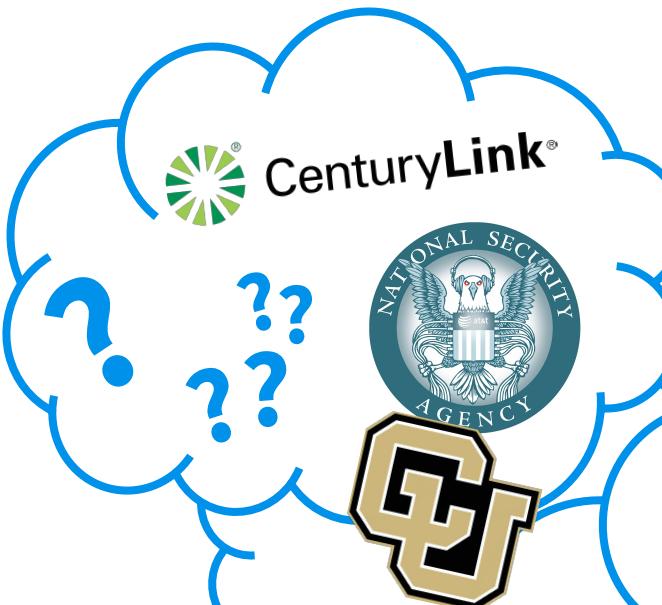












# Alice and Bob

Alice

Bob

*Hello! What is my bank balance right now?*

*You have \$1,436 in your account.*

*Great. Can you transfer \$30 to Carol?*

# Alice and Bob

*Hello! What is my bank balance right now?*

Alice

*You have \$1,436 in your account.*

Bob

Eve

*Great. Can you transfer \$30 to Carol?*

# Alice and Bob

*Hello! What is my bank balance right now?*

Alice

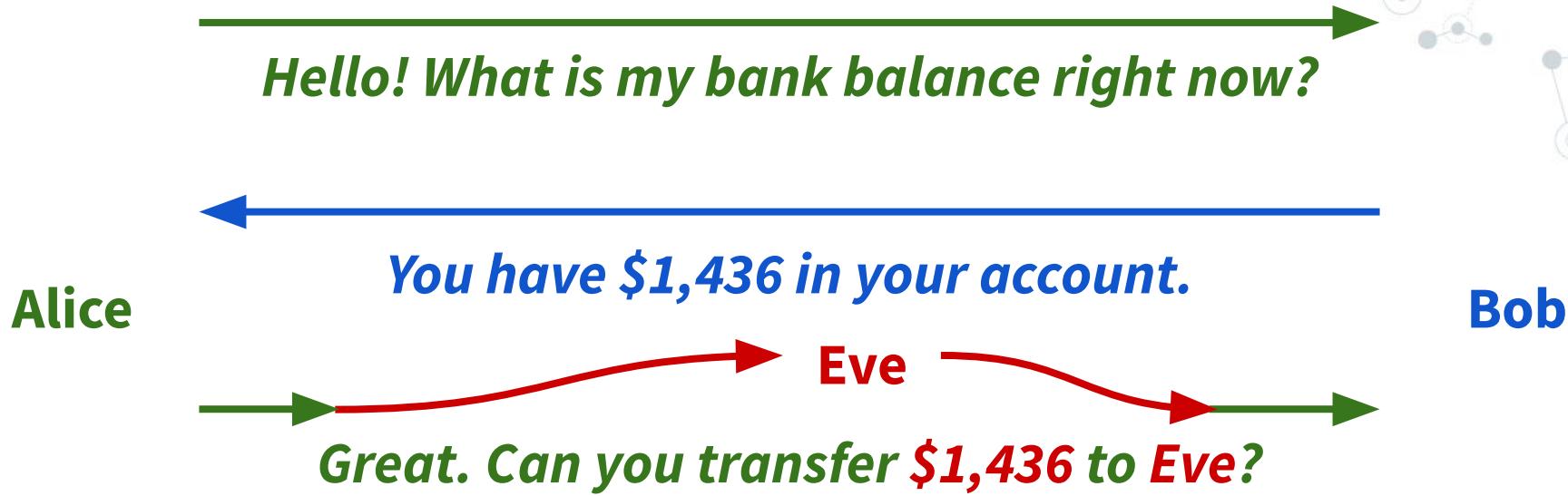
*You have \$1,436 in your account.*

Bob

Eve

*Great. Can you transfer \$1,436 to Eve?*

# Alice and Bob



**Man-in-the-Middle (MitM) attack:**  
Reading or modifying messages like this



# [Man-in-the-middle demo]

# Recent examples

*OPT-OUTS DON'T MEAN MUCH THESE DAYS —*

## Verizon overrides users' opt-out preferences in push to collect browsing history

Verizon renamed scanning program and enrolled customers who previously opted out.

JON BRODKIN · 12/7/2021, 3:15 PM



Enlarge

<https://arstechnica.com/information-technology/2021/12/verizon-ignored-users-previous-opt-outs-in-latest-push-to-scan-web-browsing/>

# Alice and Bob



## Options:

- ◎ Don't communicate sensitive information



# Alice and Bob

## Options:

- Don't communicate sensitive information
- Use another (safe) channel

# Alice and Bob

## Options:

- Don't communicate sensitive information
- Use another (safe) channel
- Alter the message so Eve cannot understand it

# Alice and Bob

## Options:

- Don't communicate sensitive information
- Use another (safe) channel
- Alter the message so Eve cannot understand it

# Alice and Bob

## Options:

- Don't communicate sensitive information
- Use another (safe) channel
- Alter the message so Eve cannot understand it

*This is not always possible*

# Alice and Bob

## Options:

- Don't communicate sensitive information
- Use another (safe) channel
- Alter the message so Eve cannot understand it**

*This is not always possible*

**Encryption: Altering a message so it can't be understood**

# Encryption

*Hello! What is my bank balance right now?*

Alice

*You have \$1,436 in your account.*

Bob

*Great. Can you transfer \$30 to Carol?*

# Encryption

Alice

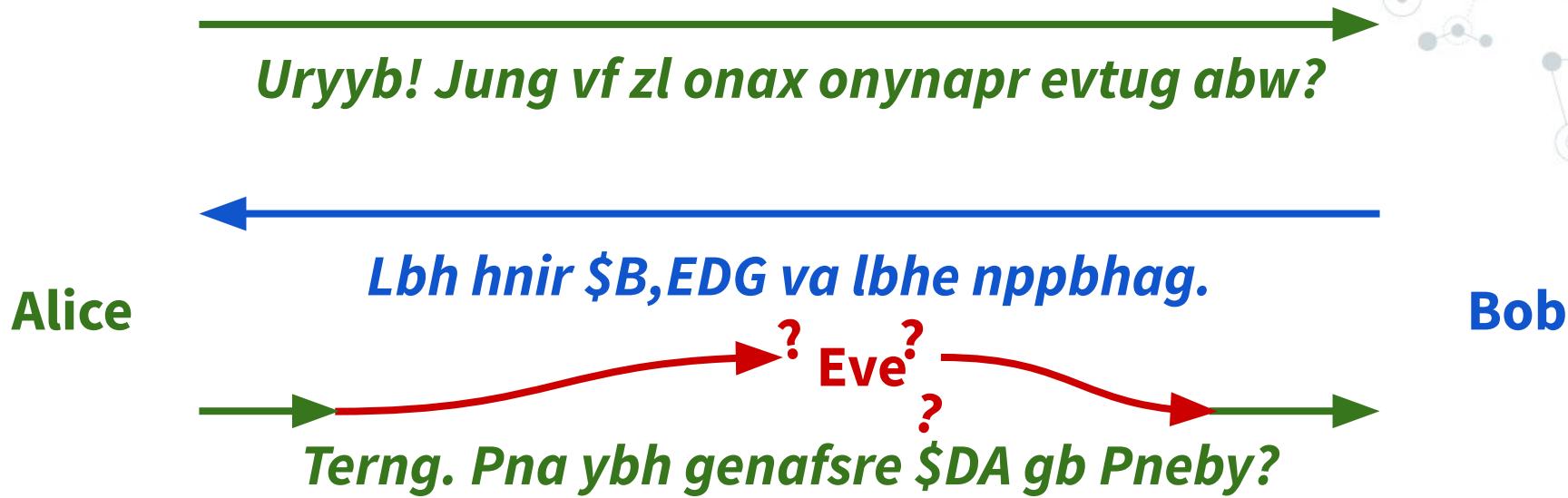
Bob

*Uryyb! Jung vf zl onax onynapr evtug abw?*

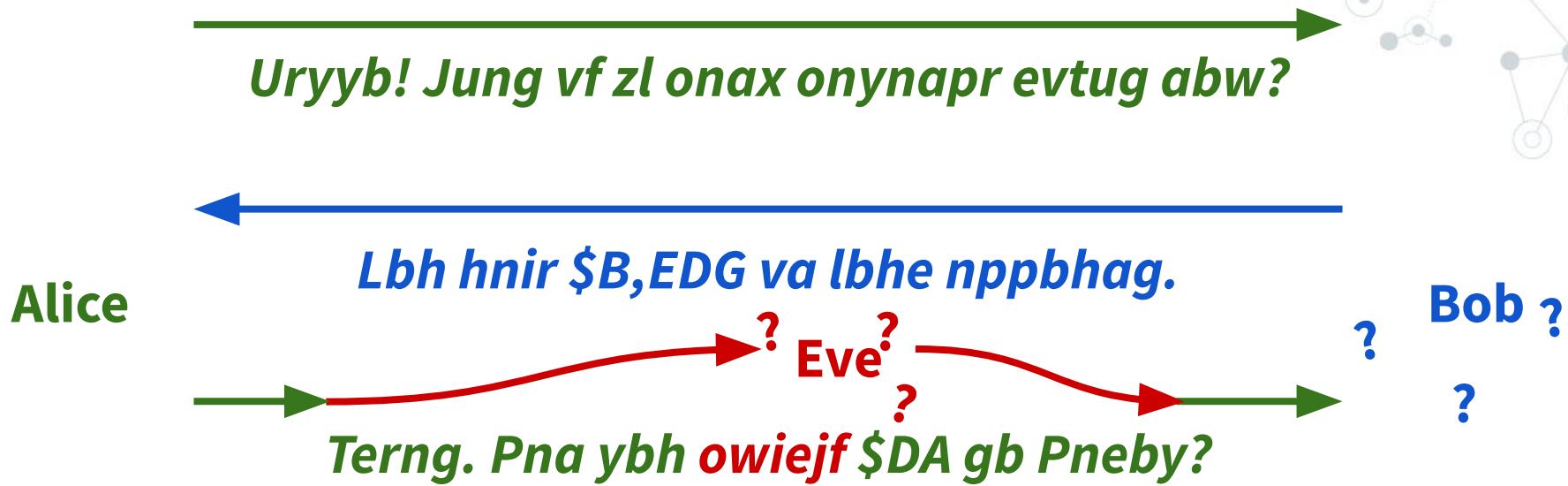
*Lbh hnir \$B,EDG va lbhe nppbhag.*

*Terng. Pna ybh genafsre \$DA gb Pneby?*

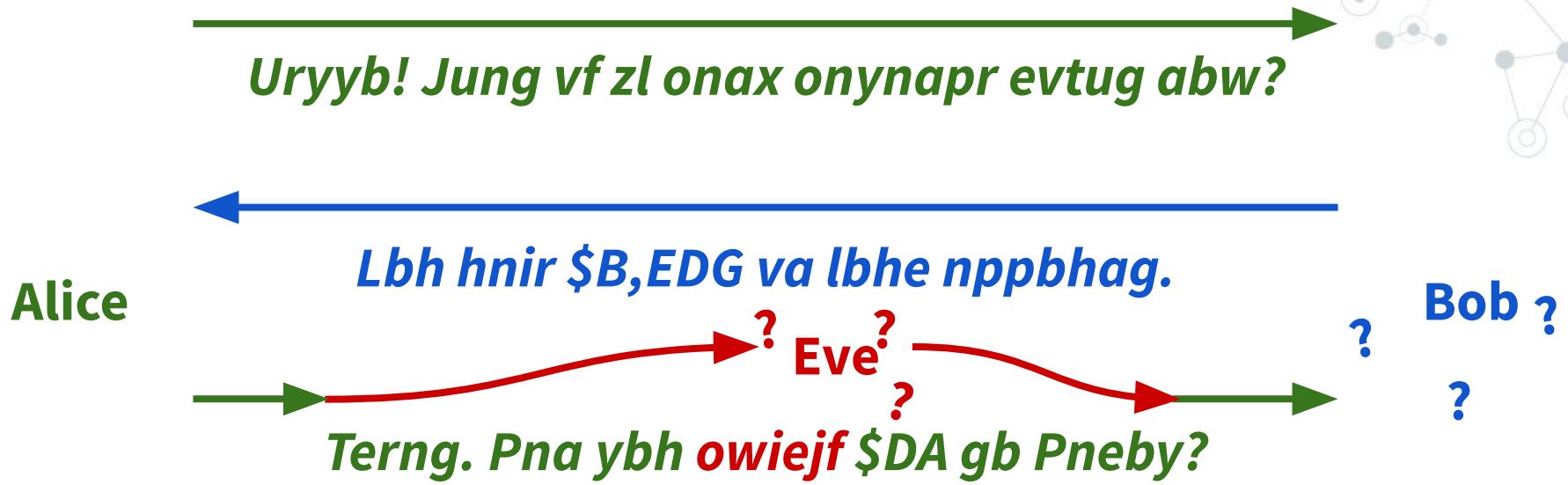
# Encryption



# Encryption

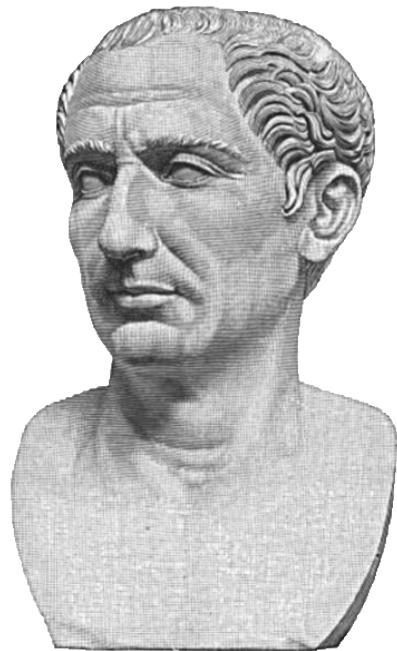


# Encryption



- Primarily protects **confidentiality**
- Does not guarantee **integrity**
- Integrity checks can be added, we will discuss those later

# Caesar Cipher

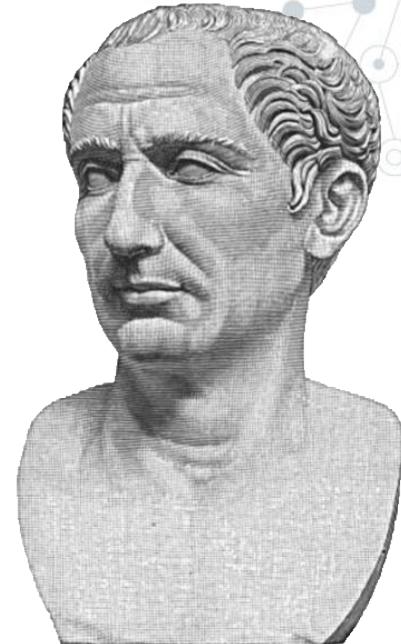


# Caesar Cipher

1. Pick a secret number from 1 to 25 in advance
2. Sender: Shift each letter forward by the secret

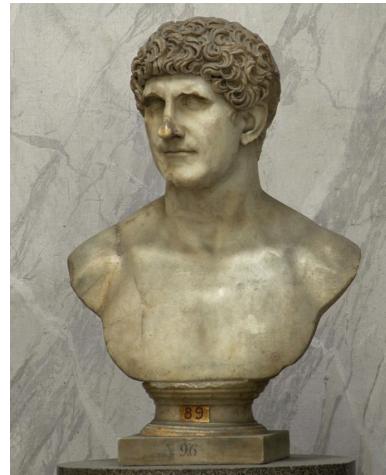
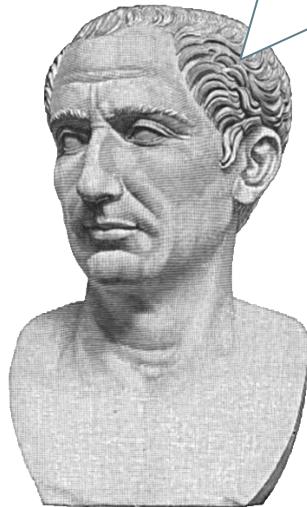
*Send the shifted message*

3. Receiver: Shift each letter back by the secret



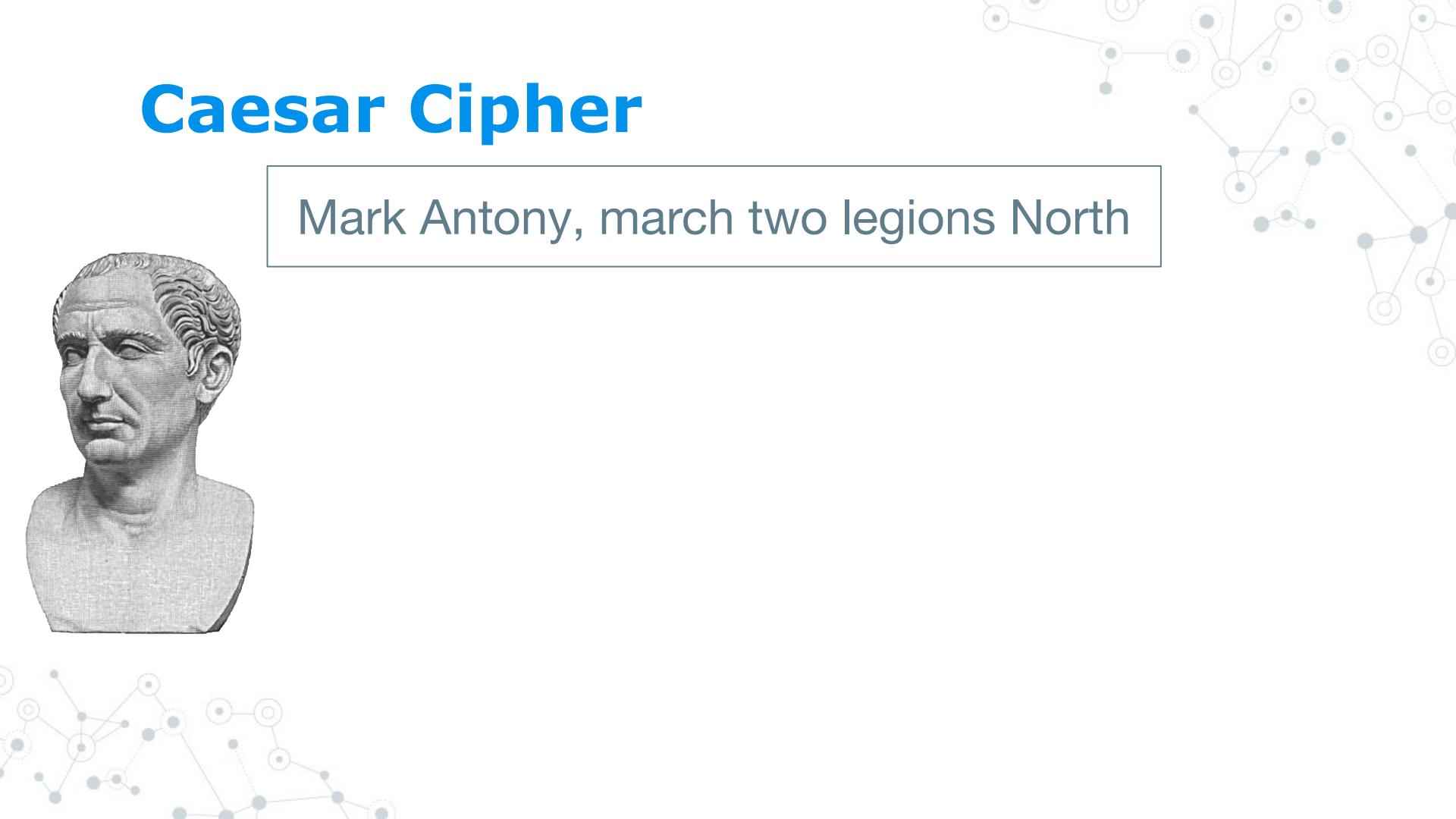
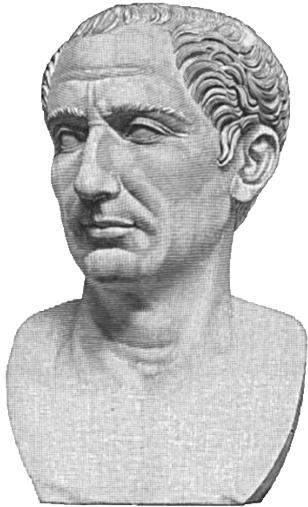
# Example

The secret number is **three**



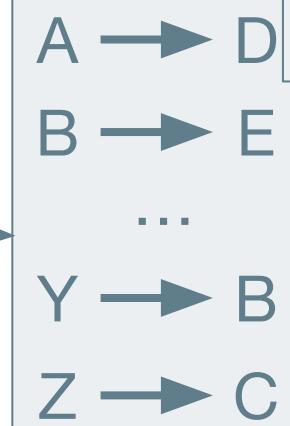
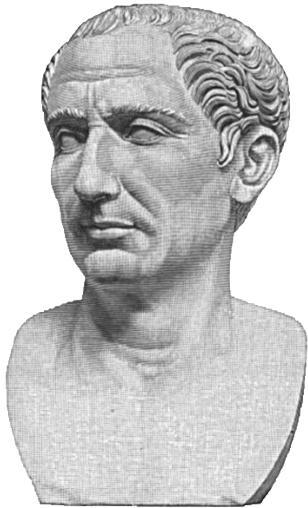
# Caesar Cipher

Mark Antony, march two legions North



# Caesar Cipher

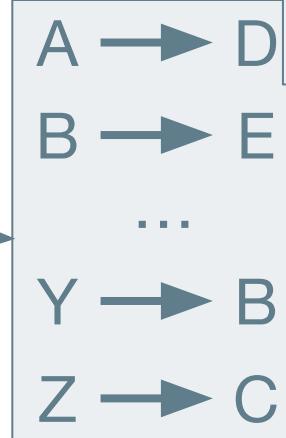
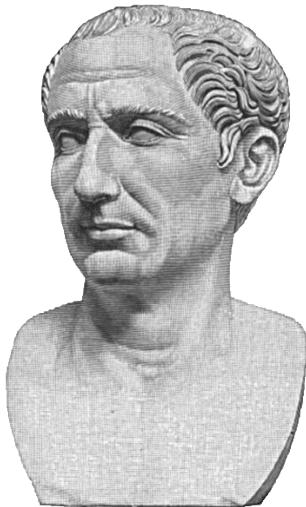
Mark Antony, march two legions North



Shift **forward** by three  
(wrapping around if needed)

# Caesar Cipher

Mark Antony, march two legions North



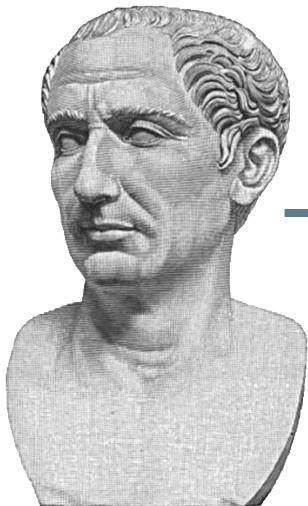
Shift **forward** by three  
(wrapping around if needed)

A callout box to the right of the mapping table, containing instructions for applying the cipher rule.

Pdun Dqwrqb, pdufk wzf ohjlrqv Qruwk

The final encrypted text, resulting from applying the Caesar cipher rule to the original message "Mark Antony, march two legions North".

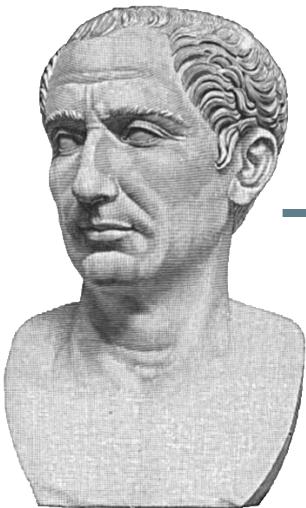
# Caesar Cipher



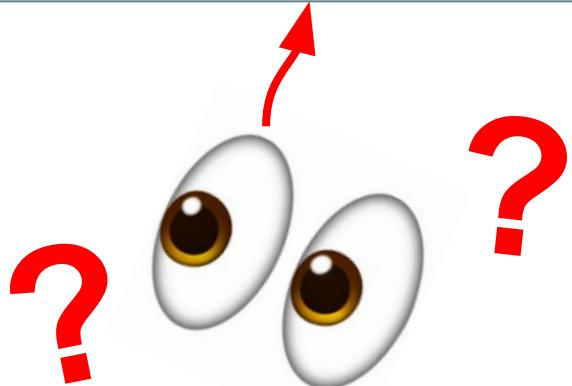
Pdun Dqwrqb, pdufk wzr ohjlrqv Qruwk



# Caesar Cipher



Pdun Dqwrqb, pdufk wzc ohjlrqv Qruwk



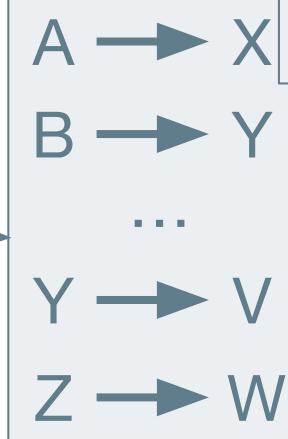
# Caesar Cipher

Pdun Dqwrqb, pdufk wzr ohjlrqv Qruwk



# Caesar Cipher

Pdun Dqwrqb, pdufk wzr ohjlrqv Qruwk



Shift **backward** by three  
(wrapping around if needed)



Mark Antony, march two legions North

# Jargon

- ◎ **Plaintext:** The original message
  - e.g. “Mark Antony, march two legions North”
- ◎ **Ciphertext:** The encrypted message
  - e.g. “Pdun Dqwrqb, pdufk wzr ohjlrqv Qruwk”

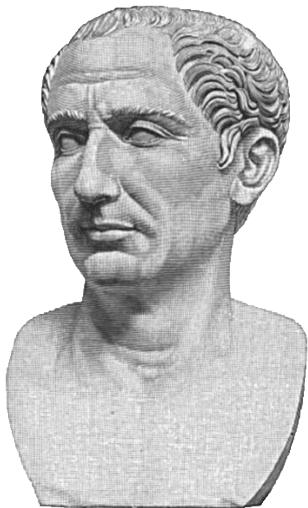
# Jargon

- **Plaintext:** The original message
  - e.g. “Mark Antony, march two legions North”
- **Ciphertext:** The encrypted message
  - e.g. “Pdun Dqwrqb, pdufk wzr ohjlrqv Qruwk”
- **Encrypt:** Turn *plaintext* into *ciphertext*
- **Decrypt:** Turn *ciphertext* into *plaintext*

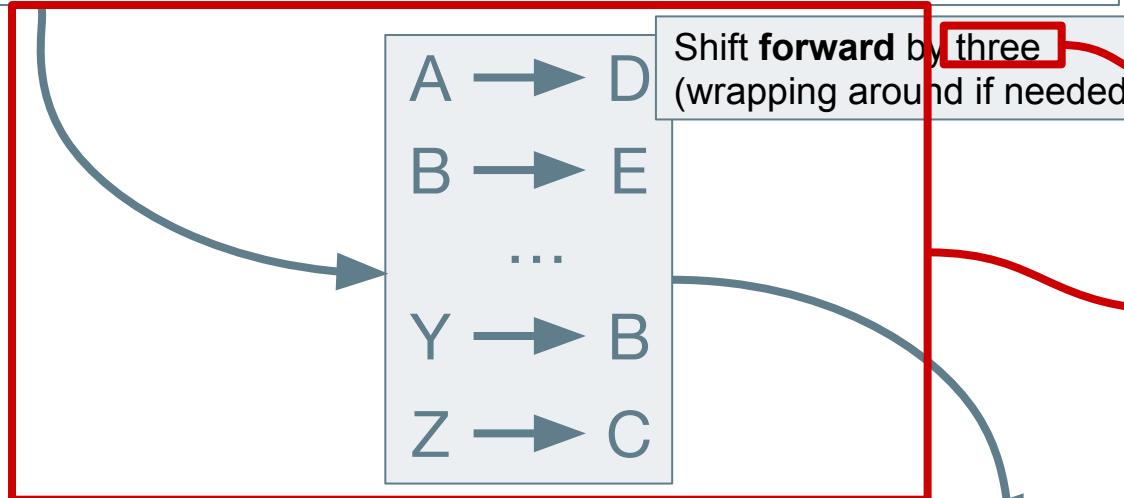
# Jargon

- **Plaintext:** The original message
  - e.g. “Mark Antony, march two legions North”
- **Ciphertext:** The encrypted message
  - e.g. “Pdun Dqwrqb, pdufk wzr ohjlrqv Qruwk”
- **Encrypt:** Turn *plaintext* into *ciphertext*
- **Decrypt:** Turn *ciphertext* into *plaintext*
- **Key:** Any secret that is used to *encrypt* or *decrypt*  
*Named because it “locks” or “unlocks” the message*

# Caesar Cipher



Mark Antony, march two legions North



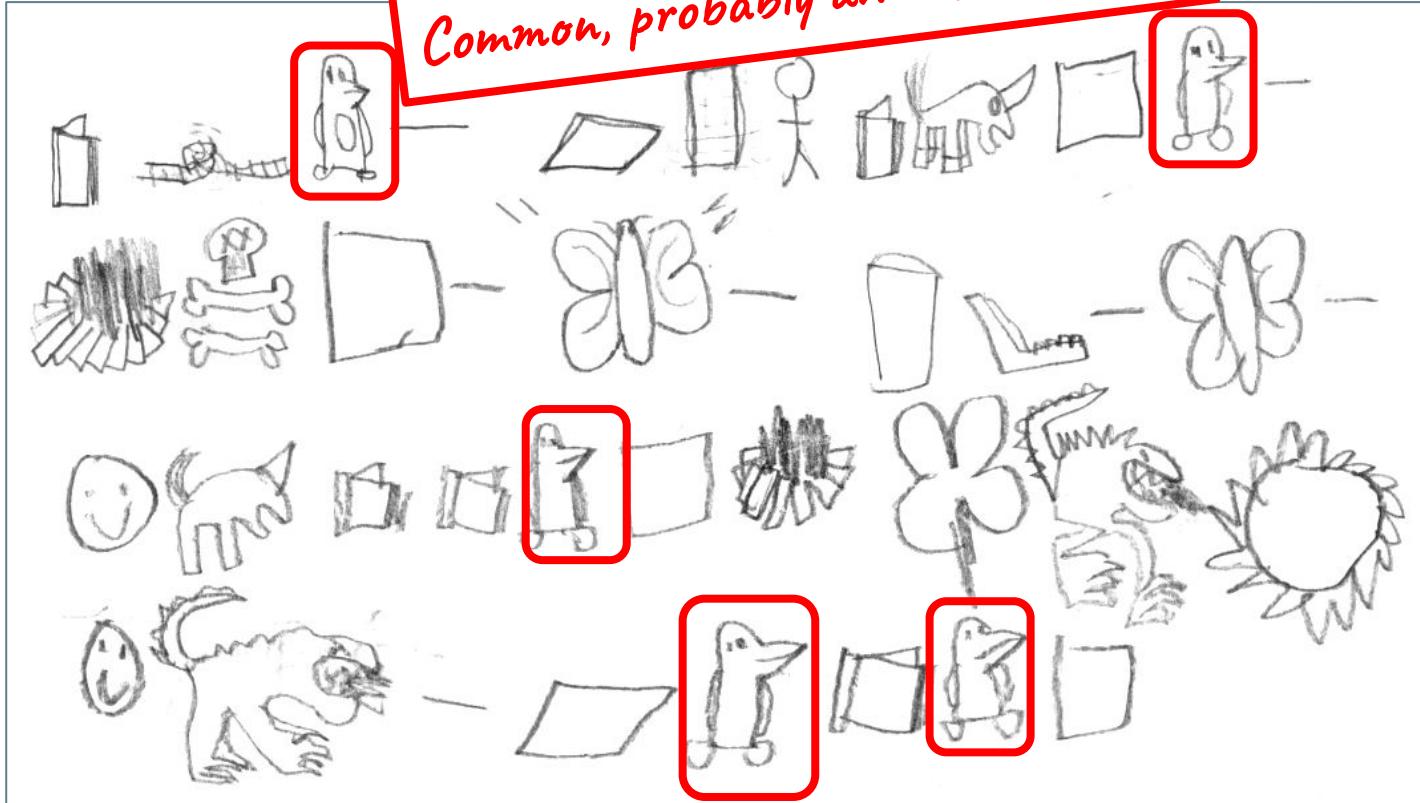
Pdun Dqwrqb, pdufk wzk ohjlrqv Qruwk

# Problems

- ◎ Can brute-force all 25 keys
  - **Keyspace:** Number of possible keys
    - 1: Octm Cpvqpa, octej vyq ngikqpu Pqtvj
    - 2: Nbsl Boupoz, nbsdi uxpmfhjpot Opsui
    - 3: **Mark Antony, march two legions North**
  - Key must be agreed on beforehand
    - **Shared secret:** A secret both parties know
  - Patterns exist in ciphertext



Common, probably an "e", "a", or "t"



Common, probably an "e", "a", or "t"

Must be "a" or "I"



Common, probably an "e", "a", or "t"



Must be "a" or "I"



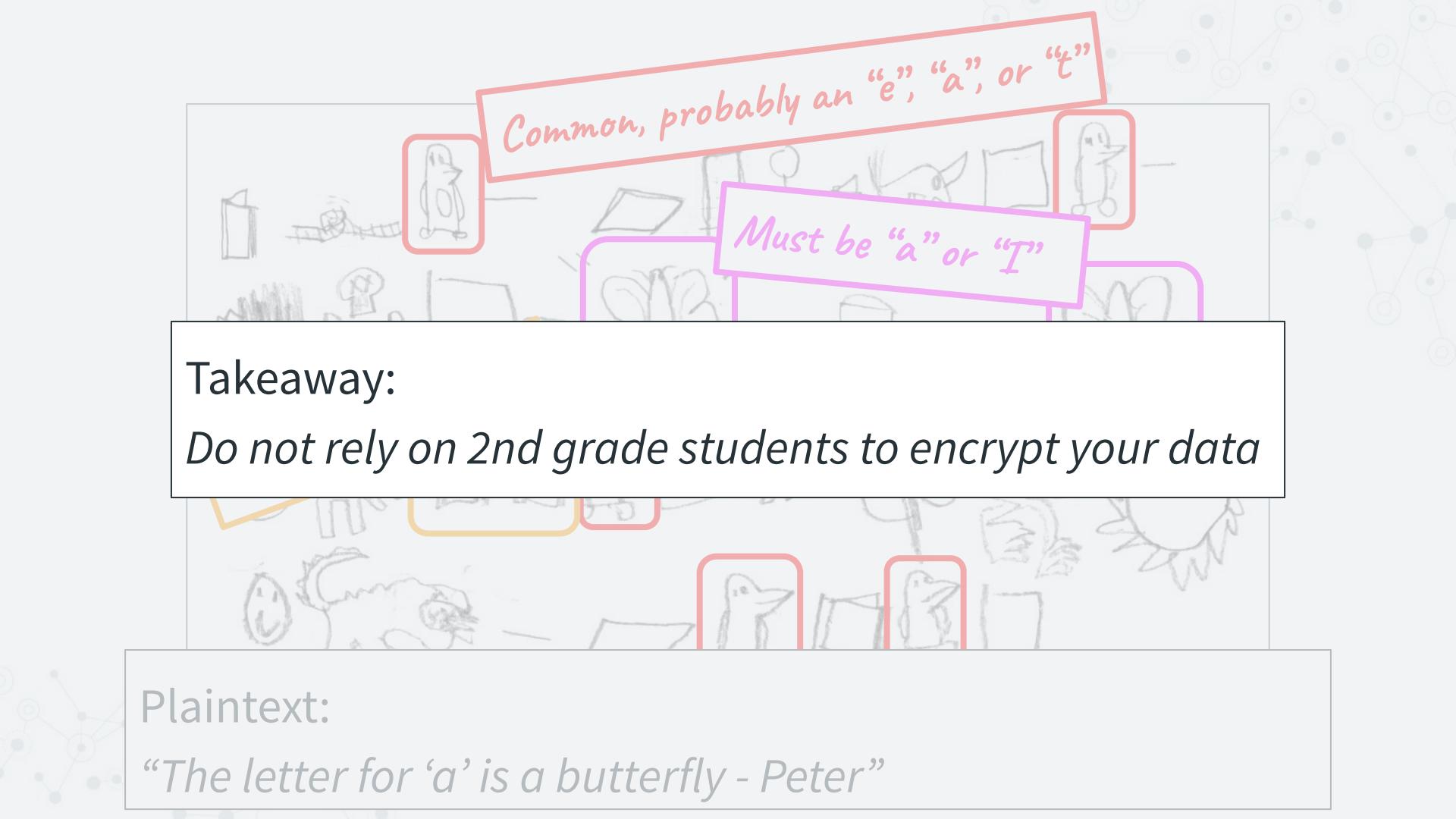
Likely "L", "SS", etc





Plaintext:

*"The letter for 'a' is a butterfly - Peter"*



Common, probably an "e", "a", or "t"

Must be "a" or "I"

Takeaway:

*Do not rely on 2nd grade students to encrypt your data*

Plaintext:

*"The letter for 'a' is a butterfly - Peter"*

# Actual takeaway

Encryption should:

1. Have a large **keyspace** (many potential secret keys)
2. Avoid patterns that can speed up decryption

# Actual takeaway

Encryption should:

1. Have a large **keyspace** (many potential secret keys)
2. Avoid patterns that can speed up decryption

*...and a couple more protections, which will come later*

# Recap

## Vocab

- Man-in-the-Middle
- Plaintext / ciphertext
- Encryption / decryption
- Key
- Keyspace
- Shared secret

## Takeaways

- Encryption provides confidentiality
- Requirements:
  - Large keyspace
  - No patterns

Questions?

# Disclaimer



We are about to talk about a couple modern ciphers

Takeaways, in advance:

- General intuition for how these work
- Terminology and major differences
- Use cases for different types

*You will likely never need to recreate these yourself*



# Advanced Encryption Standard (AES)

- ◎ One of the most common encryption methods
  - Used by this very slideshow!
- ◎ Has not been broken with current tools
- ◎ Operates on a bitstring:

Plaintext: 010100110111101000101101

# Advanced Encryption Standard (AES)

- **Secret key:** A 128, 192, or 256 bit number
- **Key space:**  
115792089237316195423570985008687907853269984  
665640564039457584007913129639936 possibilities!
- Key is treated as a bitstring, like the plaintext:

Key: 01010011011101000101101...

# Advanced Encryption Standard (AES)

- **Secret key:** A 128, 192, or 256 bit number
- **Key space:**  
115792089237316195423570985008687907853269984  
665640564039457584007913129639936 possibilities!
- Key is treated as a bitstring, like the plaintext:

Key: 01010011011101000101101...



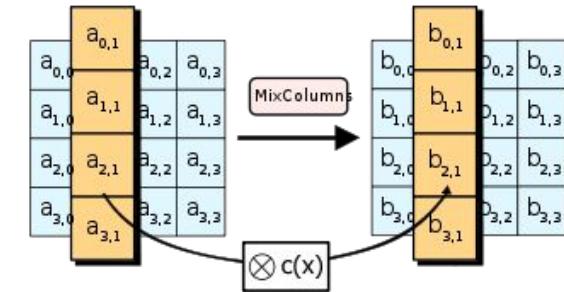
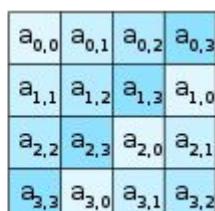
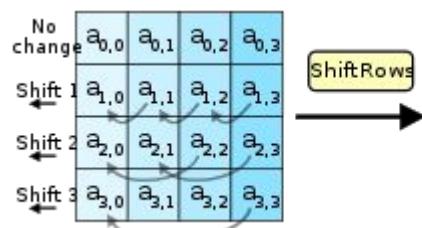
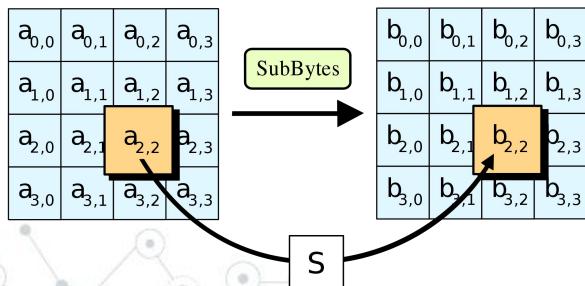
Have a large keyspace

# Advanced Encryption Standard (AES)

## Step 1:

Shift around the bits using a standard set of operations

- Hides patterns, but is known and reversible



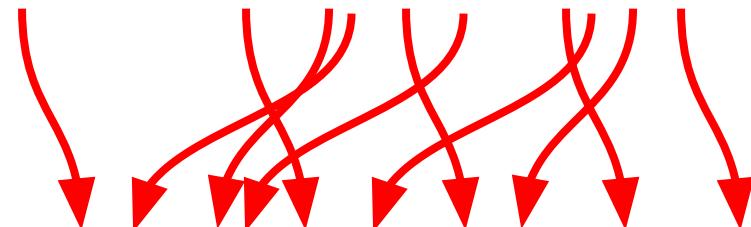
# Advanced Encryption Standard (AES)

## Step 1:

Shift around the bits using a standard set of operations

- Hides patterns, but is known and reversible
- Not-entirely-accurate example:

010100110111101000101101 [plaintext]



0001111111101000001111 [mixed plaintext]

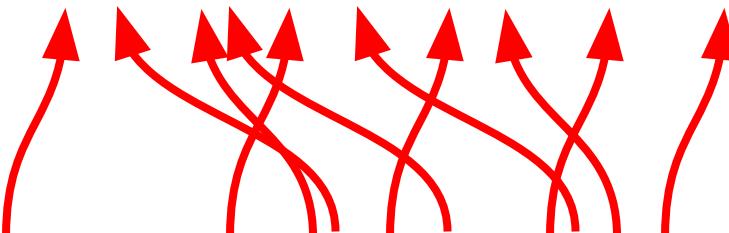
# Advanced Encryption Standard (AES)

## Step 1:

Shift around the bits using a standard set of operations

- Hides patterns, but is known and reversible
- Not-entirely-accurate example:

010100110111101000101101 [plaintext]



00011111111101000001111 [mixed plaintext]

# Advanced Encryption Standard (AES)

## Step 2:

XOR shifted bits with a secret key

- ◎ Cannot be reversed without the key

0001111111101000001111 [*mixed plaintext*]

**xor**

011110011000110001000110 [*secret key*]

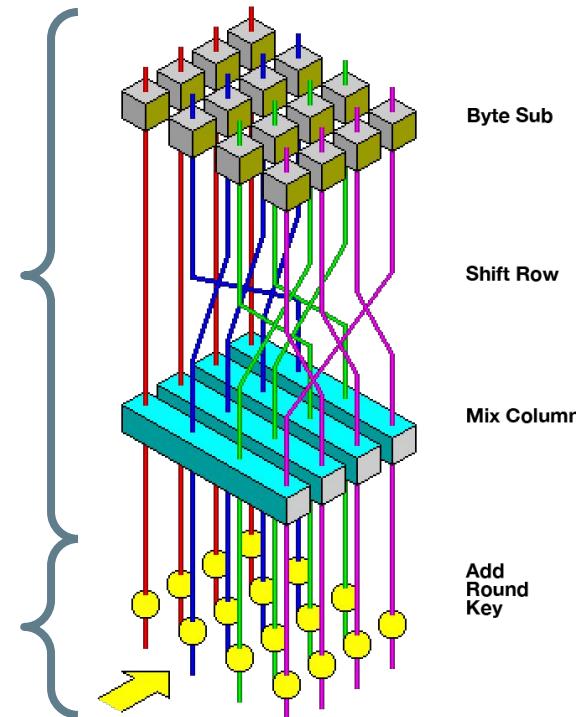
=

011001100111011001001001 [*ciphertext*]

# Advanced Encryption Standard (AES)

**Step 1: Mix (reversible)**

**Step 2: XOR with secret**



# Advanced Encryption Standard (AES)

## Step 3:

- ◎ Repeat 10-14 times, depending on key size

# Advanced Encryption Standard (AES)

## Step 3:

- ◎ Repeat 10-14 times, depending on key size
  - Due to shifting bits, small changes in the input completely change the output

“Test message #1!” => 0x1e30c25bfd494ed54d8e2ad29cd7f723

“Test message #2!” => 0x1acd9034c4b413fc03f23dbded83ddf0

# Advanced Encryption Standard (AES)

## Step 3:

- ◎ Repeat 10-14 times, depending on key size
  - Due to shifting bits, small changes in the input completely change the output

“Test message #1!” => 0x1e30c25bfd494ed54d8e2ad29cd7f723

“Test message #2!” => 0x1acd9034c4b413fc03f23dbded83ddf0



Avoid patterns that can speed up decryption

# Advanced Encryption Standard (AES)

Recap:

1. Mix up plaintext (in a non-secret, easily reversible way)
2. XOR with key (reversible, but only with the same key)
3. Repeat

**Takeaways:**

- XOR is only reversible if both parties have the same key
- Mixing and repetition is needed to avoid patterns

# Advanced Encryption Standard (AES)

## Non-takeaways:

- ◎ Technical specifics
  - Why 128, 192, or 256 bit keys?
  - How and why are the bits mixed up each repetition?
  - Why 10-14 repetitions?

The reasoning behind these boils down to “advanced statistical analysis”, which is a whole class on its own.

**Questions?**



# Potential Problems

*and how to fix them*

# Advanced Encryption Standard (AES)

**Problem:** The secret key is a fixed length. How do we XOR if the plaintext larger than 128, 192, or 256 bits?

Plaintext

1101001010100010101010110110111101001111

Key

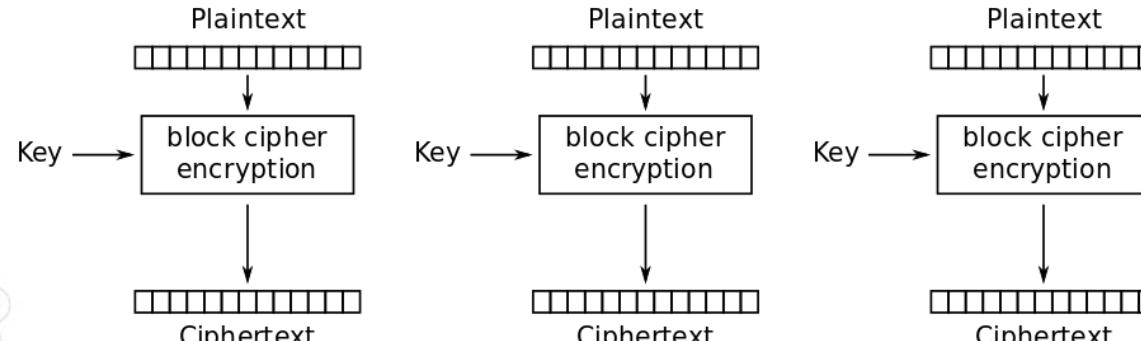
010100110111101000101101

?

# Advanced Encryption Standard (AES)

**Problem:** The secret key is a fixed length. How do we XOR if the plaintext larger than 128, 192, or 256 bits?

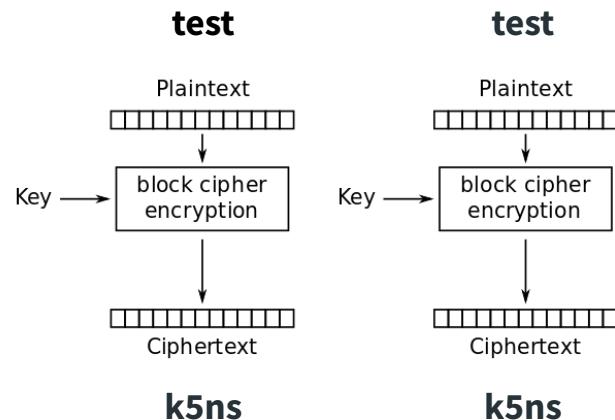
- Solution: Just split it into key-sized blocks and do it a bunch of times?



Electronic Codebook (ECB) mode encryption

# Advanced Encryption Standard (AES)

**Problem #1:** Identical plaintexts map to identical ciphertexts



# Advanced Encryption Standard (AES)

**Problem #1:** Identical plaintexts map to identical ciphertexts

- ◎ Example (uses 4-byte keys for demonstration purposes)

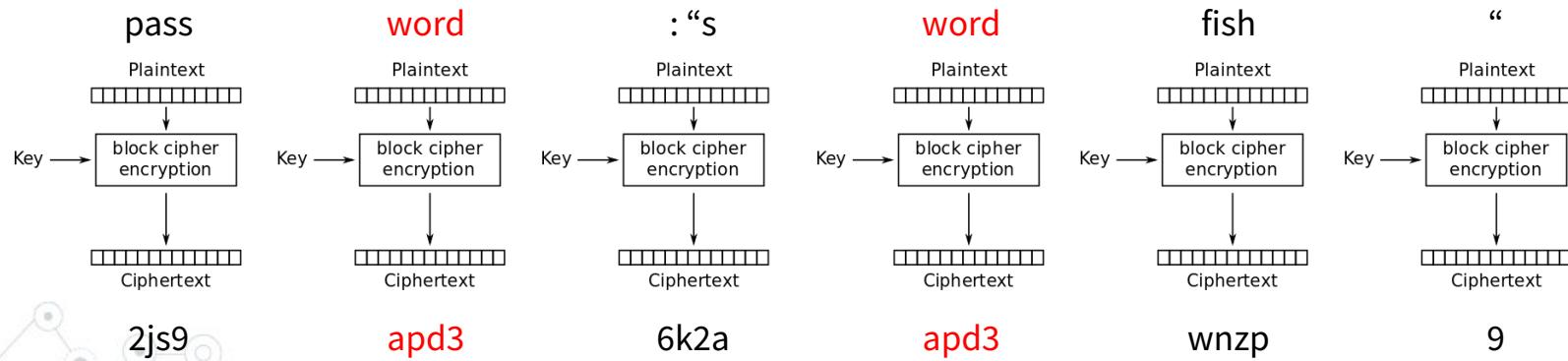
*Plaintext: password: “swordfish”*

# Advanced Encryption Standard (AES)

**Problem #1:** Identical plaintexts map to identical ciphertexts

- Example (uses 4-byte keys for demonstration purposes)

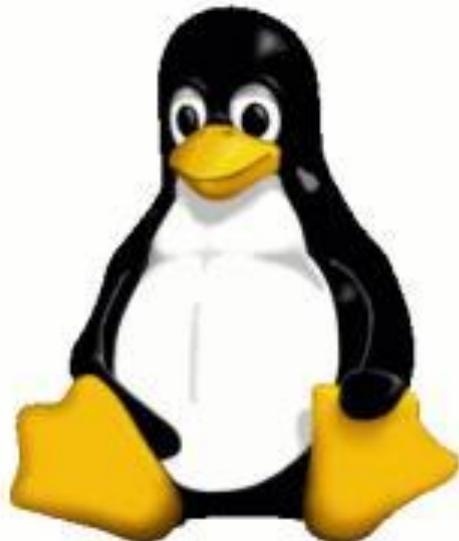
*Plaintext: password: “swordfish”*



*Ciphertext: 2js9apd36k2aapd3wnzp9*

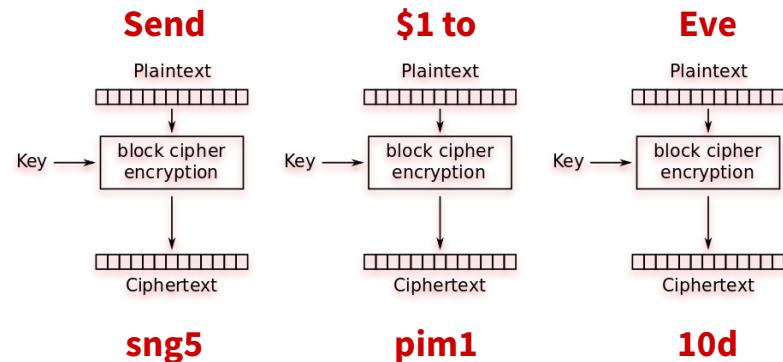
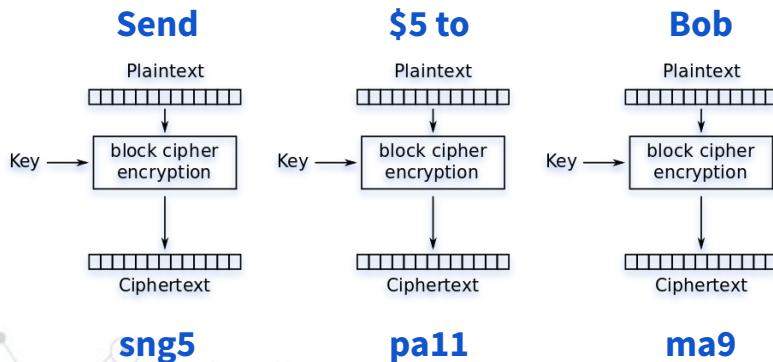
# Advanced Encryption Standard (AES)

**Problem #1:** Identical plaintexts map to identical ciphertexts



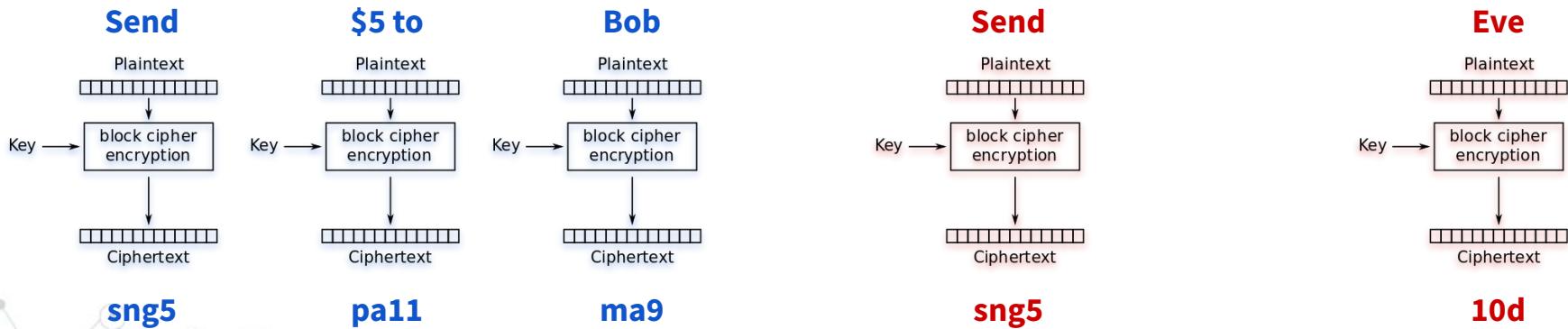
# Advanced Encryption Standard (AES)

**Problem #2:** Attacker can add or remove specific blocks



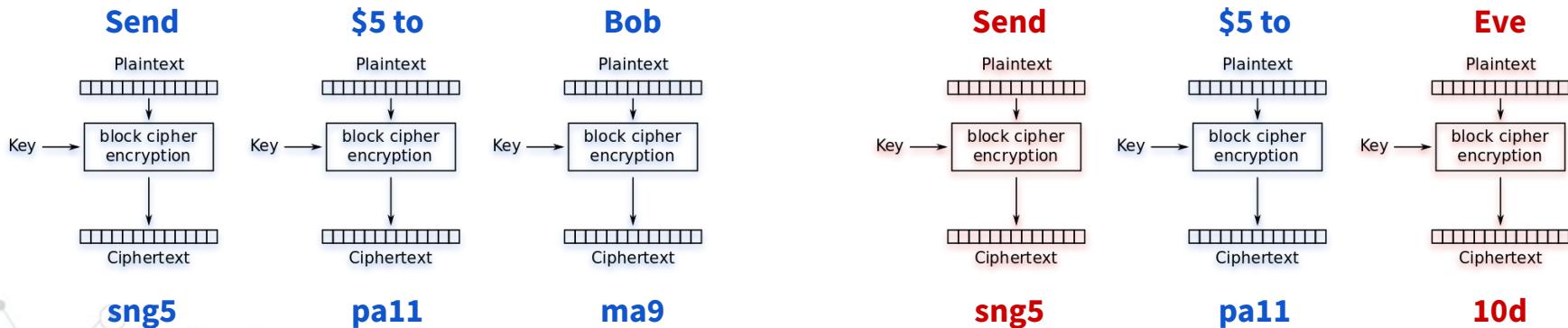
# Advanced Encryption Standard (AES)

**Problem #2:** Attacker can add or remove specific blocks



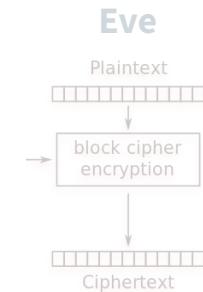
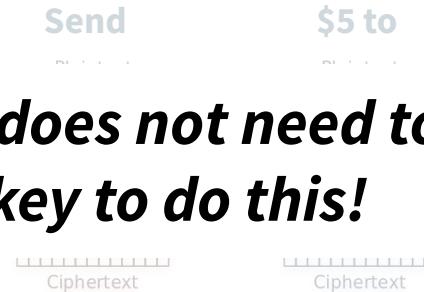
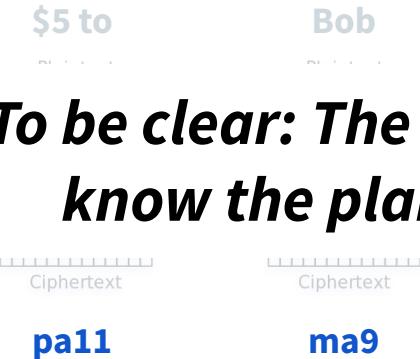
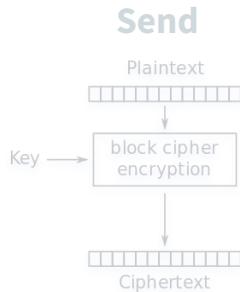
# Advanced Encryption Standard (AES)

**Problem #2:** Attacker can add or remove specific blocks



# Advanced Encryption Standard (AES)

**Problem #2:** Attacker can add or remove specific blocks



***To be clear: The attacker does not need to know the plaintext or key to do this!***

sng5

pa11

ma9

sng5

pa11

10d

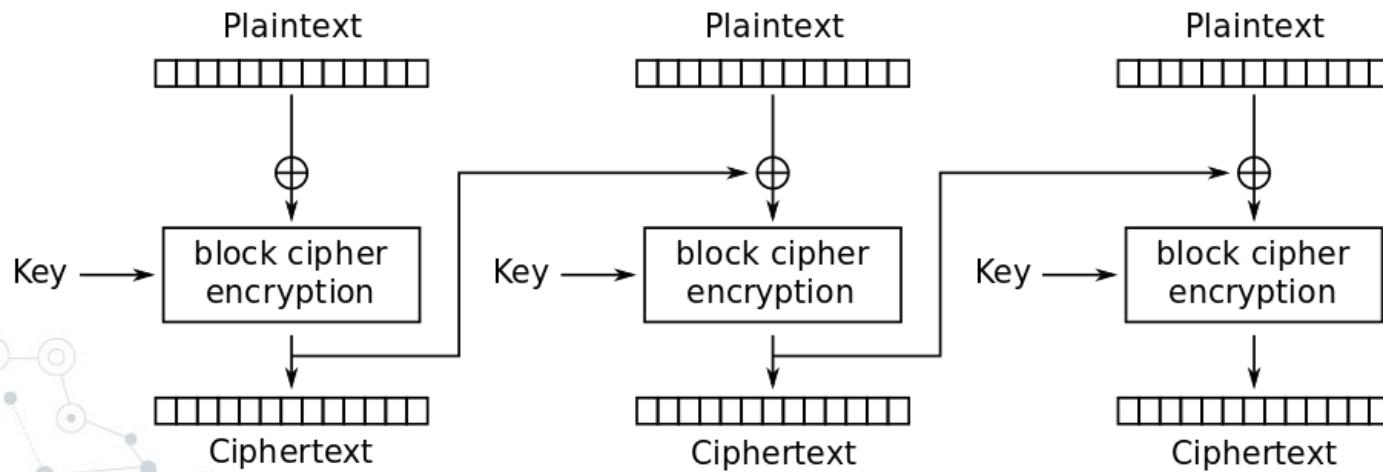
# Advanced Encryption Standard (AES)

**Takeaway:** Each individual ciphertext block is secure, but multiple blocks with the same key contain exploitable patterns



# Advanced Encryption Standard (AES)

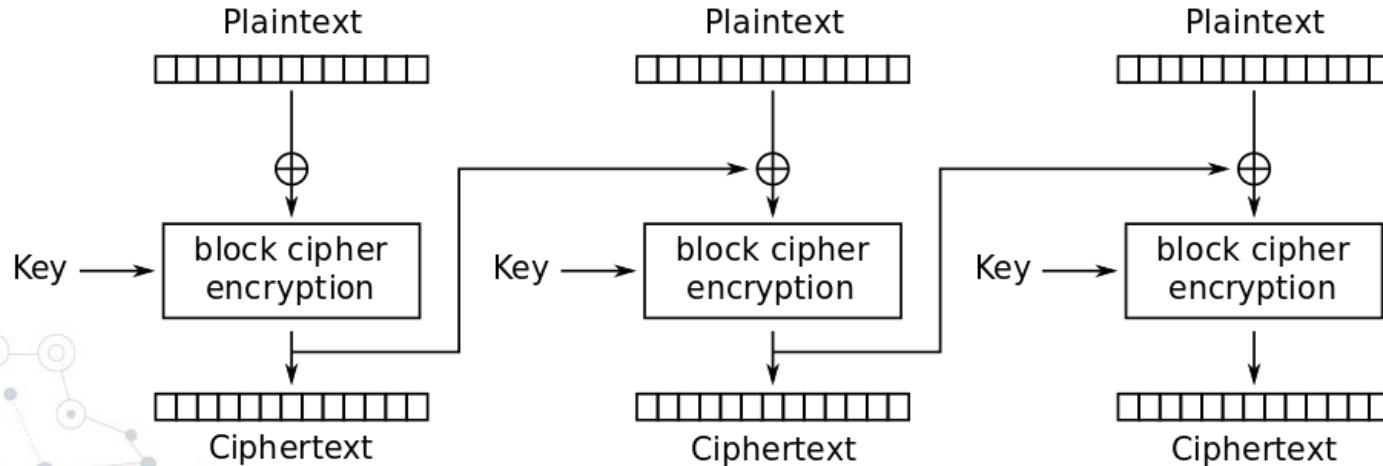
**Solution:** Use the result of each block when encrypting the next



# Advanced Encryption Standard (AES)

**Solution:** Use the result of each block when encrypting the next

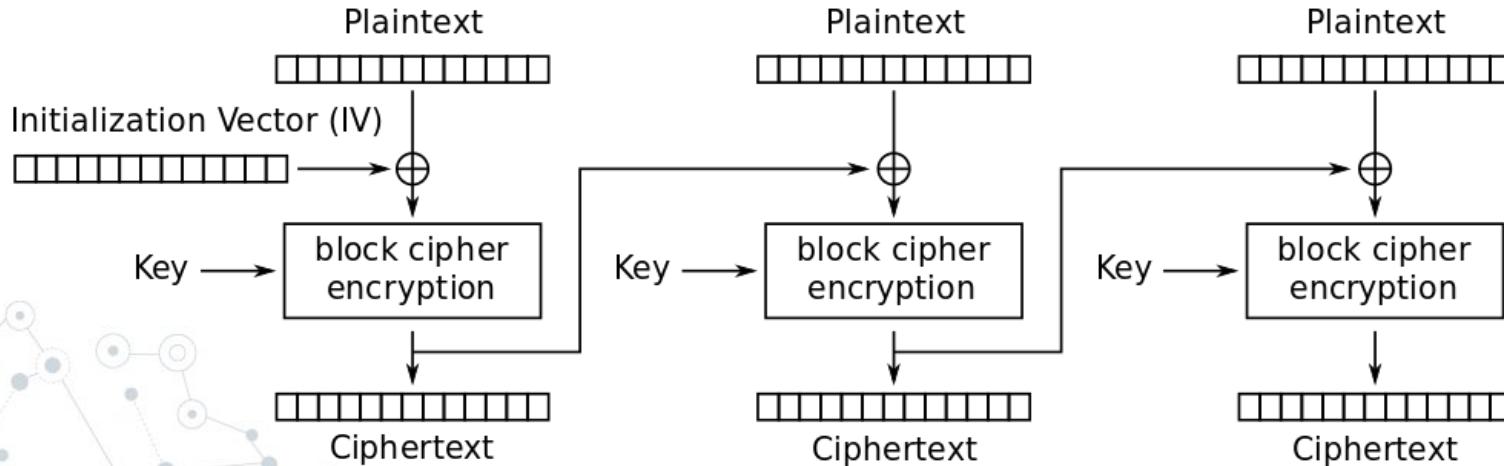
- Still reversible, but ciphertext now depends on previous block



# Advanced Encryption Standard (AES)

**Solution:** Use the result of each block when encrypting the next

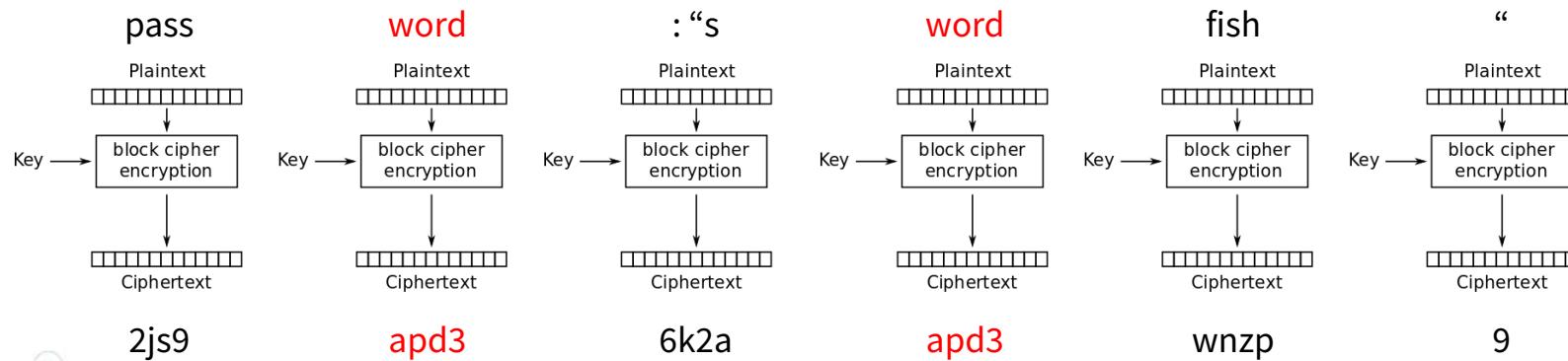
- Still reversible, but ciphertext now depends on previous block
- Initialization Vector (IV):** Non-secret number for first block



# Advanced Encryption Standard (AES)

No block chaining

Plaintext: password: “swordfish”

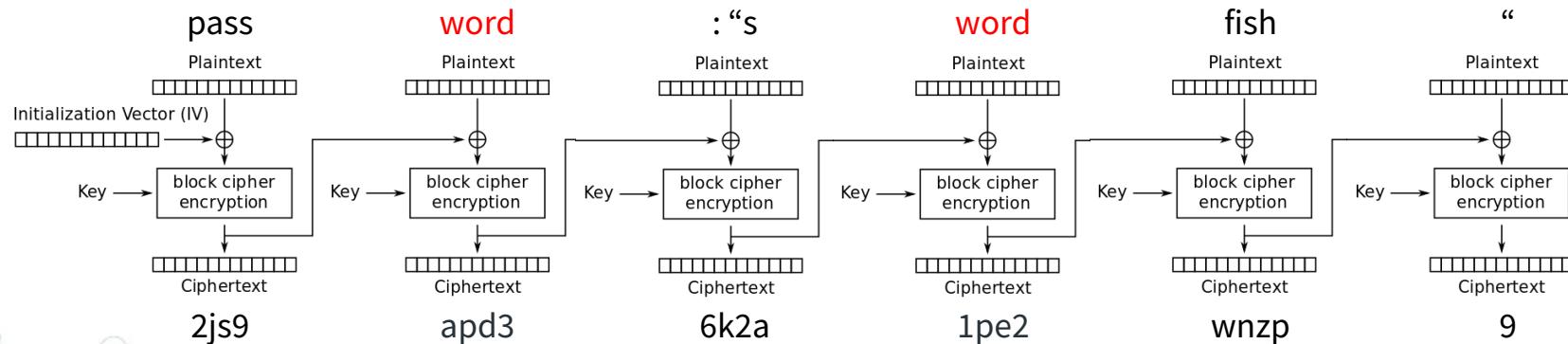


Ciphertext: 2js9apd36k2aapd3wnzp9

# Advanced Encryption Standard (AES)

With block chaining

Plaintext: password: “swordfish”

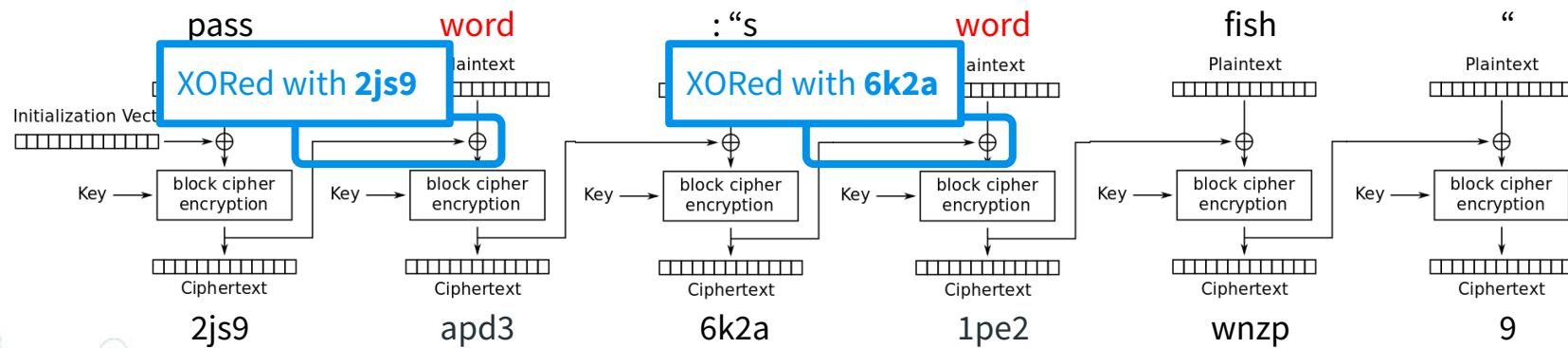


Ciphertext: 2js9apd36k2a1pe2wnzp9

# Advanced Encryption Standard (AES)

With block chaining

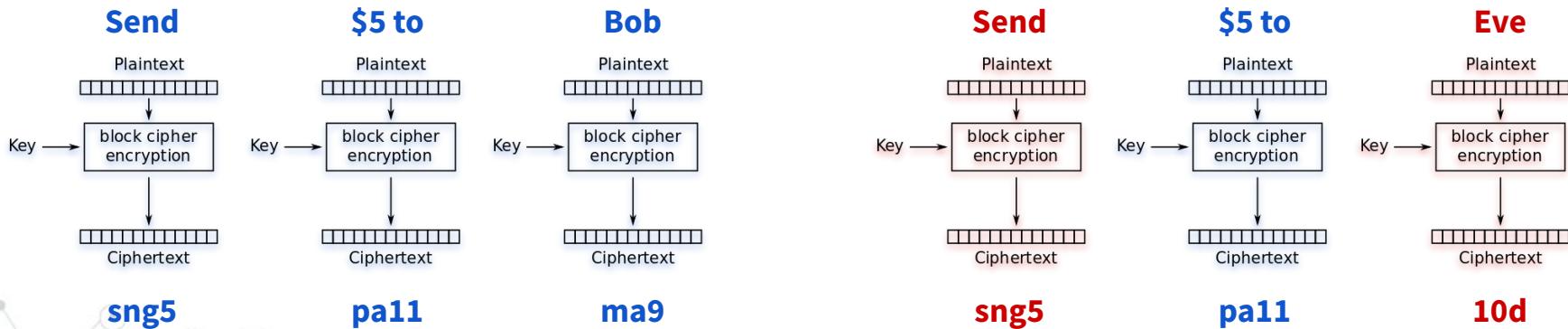
Plaintext: password: “swordfish”



Ciphertext: 2js9apd36k2a1pe2wnzp9

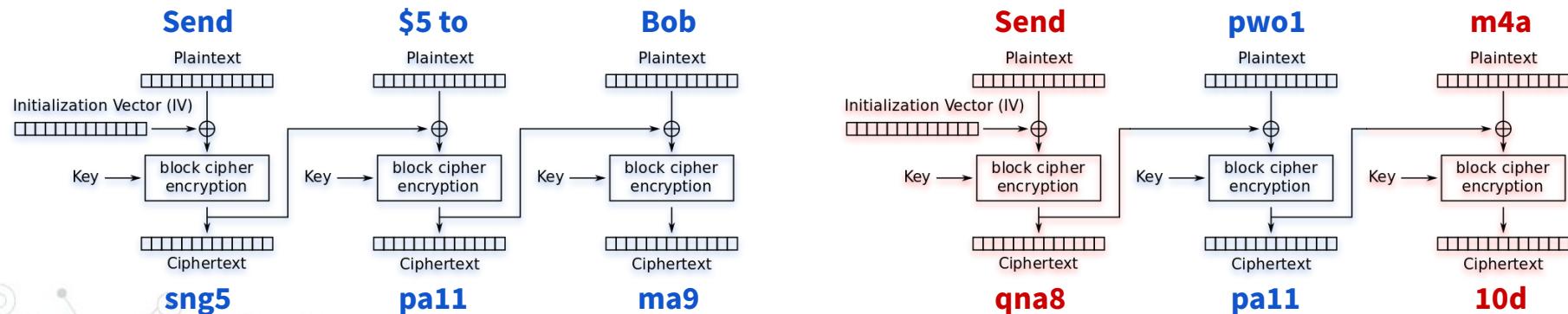
# Advanced Encryption Standard (AES)

No block chaining



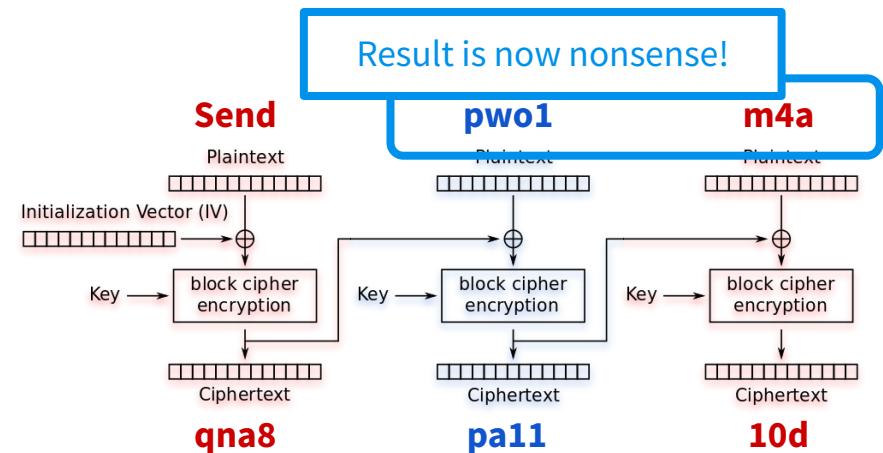
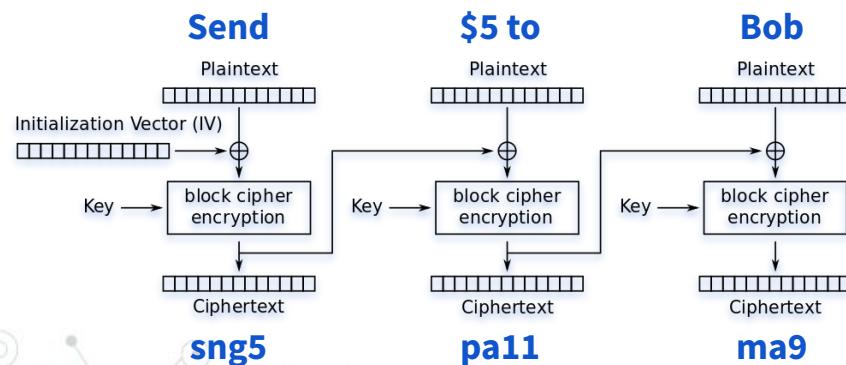
# Advanced Encryption Standard (AES)

With block chaining



# Advanced Encryption Standard (AES)

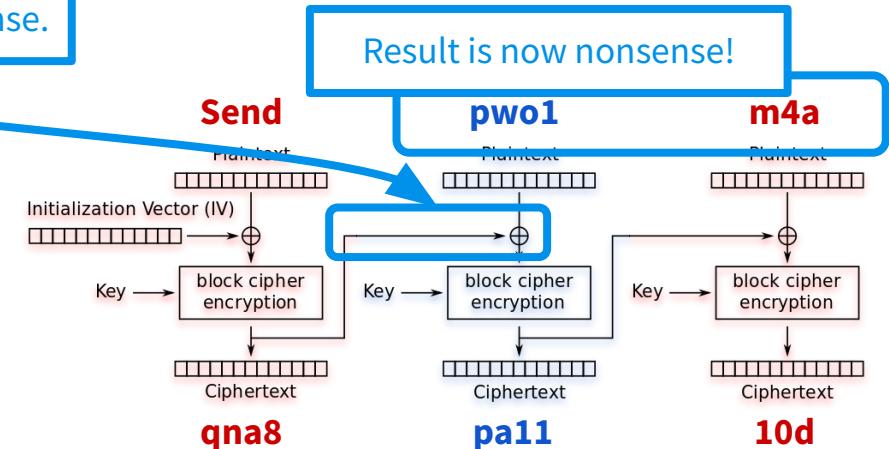
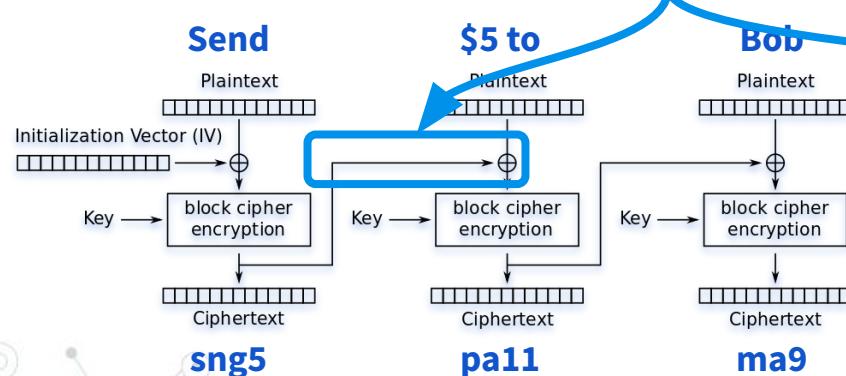
With block chaining



# Advanced Encryption Standard (AES)

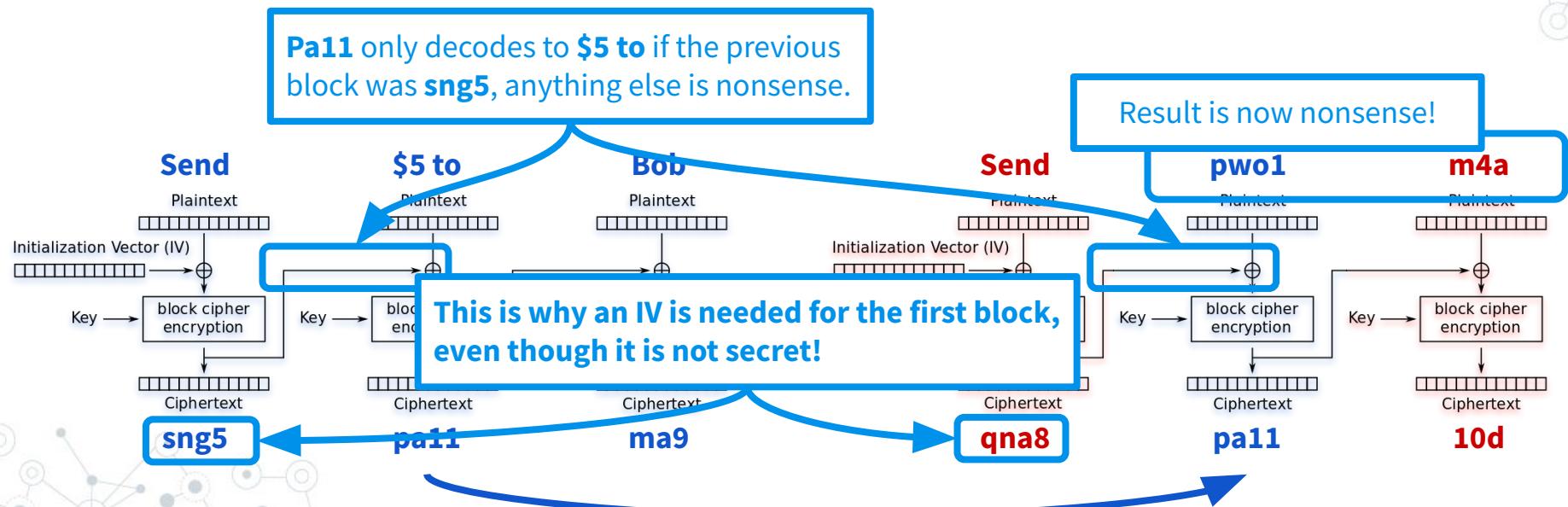
## With block chaining

**Pa11** only decodes to **\$5 to** if the previous block was **sng5**, anything else is nonsense.



# Advanced Encryption Standard (AES)

## With block chaining





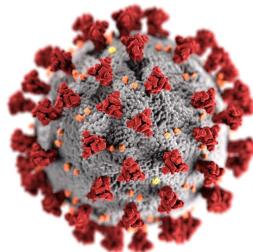
# Data Security: Day 2

# Logistics!

## News!

So, I either have Covid or likely will soon

- Lectures will stay remote for 2 weeks
- Recitations will resume their original mode next week



# Logistics!

- Office hours are now on Canvas
- First homework will be assigned a week from today\*

\* Assuming no interference from the spiky plague

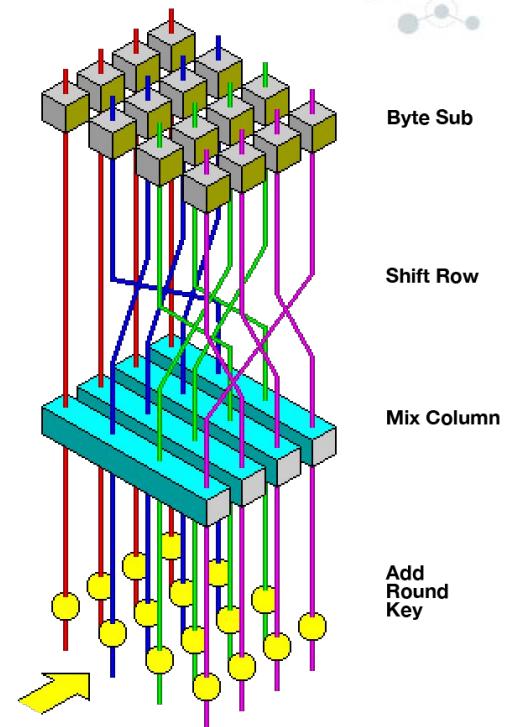
# Question from last lecture

- ➊ Export-grade encryption:
  - Still illegal in some cases, apparently!
  - Fun Wikipedia rabbit hole:  
*[https://en.wikipedia.org/wiki/Export\\_of\\_cryptography  
\\_from\\_the\\_United\\_States](https://en.wikipedia.org/wiki/Export_of_cryptography_from_the_United_States)*

# Recap

## AES encryption:

1. Mix up plaintext
  - (non-secret, reversible way)
2. XOR with key
  - (reversible, but only with the same key)
3. Repeat



# Recap

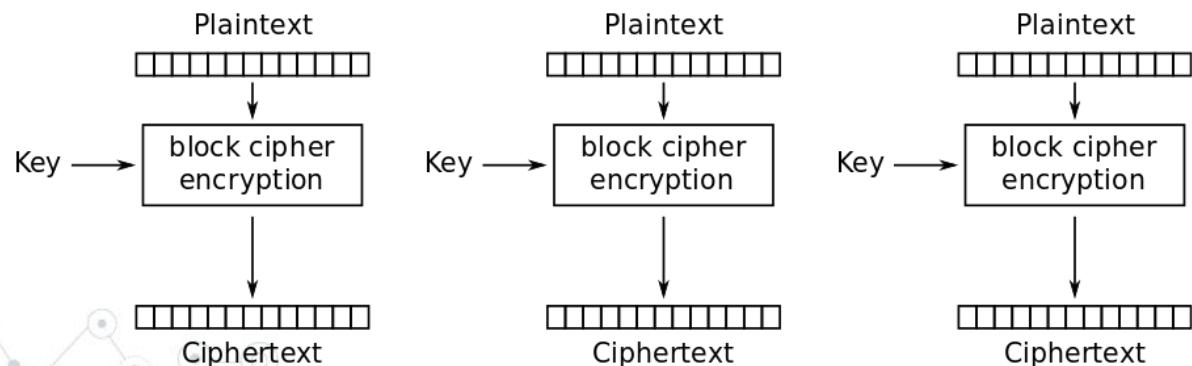
Note: XOR is reversible by doing it again

11110000 [mixed plaintext]  
**xor**  
10100011 [secret key]  
=  
**01010011** [ciphertext]

**01010011** [ciphertext]  
**xor**  
10100011 [secret key]  
=  
11110000 [mixed plaintext]

# Recap

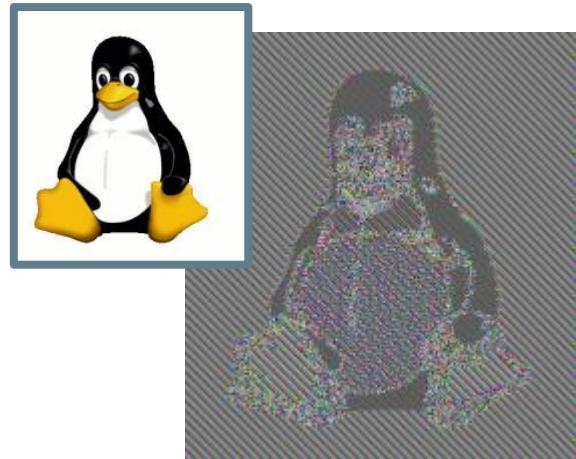
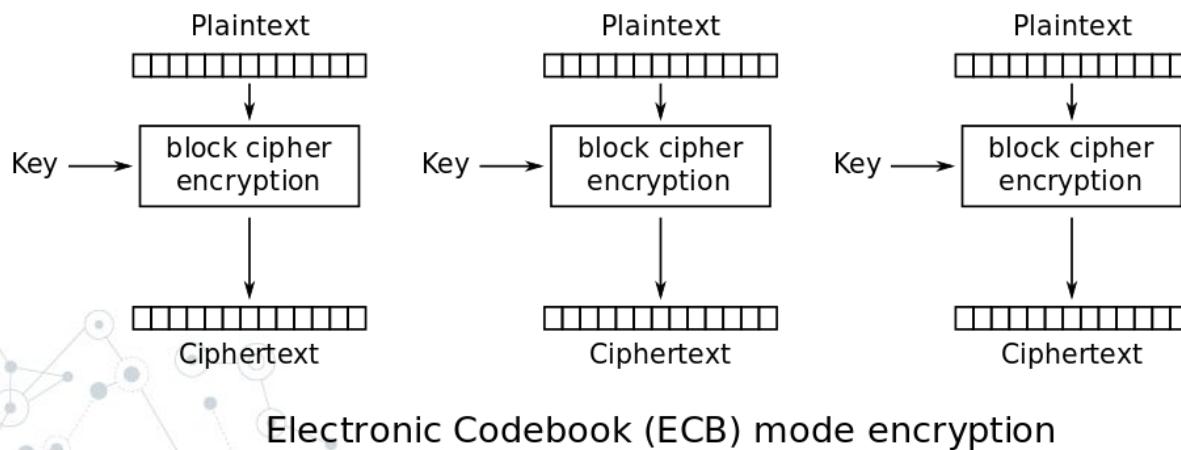
- Limited by key size, need to use multiple blocks for large inputs



Electronic Codebook (ECB) mode encryption

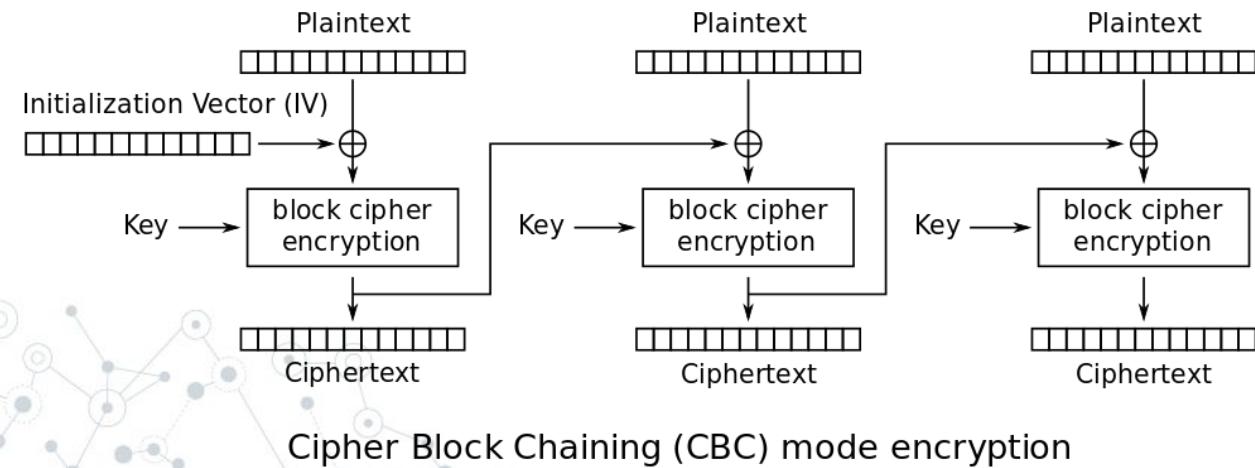
# Recap

- Limited by key size, need to use multiple blocks for large inputs
- Naive approach leaves patterns between blocks



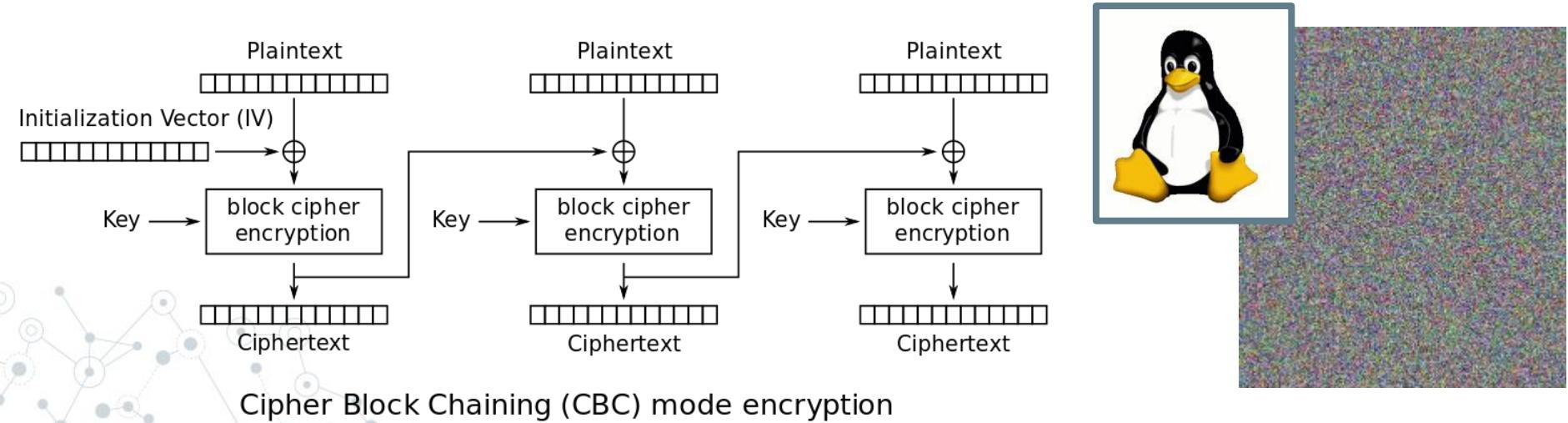
# Recap

- Blocks can be “chained” by doing another XOR from each encrypted block to the plaintext of the next
- The very first block uses a (non-secret) initialization vector



# Recap

- Blocks can be “chained” by doing another XOR from each encrypted block to the plaintext of the next
- The very first block uses a (non-secret) initialization vector



# Advanced Encryption Standard (AES)

Benefits of chaining blocks:

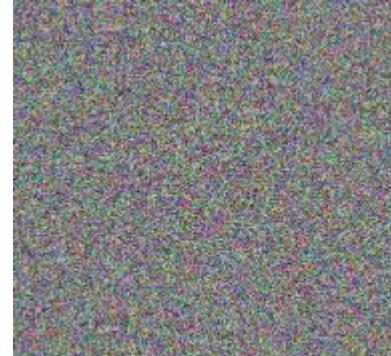
- ◎ No patterns between blocks
- ◎ Cannot swap blocks between ciphertexts or concatenate data without corrupting later block



Plaintext



No block chaining

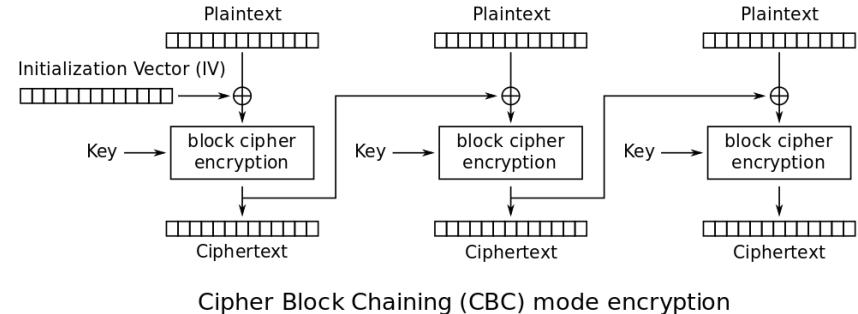
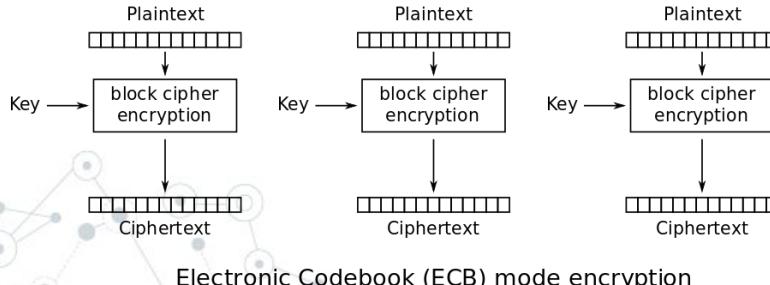


Block chaining

# Advanced Encryption Standard (AES)

**Mode of operation:** Different ways of applying block ciphers

- Same underlying cipher code for each block
- More secure modes reduce patterns between blocks
- Other modes exist, but we will omit them for time reasons



# Advanced Encryption Standard (AES)

Bad practices from your recitation:

```
aes_cipher = AES.new(secret, AES.MODE_ECB)
```

(Using the block chaining mode would have been more secure, but require the IV)

```
aes_cipher = AES.new(  
    secret, AES.MODE_CBC, b'ASDFASDFASDFASDF')
```

# Advanced Encryption Standard (AES)

Takeaways (from this and lecture):

- General idea of encryption uses and challenges
- Know what all of these mean:

```
AES.new(secret, AES.MODE_CBC, b'ASDFASDFASDFASDF')
```

^^^^^

^^^^^

^^^^^

Key

Mode

Initialization Vector

# Quick note #1: Ransomware

Encryption is not always a good thing!

- ➊ **Ransomware:** The hot new trend in viruses



[https://en.wikipedia.org/wiki/WannaCry\\_ransomware\\_attack](https://en.wikipedia.org/wiki/WannaCry_ransomware_attack)

# Quick note #1: Ransomware

Encryption is not always a good thing!

- **Ransomware:** The hot new trend in viruses
- Encrypts files and demands a ransom to reveal the secret key

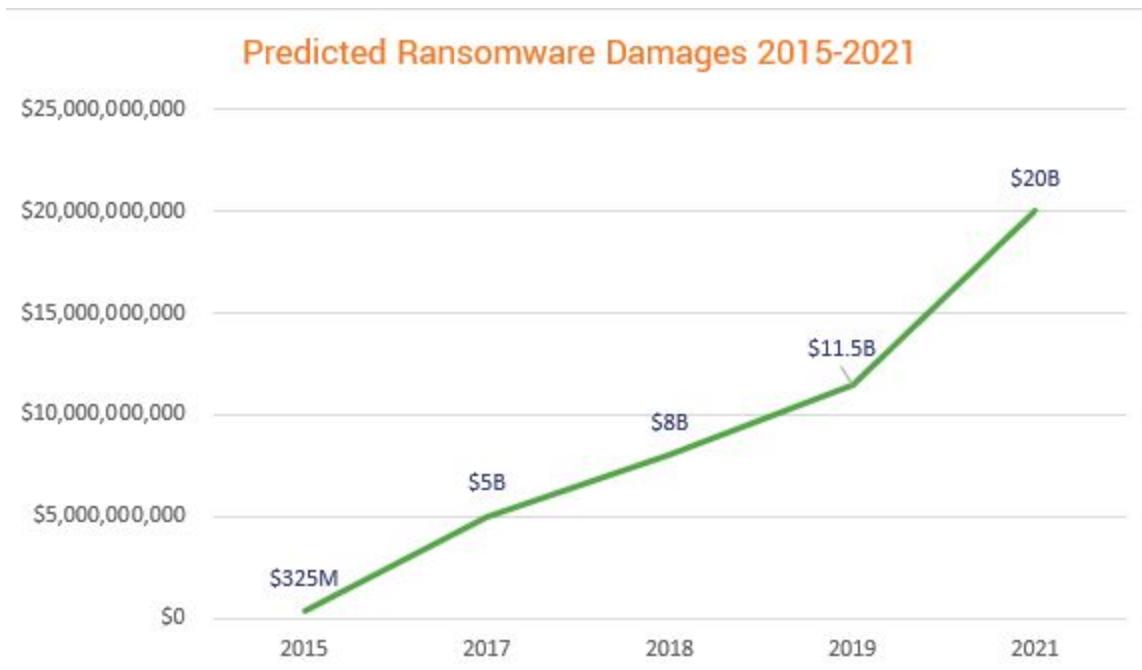


[https://en.wikipedia.org/wiki/WannaCry\\_ransomware\\_attack](https://en.wikipedia.org/wiki/WannaCry_ransomware_attack)

# Quick note #1: Ransomware



# Quick note #1: Ransomware



<https://securityboulevard.com/2020/02/20-ransomware-statistics-youre-powerless-to-resist-reading/>

# Quick note #1: Ransomware

## White House confirms person behind Colonial Pipeline ransomware attack nabbed during Russian REvil raid

Russian officials arrested 14 alleged members of the REvil ransomware group on Friday.



Written by **Jonathan Greig**, Staff Writer  
on January 14, 2022 | Topic: Ransomware

White House officials told reporters on Friday that the person behind the ransomware attack on [Colonial Pipeline](#) last year was arrested as part of the [larger raid against the REvil ransomware group](#) by Russian law enforcement on Friday, confirming [reporting from The Washington Post](#).

On Friday afternoon, Washington Post reporter Ellen Nakashima



**Ransomware: An executive guide to one of the biggest menaces on the web**

Everything you need to

### RELATED



[Moscow court charges 8 alleged REvil ransomware hackers](#)

[Russian authorities take down REvil ransomware gang](#)

[Ransomware warning: Cyber criminals are mailing out USB drives that install malware](#)

# Quick note #2: Randomness



We rely on randomness to pick a good secret

- ➊ Randomness is very hard!
- ➋ **Pseudorandomness:** Picking numbers that appear random, but are deterministic



# Quick note #2: Randomness

**Pseudorandomness** is often a deterministic operation starting with an original “seed” value

- ◎ Same seed, same “randomness”

```
>>> import random  
>>> random.seed(0)  
>>> random.choices([0, 1], k=20)  
[1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1]  
>>> random.seed(0)  
>>> random.choices([0, 1], k=20)  
[1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1]
```

## Quick note #2: Randomness



# Quick note #2: Randomness

Sometimes based on guessible factors!

```
now = _PyTime_GetSystemClock();
key[0] = (uint32_t)(now & 0xffffffffU);
key[1] = (uint32_t)(now >> 32);

// [...]
key[2] = (uint32_t)getpid();

// [...]
now = _PyTime_GetMonotonicClock();
key[3] = (uint32_t)(now & 0xffffffffU);
key[4] = (uint32_t)(now >> 32);
```

# Quick note #2: Randomness

Sometimes based on guessable factors!

```
now = _PyTime_GetSystemClock();  
key[0] = (uint32_t)(now & 0xffffffffU); → Current time  
key[1] = (uint32_t)(now >> 32);
```

```
// [...]  
key[2] = (uint32_t)getpid(); → Process ID
```

```
// [...]  
now = _PyTime_GetMonotonicClock(); → System uptime  
key[3] = (uint32_t)(now & 0xffffffffU);  
key[4] = (uint32_t)(now >> 32);
```

# Quick note #2: Randomness

Sometimes based on guessable factors!

```
now = _PyTime_GetSystemClock(); → Current time  
key[0] = (uint32_t)(now & 0xffffffffU);
```

**Warning:** The pseudo-random generators of this module should not be used for security purposes. For security or cryptographic uses, see the `secrets` module.

```
key[2] = (uint32_t)get → Process ID
```

```
// [...]
```

```
now = _PyTime_GetMonotonicClock(); → System uptime  
key[3] = (uint32_t)(now & 0xffffffffU);  
key[4] = (uint32_t)(now >> 32);
```

# Quick note #2: Randomness

CRIME & COURTS

## Imprisoned lottery scammer Eddie Tipton seeks to overturn sentence

Associated Press

Published 2:20 p.m. CT Nov. 11, 2021

[View Comments](#)



Get an overview of how Eddie Tipton created one of the largest lottery scams in history. *Wochit*

# Quick note #2: Randomness

Bad sources of randomness:

- People
- Guessable factors like time

Good sources of randomness:

- Actual random events (e.g. random background OS stuff)
  - Powers /dev/random and /dev/urandom!

# Quick note #2: Randomness

Bad sources of randomness:

- People
- Guessable factors like time

Good sources of randomness:

- Actual random events (e.g. random background OS stuff)
  - Powers /dev/random and /dev/urandom!
- Lava lamps?

# Quick note #2: Randomness



<https://www.cloudflare.com/learning/ssl/lava-lamp-encryption/>

# Takeaways

- ➊ **Ransomware:**  
Using encryption for evil, to attack availability
- ➋ **Pseudorandomness:**  
Deterministic “randomness” based off a seed value
- ➌ **Cryptographically secure randomness:**  
Randomness that is suitably “random” enough to use for secure encryption

# Key sharing problem

AES is a **symmetric-key** algorithm

- ➊ Relies on both parties having a shared secret key

Chicken-and-egg problem: Shared secret key is needed for encrypted channel, but an encrypted channel is needed to safely share key

# Key sharing problem



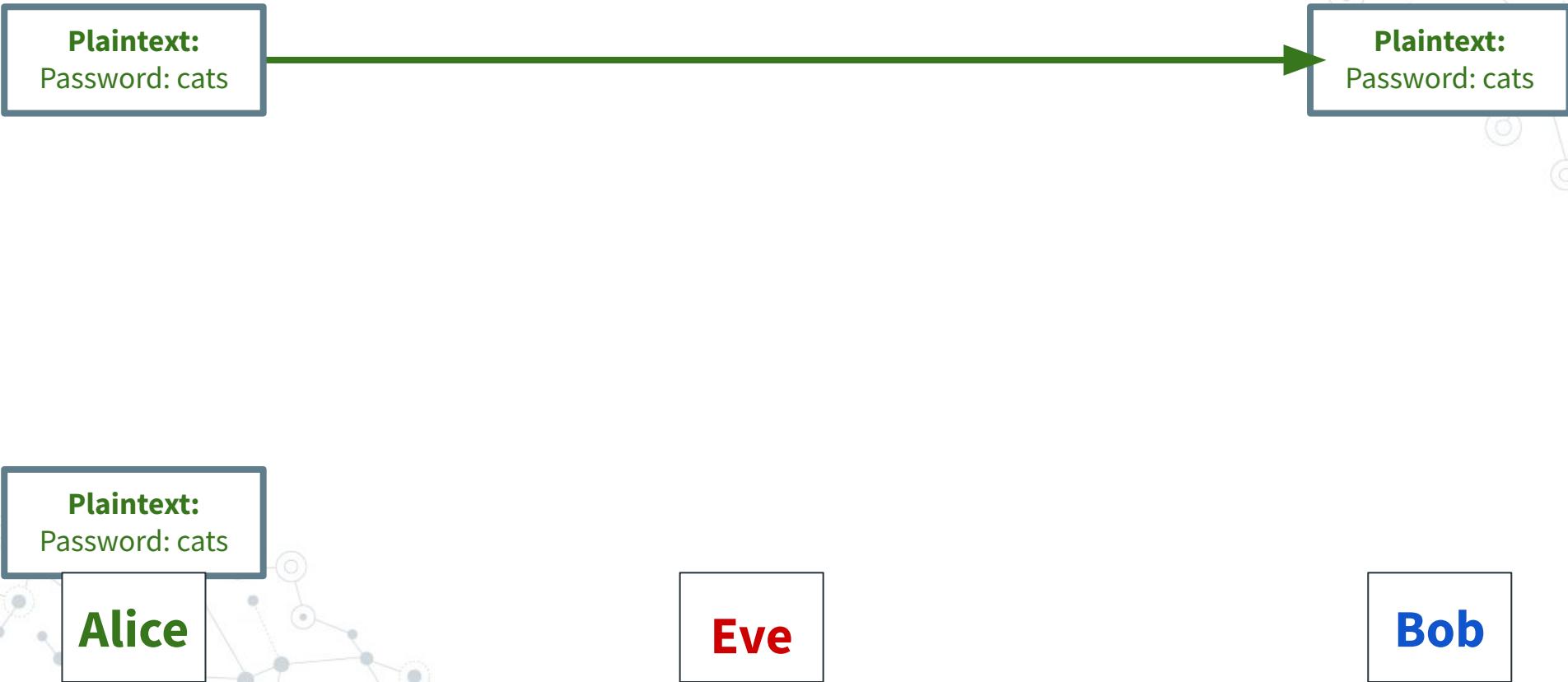
**Plaintext:**  
Password: cats

**Alice**

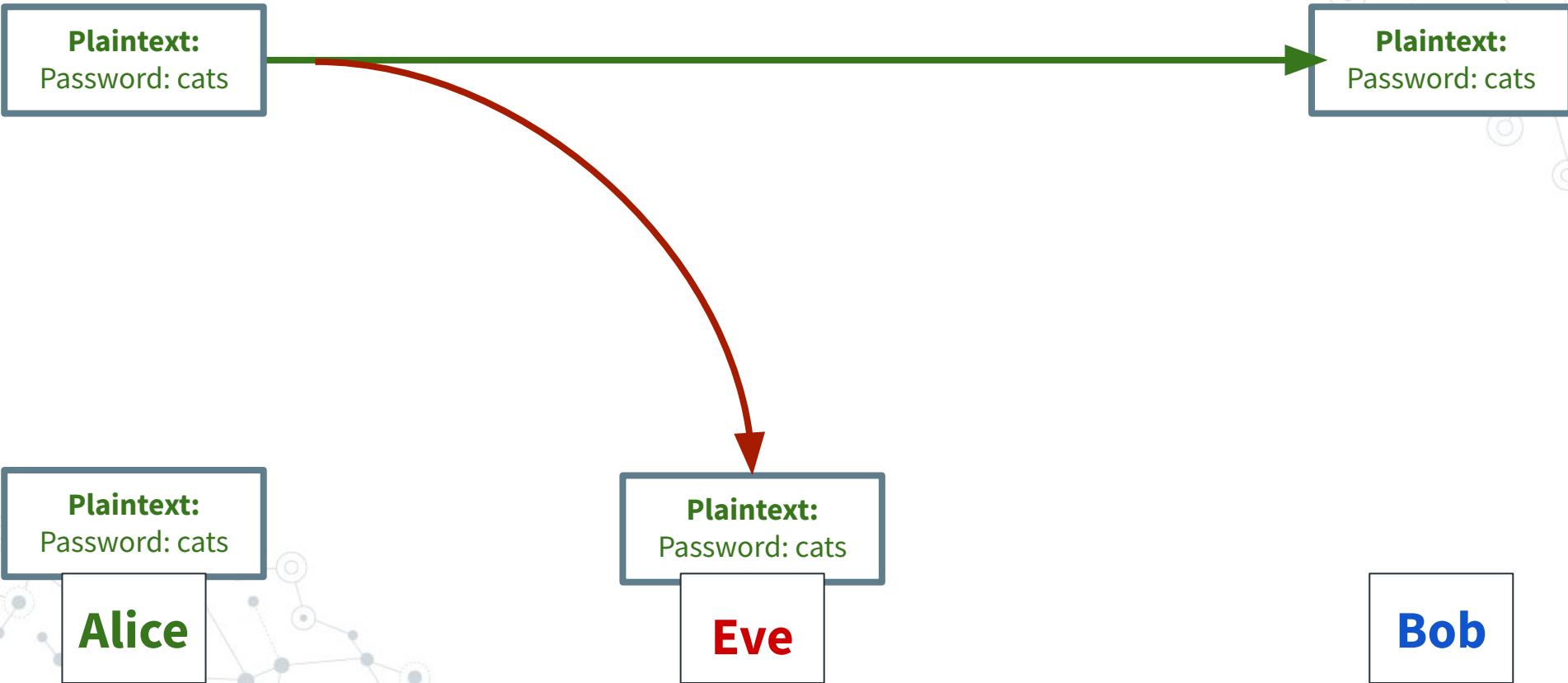
**Eve**

**Bob**

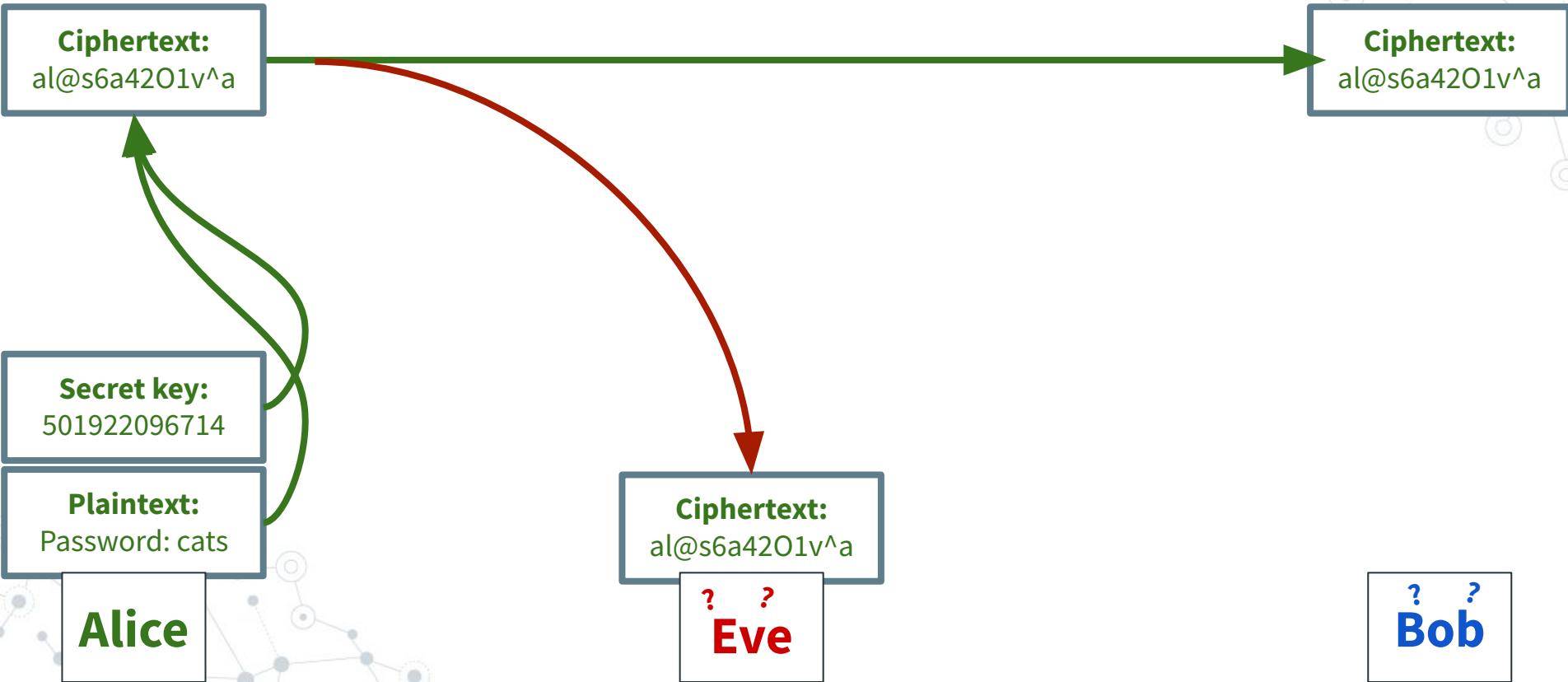
# Key sharing problem



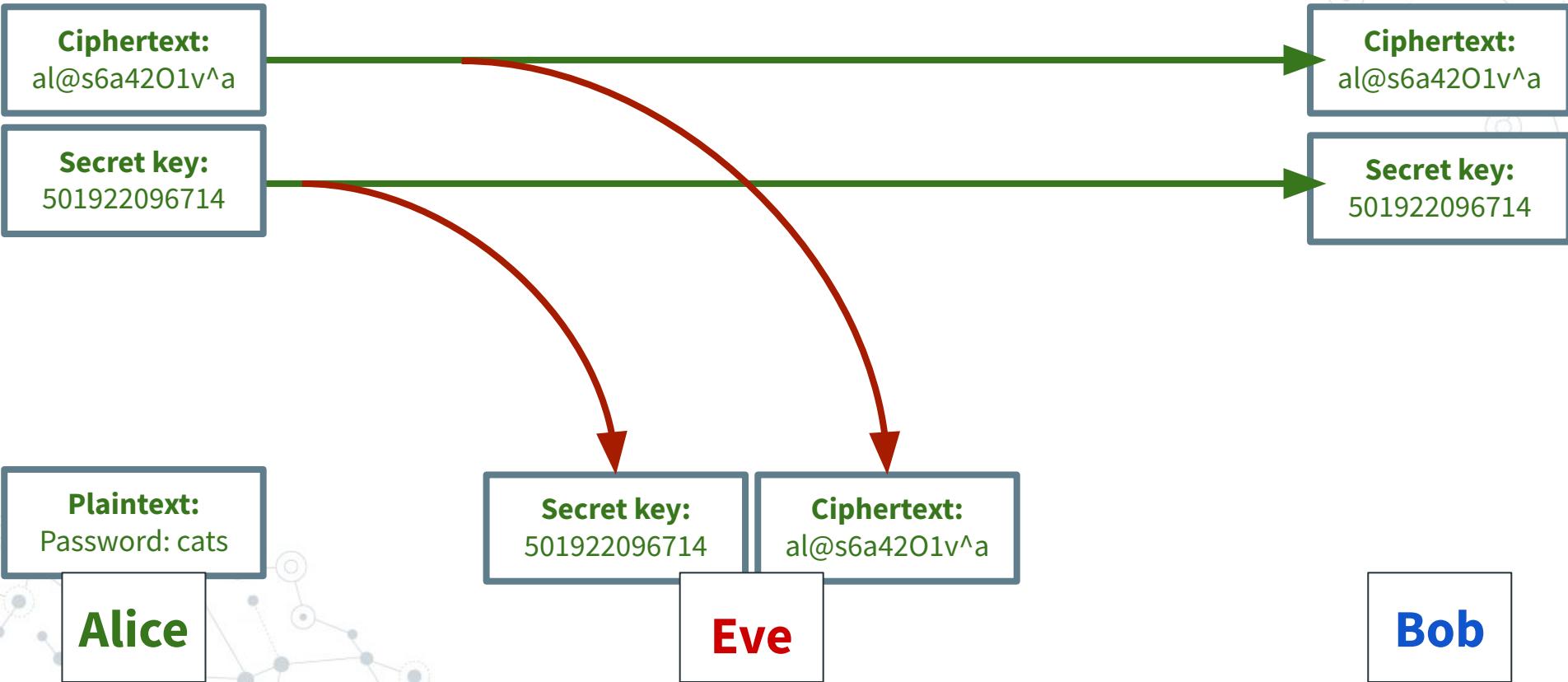
# Key sharing problem



# Key sharing problem



# Key sharing problem

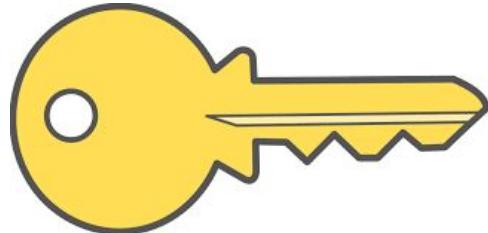


# Asymmetric encryption

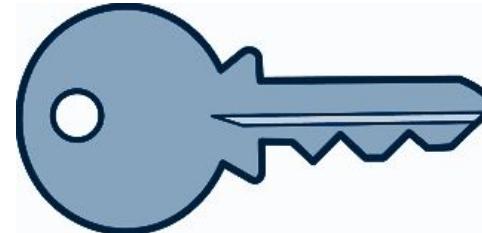
Asymmetric encryption

- ◎ Uses different keys to encrypt and decrypt

Encryption key



Decryption key



# Asymmetric encryption

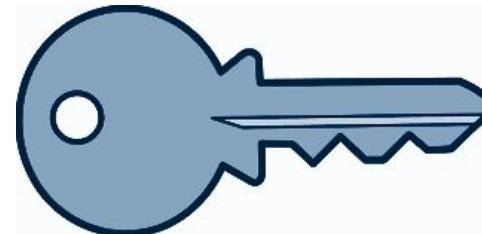
Asymmetric encryption

- Uses different keys to encrypt and decrypt
- Acts more like a “lock” and a “key”

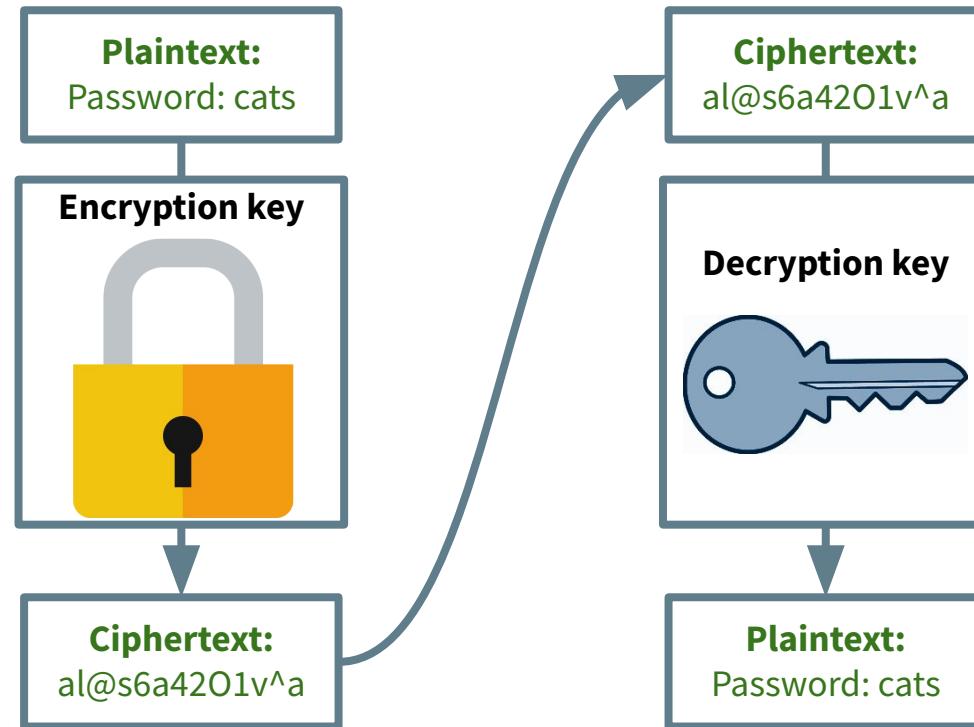
Encryption key



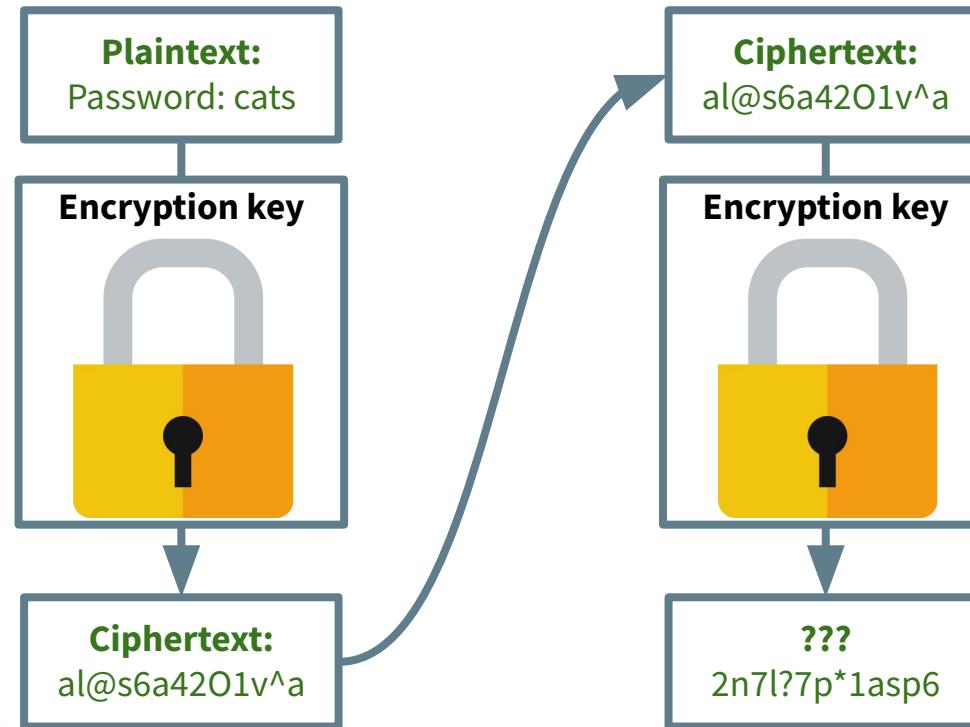
Decryption key



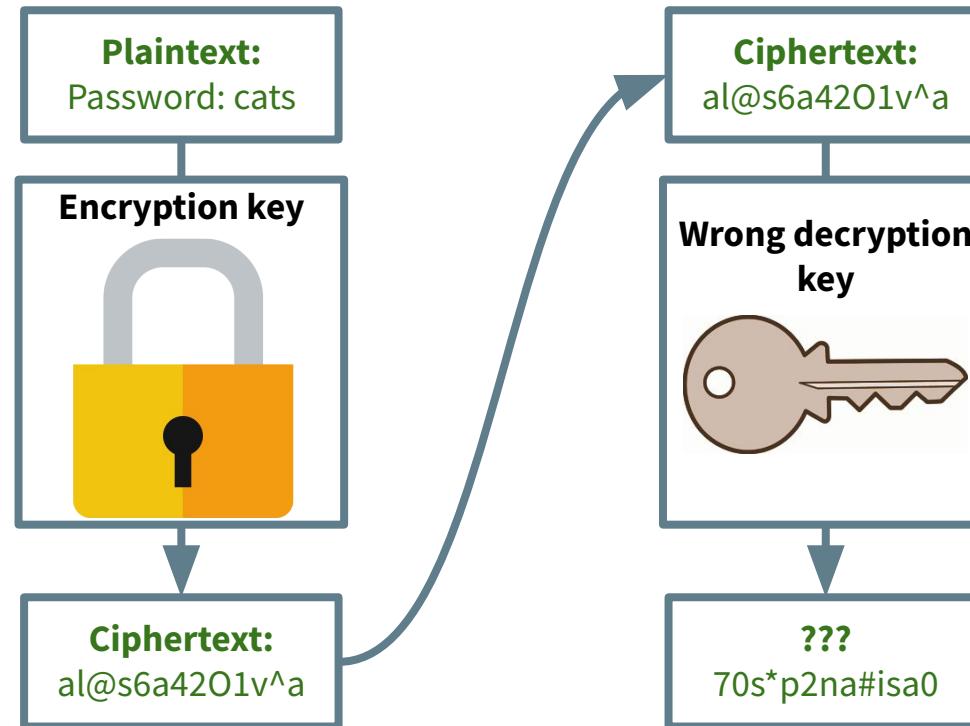
# Asymmetric encryption



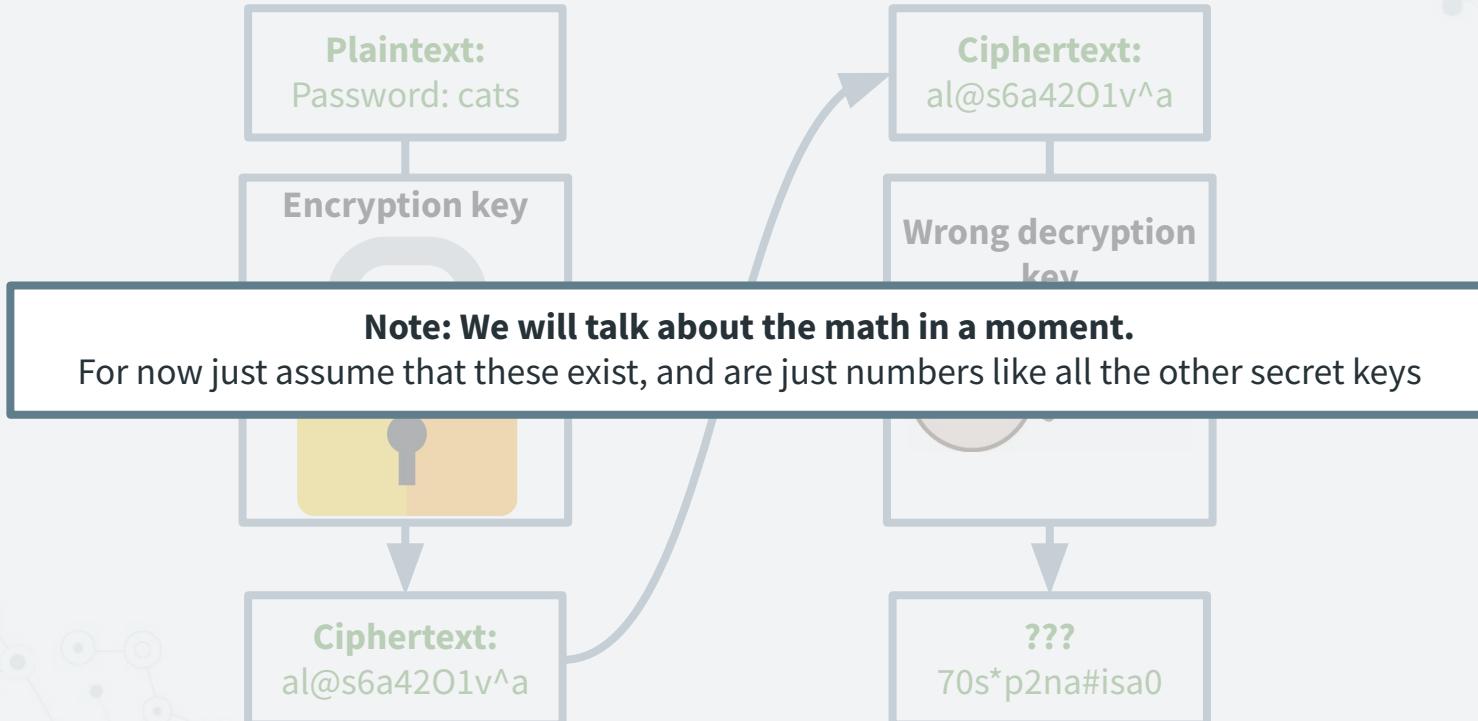
# Asymmetric encryption



# Asymmetric encryption



# Asymmetric encryption



# Asymmetric Encryption



**Plaintext:**  
Password: cats

**Alice**

**Eve**

**Bob**

# Asymmetric Encryption

Decryption key



Encryption key



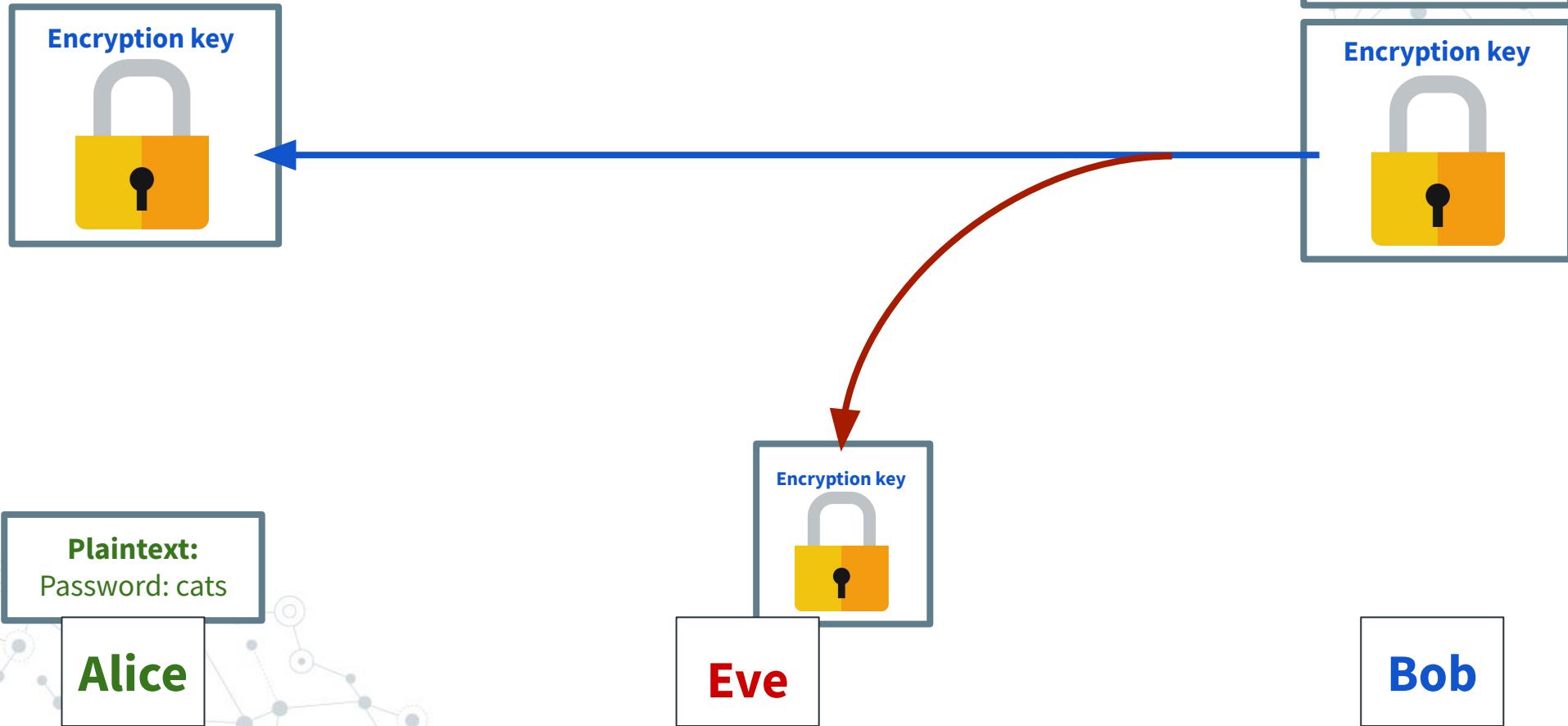
**Plaintext:**  
Password: cats

**Alice**

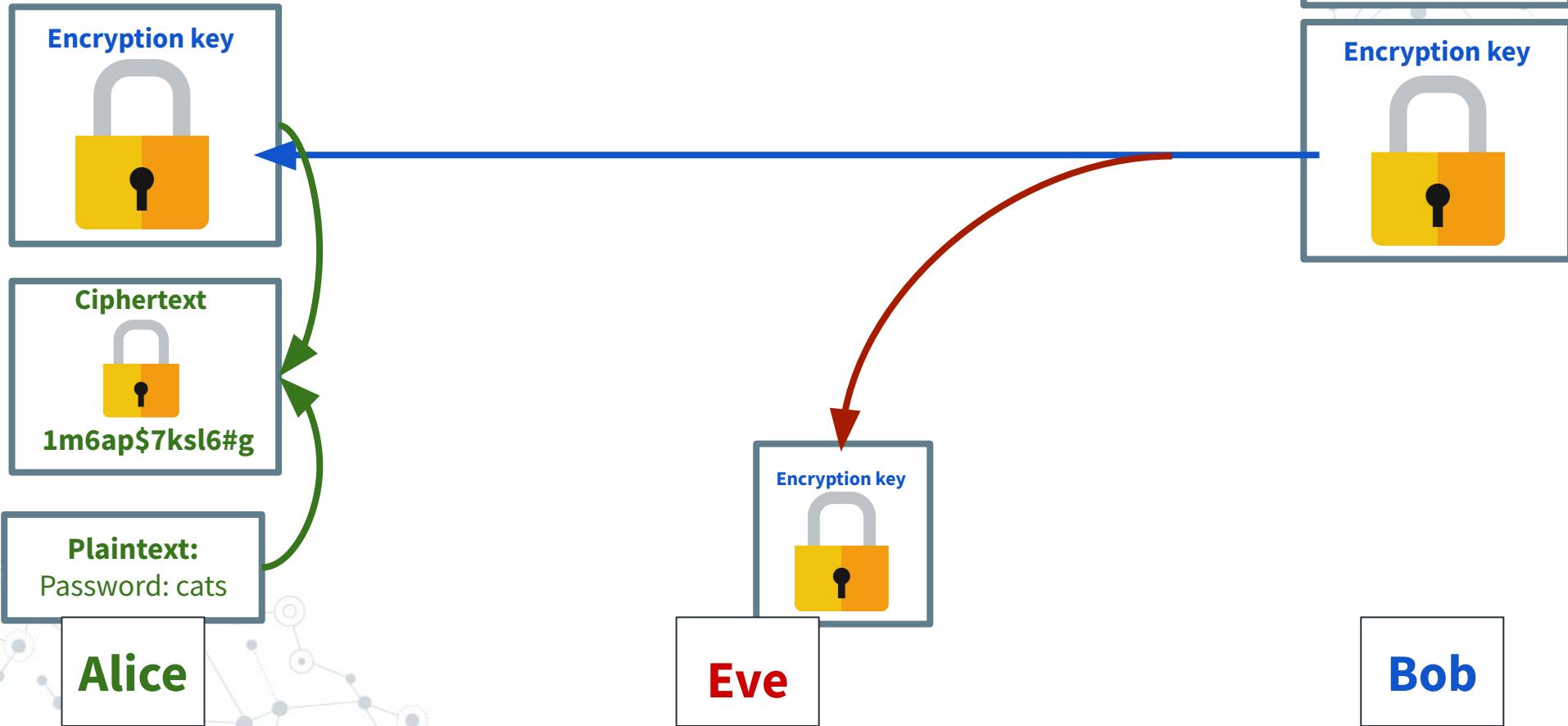
**Eve**

**Bob**

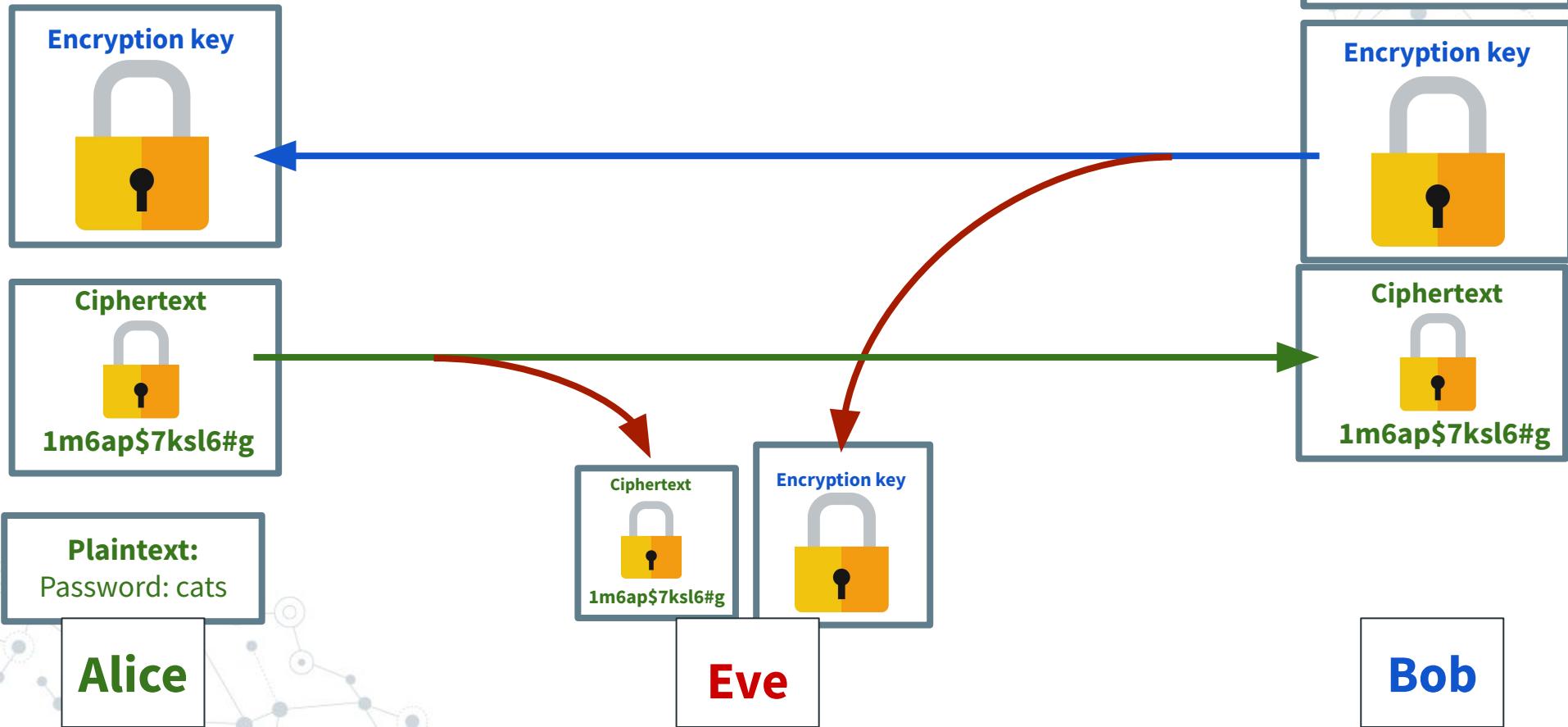
# Asymmetric Encryption



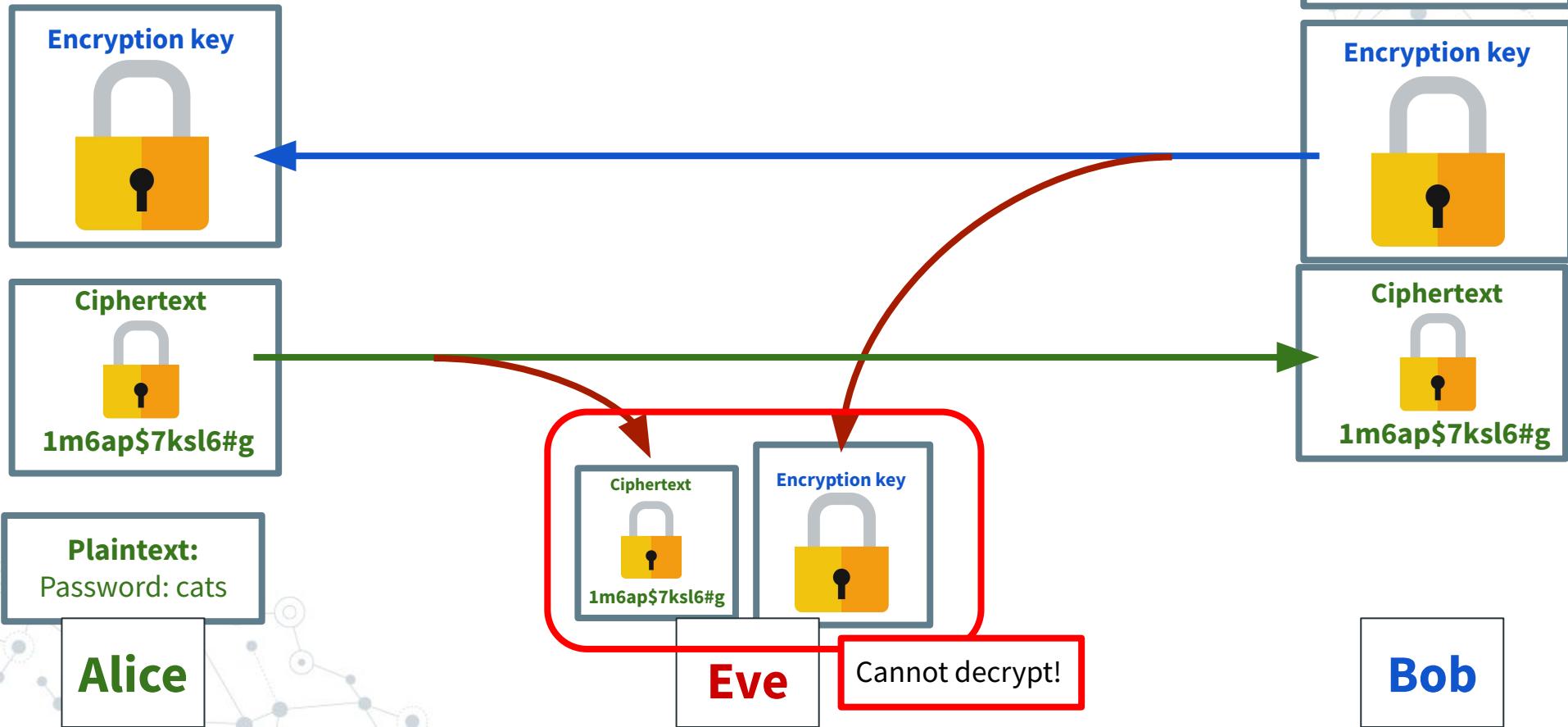
# Asymmetric Encryption



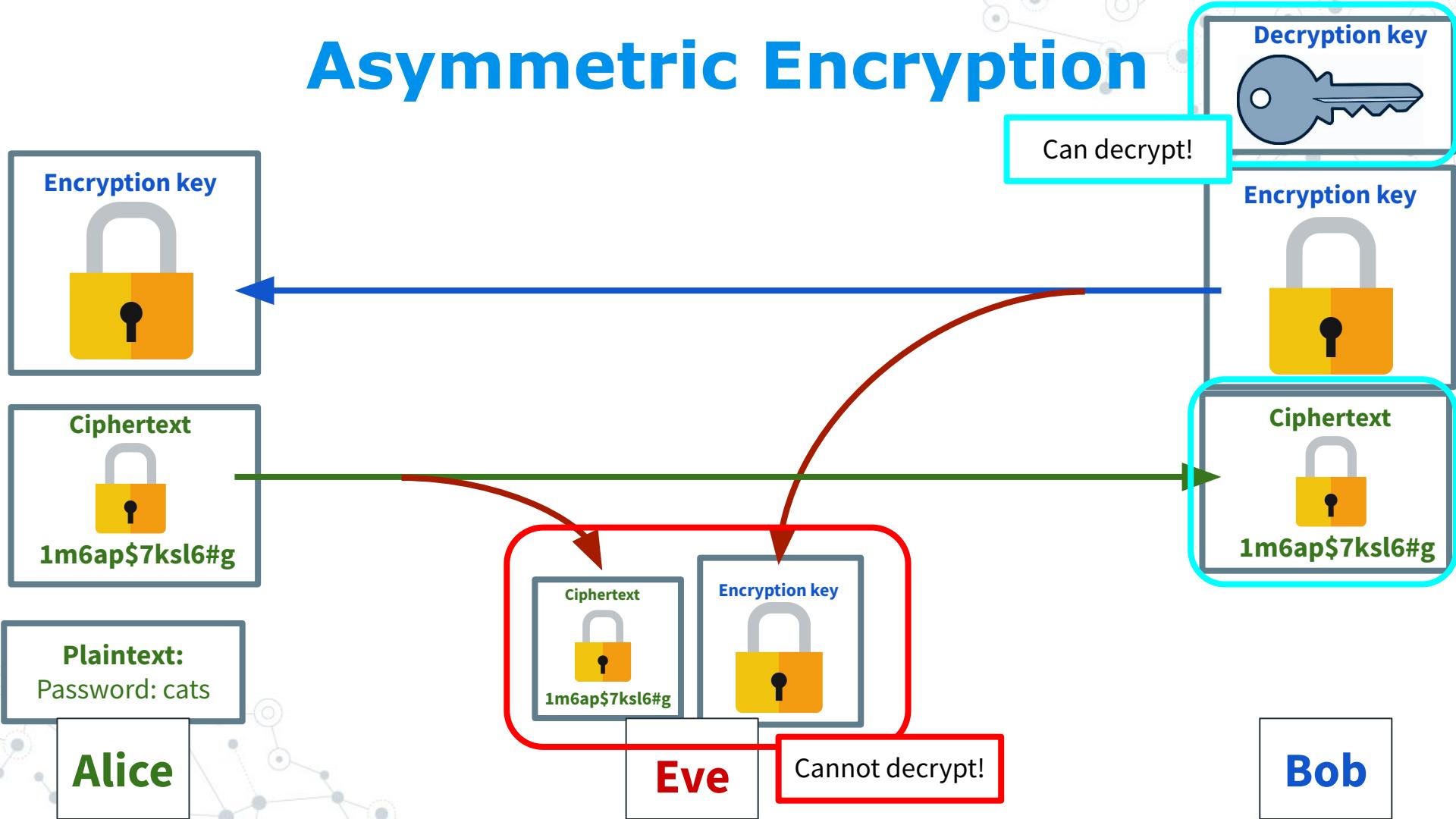
# Asymmetric Encryption



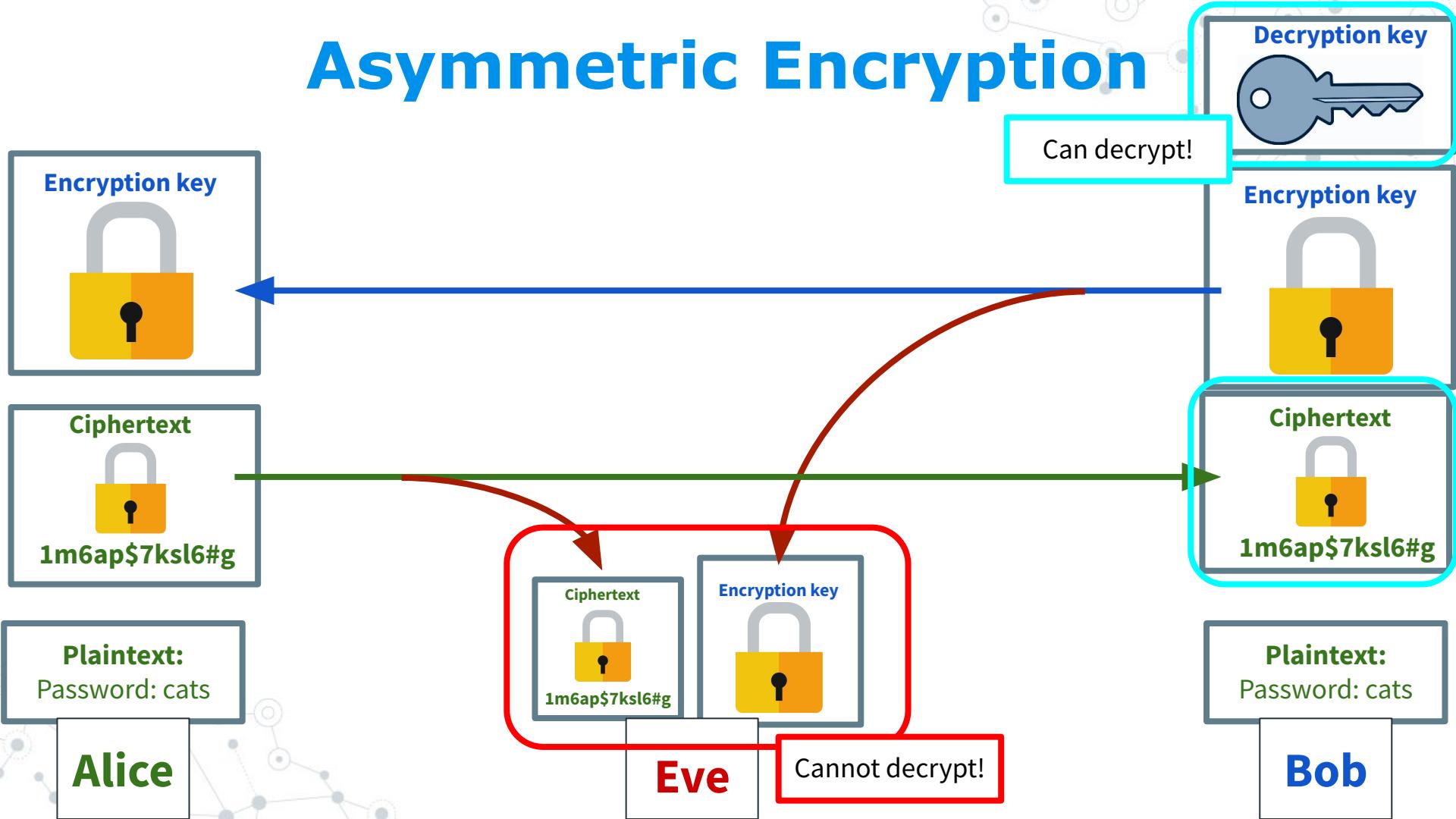
# Asymmetric Encryption



# Asymmetric Encryption



# Asymmetric Encryption



# RSA

**RSA** (Rivest–Shamir–Adleman): Asymmetric key algorithm

## Upside:

- Can send messages without needing a pre-shared key

## Downside:

- Relatively slow (creating a key can take minutes)
- Solution: Use once to send a symmetric key, and use that key from there

# RSA

For some numbers **N** and **E** (**E**nrypt), it is possible to compute another number **D** (**D**ecrypt) such that:

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

# RSA

For some numbers **N** and **E** (**E**nrypt), it is possible to compute another number **D** (**D**ecrypt) such that:

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

We can encrypt with E:

$$x^e \pmod{n} = ?$$

And decrypt with D:

$$?^d \pmod{n} = x$$

# RSA

For some numbers **N** and **E** (**E**nrypt), it is possible to compute another number **D** (**D**ecrypt) such that:

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

We can encrypt with E:

$$x^e \pmod{n} = ?$$

And decrypt with D:

$$?^d \pmod{n} = x$$

*Note: It is only possible to easily compute D if you know two factors of N. So if we multiply two large prime numbers together to get N, it is very hard to reverse!*

# RSA

N: 133

E (Encrypt): 5 

D (Decrypt): 65 

Plaintext: The number 35 (or '#' in ASCII)

# RSA

**N:** 133

**E (Encrypt):** 5 

**D (Decrypt):** 65 

**Plaintext:** The number **35** (or '#' in ASCII)

**Encrypt:**  $35^5 \text{ mod } 133 = 42$

# RSA

N: 133

E (Encrypt): 5 

D (Decrypt): 65 

Plaintext: The number 35 (or '#' in ASCII)

Encrypt:  $35^5 \text{ mod } 133 = 42$

Decrypt:  $42^{65} \text{ mod } 133 = 35$

# RSA

N: 133

E (Encrypt): 5 

D (Decrypt): 65 

Plaintext: The number 35 (or '#' in ASCII)

*Ciphertext*

Encrypt:  $35^5 \text{ mod } 133 = 42$  

Decrypt:  $42^{65} \text{ mod } 133 = 35$

# RSA

Can we reverse the original equation?

$$x^5 \bmod 133 = 42$$

# RSA



Can we reverse the original equation?

$$x^5 \bmod 133 = 42$$

X could be  $42^{1/5}$

X could be  $175^{1/5}$

X could be  $308^{1/5}$

...



# RSA

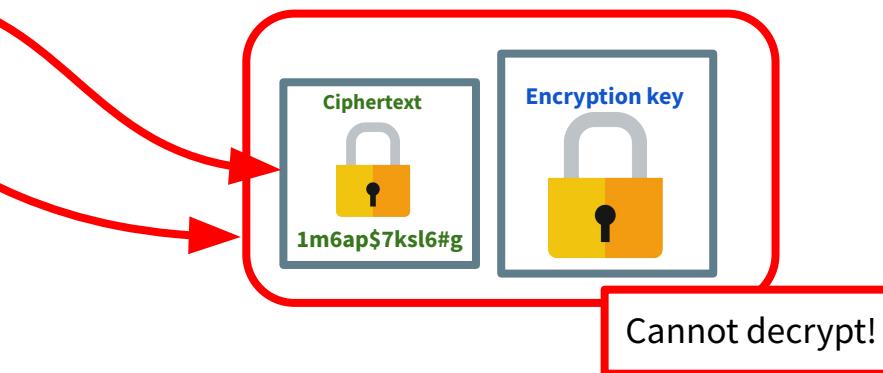
Can we reverse the original equation?

$$x^5 \bmod 133 = 42$$

X could be  $42^{1/5}$

X could be  $175^{1/5}$

X could be  $308^{1/5}$



# RSA

**What if we brute-force it?**



# RSA



## What if we brute-force it?

Normal key size:

```
1 044 388 881 413 152 506 691 752 710 716 624 382 579 964 249 047 383 780 384 233 ·  
483 283 953 907 971 557 456 848 826 811 934 997 558 340 890 106 714 439 262 837 ·  
987 573 438 185 793 607 263 236 087 851 365 277 945 956 976 543 709 998 340 361 ·  
590 134 383 718 314 428 070 011 855 946 226 376 318 839 397 712 745 672 334 684 ·  
344 586 617 496 807 908 705 803 704 071 284 048 740 118 609 114 467 977 783 598 ·  
029 006 686 938 976 881 787 785 946 905 630 190 260 940 599 579 453 432 823 469 ·  
303 026 696 443 059 025 015 972 399 867 714 215 541 693 835 559 885 291 486 318 ·  
237 914 434 496 734 087 811 872 639 496 475 100 189 041 349 008 417 061 675 093 ·  
668 333 850 551 032 972 088 269 550 769 983 616 369 411 933 015 213 796 825 837 ·  
188 091 833 656 751 221 318 492 846 368 125 550 225 998 300 412 344 784 862 595 ·  
674 492 194 617 023 806 505 913 245 610 825 731 835 380 087 608 622 102 834 270 ·  
197 698 202 313 169 017 678 006 675 195 485 079 921 636 419 370 285 375 124 784 ·  
014 907 159 135 459 982 790 513 399 611 551 794 271 106 831 134 090 584 272 884 ·  
279 791 554 849 782 954 323 534 517 065 223 269 061 394 905 987 693 002 122 963 ·  
395 687 782 878 948 440 616 007 412 945 674 919 823 050 571 642 377 154 816 321 ·  
380 631 045 902 916 136 926 708 342 856 440 730 447 899 971 901 781 465 763 473 ·  
223 850 267 253 059 899 795 996 090 799 469 201 774 624 817 718 449 867 455 659 ·  
250 178 329 070 473 119 433 165 550 807 568 221 846 571 746 373 296 884 912 819 ·  
520 317 457 002 440 926 616 910 874 148 385 078 411 929 804 522 981 857 338 977 ·  
648 103 126 085 903 001 302 413 467 189 726 673 216 491 511 131 602 920 781 738 ·  
033 436 090 243 804 708 340 403 154 190 336
```

Good luck!



# RSA

**Q: Can we calculate D, given N and E?**

# RSA

**Q: Can we calculate D, given N and E?**

A: Only with brute-force.

# RSA

**Q: Can we calculate D, given N and E?**

A: Only with brute-force.

**Q: Then how does anyone calculate D in the first place?**

# RSA

**Q: Can we calculate D, given N and E?**

A: Only with brute-force.

**Q: Then how does anyone calculate D in the first place?**

A: Well, it is easy if we know two factors of N.

# RSA



**Q: Can we calculate D, given N and E?**

A: Only with brute-force.

**Q: Then how does anyone calculate D in the first place?**

A: Well, it is easy if we know two factors of N.

**Q: Then can the attacker just find two factors of N?**



# RSA



**Q: Can we calculate D, given N and E?**

A: Only with brute-force.

**Q: Then how does anyone calculate D in the first place?**

A: Well, it is easy if we know two factors of N.

**Q: Then can the attacker just find two factors of N?**

A: Only with brute-force.



# RSA

## Takeaways:

- ◎ AES was a **symmetric encryption** algorithm. RSA is an **asymmetric encryption** algorithm
  - One (public) key only encrypts
  - One (private) key only decrypts
- ◎ Relies on math which is slow to brute-force.
- ◎ It is slow! Often used to just send symmetric keys, then an algorithm like AES is used for the bulk of the data.

# Feedback

<https://forms.gle/Xxi4JL25Vqg7U1pT8>  
*(In Canvas and Syllabus)*

# Data Security: Day 3

Wrap up of encryption  
Integrity checks

# Quiz Answers

1. Encryption primarily protects
  - a. Confidentiality
  - b. Integrity
  - c. Availability
2. Common drawbacks of symmetric encryption include:
  - a. The secret key must be shared beforehand
  - b. The maximum ciphertext length is limited by the size of the secret key

The encryption is easily broken by modern tools

# Quiz Answers

1. Encryption primarily protects
  - a. **Confidentiality**
  - b. Integrity
  - c. Availability
2. Common drawbacks of symmetric encryption include:
  - a. The secret key must be shared beforehand
  - b. The maximum ciphertext length is limited by the size of the secret key

The encryption is easily broken by modern tools

# Quiz Answers

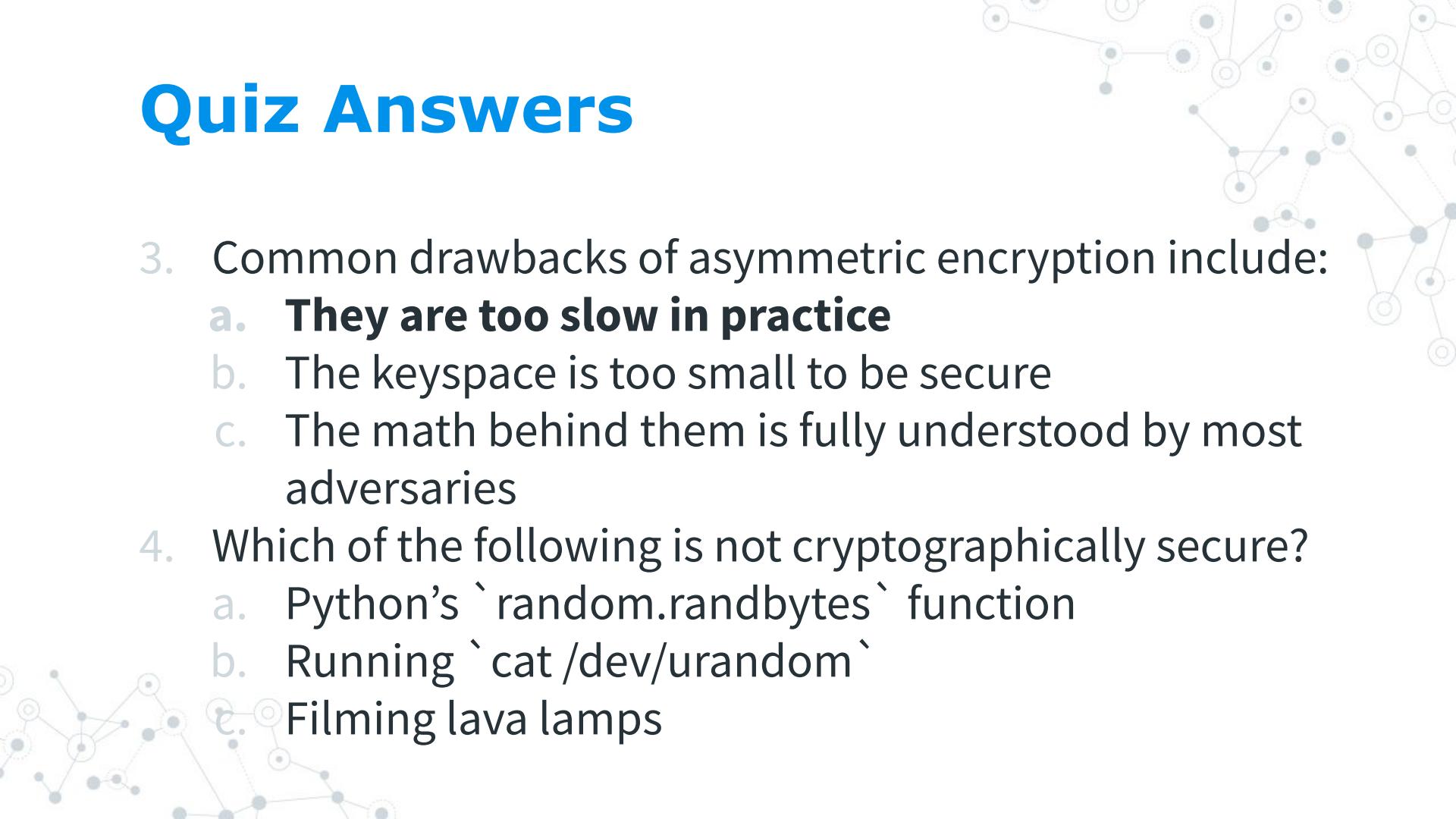
1. Encryption primarily protects
  - a. **Confidentiality**
  - b. Integrity
  - c. Availability
2. Common drawbacks of symmetric encryption include:
  - a. **The secret key must be shared beforehand**
  - b. The maximum ciphertext length is limited by the size of the secret key

The encryption is easily broken by modern tools

# Quiz Answers

3. Common drawbacks of asymmetric encryption include:
  - a. They are too slow in practice
  - b. The keyspace is too small to be secure
  - c. The math behind them is fully understood by most adversaries
4. Which of the following is not cryptographically secure?
  - a. Python's `random.randbytes` function
  - b. Running `cat /dev/urandom`
  - c. Filming lava lamps

# Quiz Answers

- 
3. Common drawbacks of asymmetric encryption include:
    - a. **They are too slow in practice**
    - b. The keyspace is too small to be secure
    - c. The math behind them is fully understood by most adversaries
  4. Which of the following is not cryptographically secure?
    - a. Python's `random.randbytes` function
    - b. Running `cat /dev/urandom`
    - c. Filming lava lamps

# Quiz Answers

3. Common drawbacks of asymmetric encryption include:
  - a. **They are too slow in practice**
  - b. The keyspace is too small to be secure
  - c. The math behind them is fully understood by most adversaries
4. Which of the following is not cryptographically secure?
  - a. **Python's `random.randbytes` function**
  - b. Running `cat /dev/urandom`
  - c. Filming lava lamps

# Quiz Answers

5. Ransomware uses cryptography maliciously to violate
  - a. Confidentiality
  - b. Integrity
  - c. Availability

# Quiz Answers

5. Ransomware uses cryptography maliciously to violate
  - a. Confidentiality
  - b. Integrity
  - c. **Availability**

# Feedback

Sounds like things are going well? Cool.

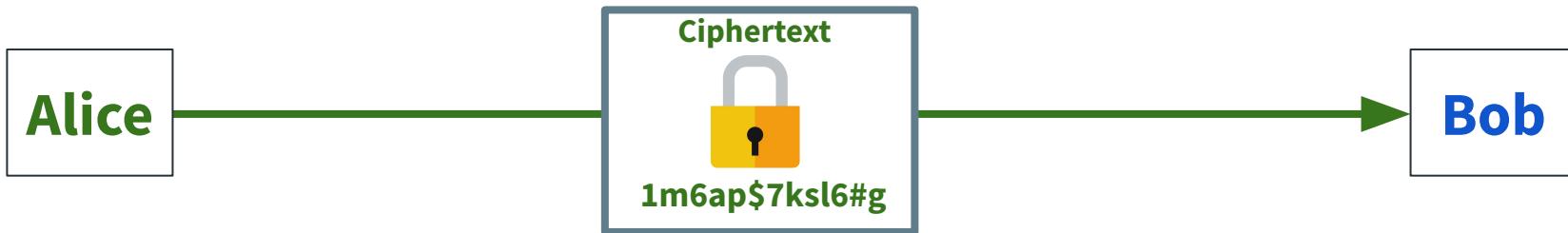
- **Slides:** Does Google Docs work or would you prefer a PDF? And one per day, per week, or per unit?
- **Previous exams:** The class has greatly changed, but I can aggregate relevant questions.
- **Pace:** A bit fast? I will try to slow it down.

*FYI: The feedback link in the syllabus is always active.*

# Wrap-up of encryption

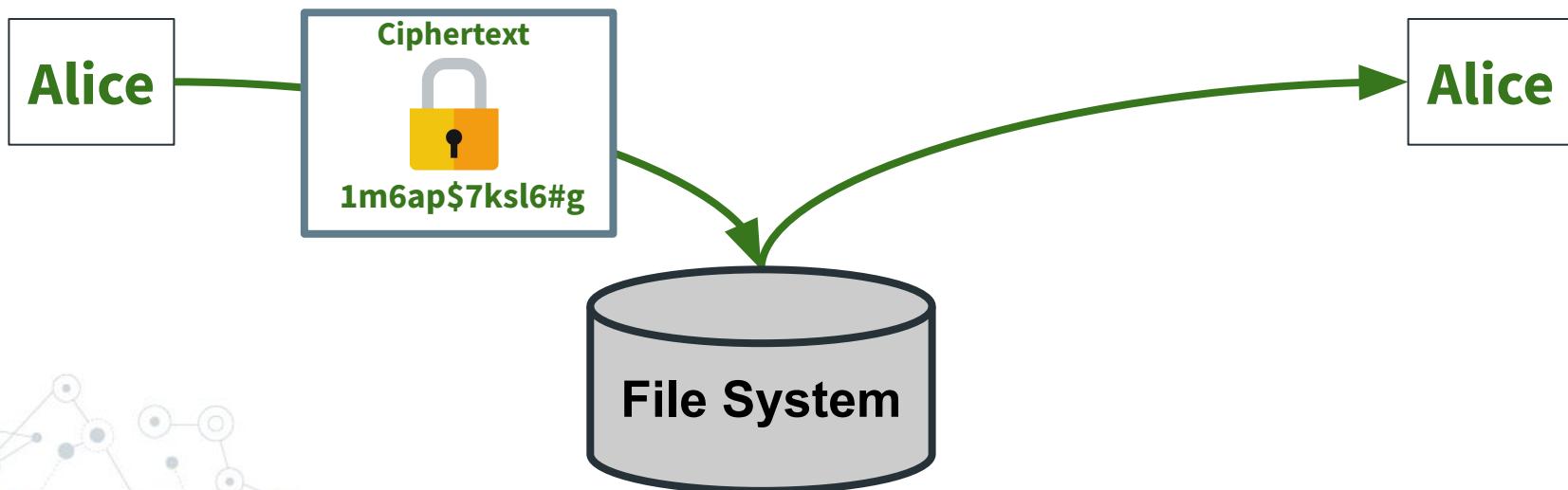
# Encryption in transit

Encryption in transit: Sent between two parties



# Encryption at rest

Encryption at rest: Used on storage media to keep contents secure in case of physical access.



# Encryption at rest

## Apple Fights Court Order to Unlock San Bernardino Shooter's iPhone

19 février 2016



On the evening of February 16th, US Magistrate Sheri Pym ordered Apple to provide "reasonable technical assistance" to the FBI, namely in the form of software or a tool that can disable the security function that erases data from the iPhone of one of the San Bernardino shooters after too many failed attempts to unlock it. The court order comes after the [shooting incident in San Bernardino, California](#), where two shooters, Tashfeen Malik and Syed Farook, killed 14 people in December 2015.



### En rapport

- ✖ Online Dating Websites Lure Japanese Customers to Scams
- ✖ Data of U.K. Train Commuters
- ✖ Leak from Misconfigured AWS Cloud Storage
- ✖ Misconfigured AWS S3 Bucket Leaks 36,000 Inmate Records
- ✖ Unsecured AWS S3 Bucket Found Leaking Data of Over 30K

<https://www.trendmicro.com/vinfo/fr/security/news/online-privacy/apple-fights-court-order-to-unlock-shooters-iphone>

# Encryption at rest

Home » wallets » \$220M in Bitcoin May Be Encrypted Forever on IronKey



## \$220M in Bitcoin May Be Encrypted Forever on IronKey

FIPS Certified Tamperproof Hardware Encryption Proves Unbreakable after 10 Years

<https://ciphertrace.com/220m-in-bitcoin-encrypted-forever-on-ironkey/>

# Recap

## Encryption at rest

- ◎ Protects files on disk
  - All secure apps *should* use it (e.g. password managers)
  - Options exist for most platforms: Bitlocker (Windows), FileVault (Mac), etc...
- ◎ Very important for businesses with lots of user data!
- ◎ Average consumers need to balance it with the risk of losing access to their data

# **Encryption problems and vulnerabilities**

# Encryption problems

Just not using encryption :(

- HTTP websites (*no lock icon, more rare nowadays*)
- SMS (*Signal or Whatsapp are secure alternatives*)
- About 10% of email clients
- Old services like FTP
- Most devices / hard drives

# Encryption problems



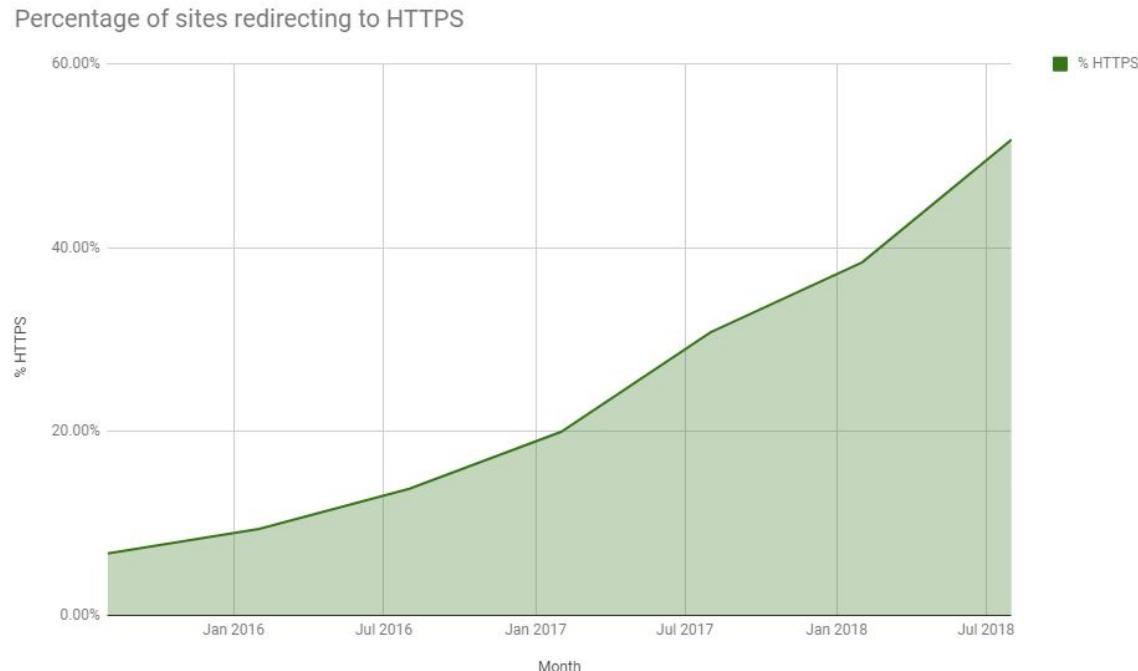
<https://www.cbsnews.com/news/equifax-ex-ceo-hacked-data-wasnt-encrypted/>

## Watchdog report says mandatory Olympic Games app has 'devastating' security flaw

MY2022, a mandatory app for all attendees of the 2022 Olympic Games, has a simple but "devastating" flaw where encryption protecting users' voice audio and file transfers can easily be hacked and sidestepped.

<https://thehill.com/changing-america/enrichment/education/590314-watchdog-report-says-mandatory-olympic-games-app-has>

# Encryption problems



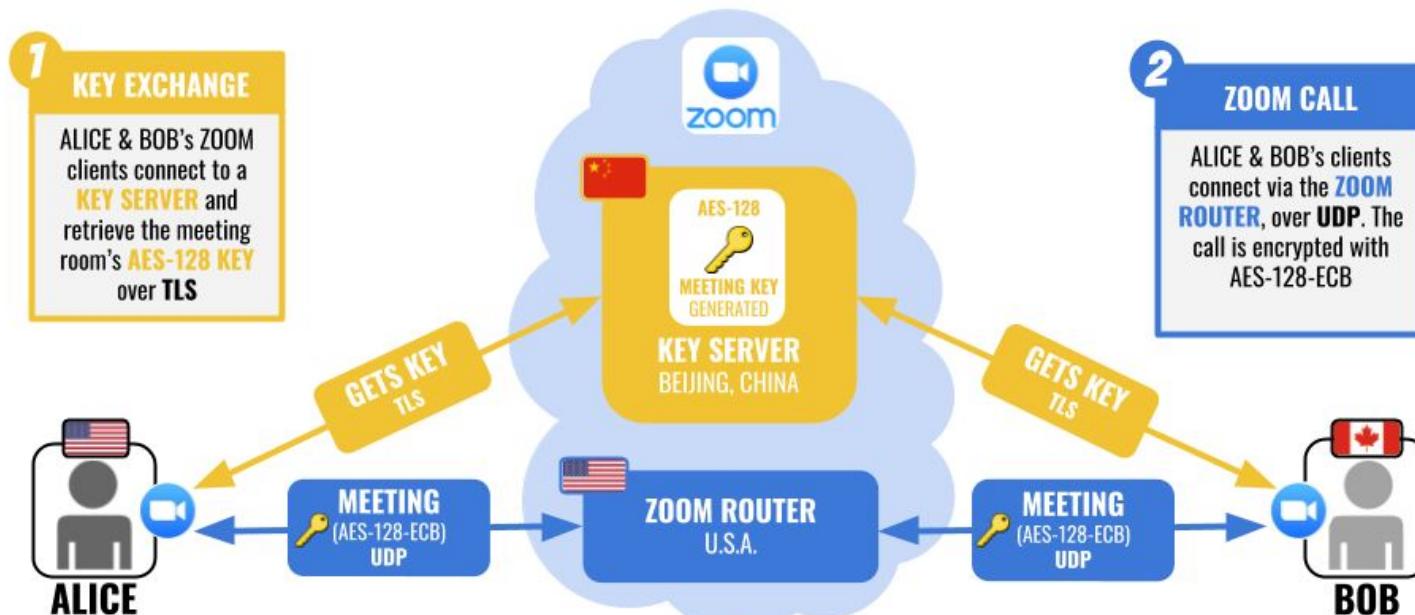
<https://www.welivesecurity.com/2018/09/03/majority-worlds-top-websites-https/>

# Encryption problems

- ◎ Not understanding / applying encryption correctly (deprecated standards, bad randomness, poor key management, etc)
- ◎ Trying to build custom encryption rather than using existing, proven solutions
  - Don't roll your own crypto! Unless it is for fun.

# Biggest problems

## OBSERVING A TEST ZOOM CALL



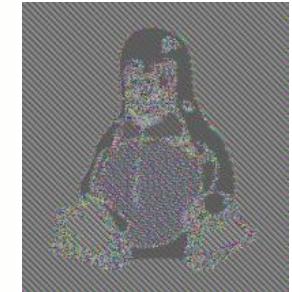
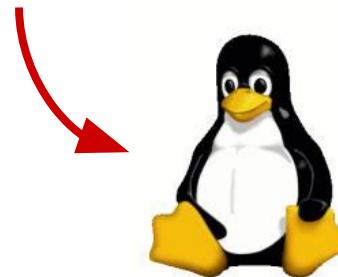
**NOTE:** Citizen Lab observed these server locations during a test call. Other ZOOM calls may use servers and call routers in other locations.

<https://citizenlab.ca/2020/04/move-fast-roll-your-own-crypto-a-quick-look-at-the-confidentiality-of-zoom-meetings/>

# Biggest problems

From that report on Zoom encryption failures:

- ◎ [In] each Zoom meeting, a single AES-128 key is used in ECB mode by all participants to encrypt and decrypt audio and video [...] ECB mode is not recommended because patterns present in the plaintext are preserved during encryption.



# Recap

- ➊ **Encryption at rest:** Encrypting a disk in case it gets stolen
- ➋ Biggest modern encryption failure is just not using it properly
  - Not encrypting at all
  - Disclosing “secret” keys

# Recap

## Questions?



<https://twitter.com/secparam/status/1382075663344095234>

# Integrity Checks

# Alice and Bob

Alice

Bob

*Hello! What is my bank balance right now?*

*You have \$1,436 in your account.*

*Great. Can you transfer \$30 to Carol?*

# Alice and Bob

*Hello! What is my bank balance right now?*

Alice

*You have \$1,436 in your account.*

Bob

Eve

*Great. Can you transfer \$30 to Carol?*

# Alice and Bob

*Hello! What is my bank balance right now?*

Alice

*You have \$1,436 in your account.*

Bob

Eve

*Great. Can you transfer \$1,436 to Eve?*

# Alice and Bob

*Hello! What is my bank balance right now?*

**Wait, didn't we just solve this with encryption?**

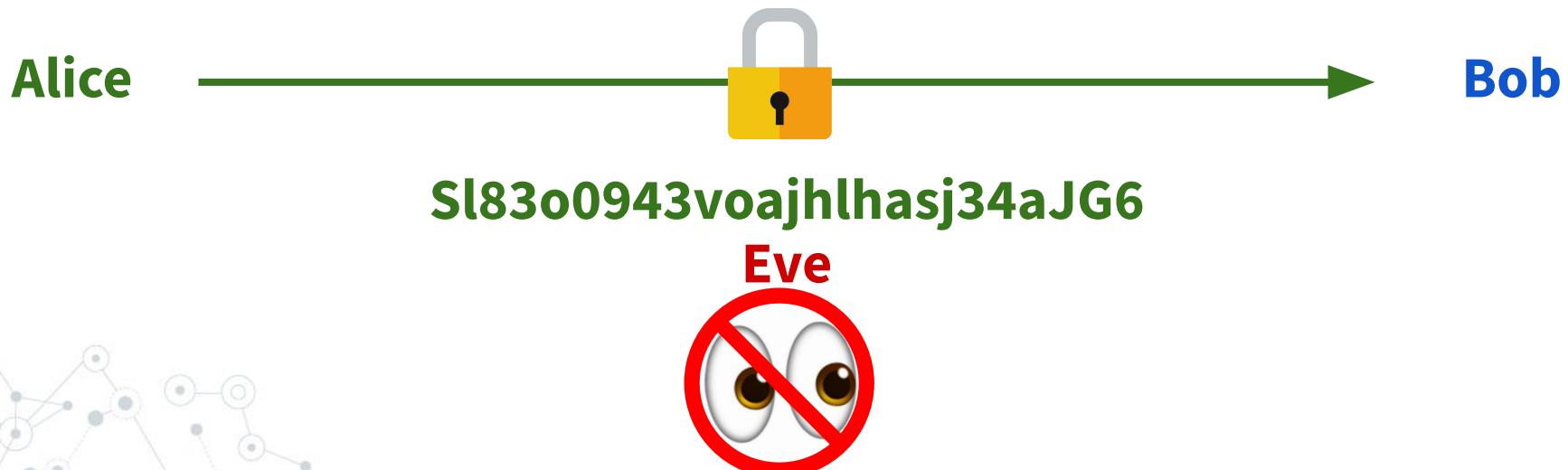
Alice

Bob

Eve

*Great. Can you transfer \$1,436 to Eve?*

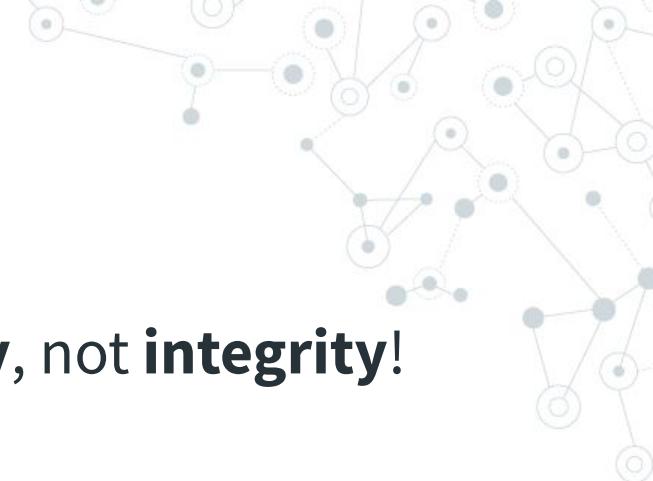
# Message integrity



# Message integrity



# Message integrity



Encryption only provides **confidentiality**, not **integrity**!



# Integrity Attacks

Also, it has many other uses where encryption is not an option

- Verifying the legitimacy of a downloaded file
- Public, unencrypted messages (also Blockchain I guess)
- Error correction

# Message hashing

**Message Hash:** Fixed-length string computed from the message, similar to a hashtable

- Common algorithms: MD5, SHA1, SHA256

*Message: Transfer \$30 to Carol*

*MD5(message): 78adcbc2cf8d0c67b9eb75f3c93ded18*

# Message hashing

Alice

Bob

*Message: Transfer \$30 to Carol*

*MD5(message): 78adcbc2cf8d0c67b9eb75f3c93ded18*



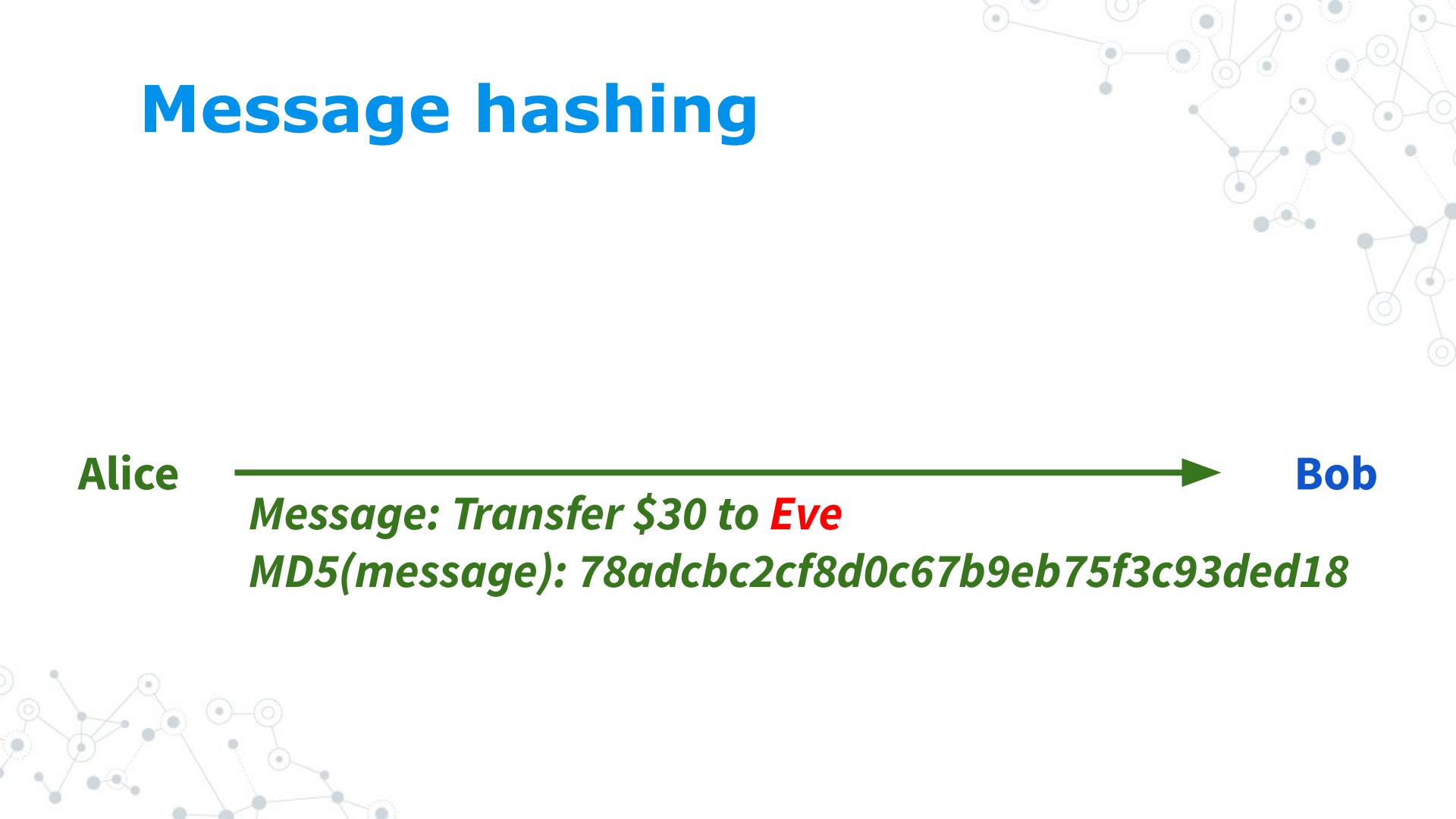
# Message hashing

Alice

Bob

*Message: Transfer \$30 to Eve*

*MD5(message): 78adcbc2cf8d0c67b9eb75f3c93ded18*



# Message hashing

If Bob computes a hash which doesn't match the one Alice sent, the message may have been tampered with!

Alice

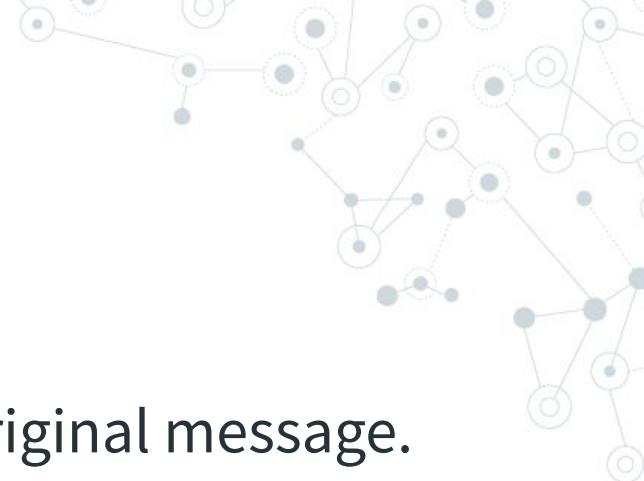
*Message: Transfer \$30 to Eve*

*MD5(message): 78adcbec2cf8d0e67b9eb75f3e93ded18*

Bob

*MD5(message): fb967eb9e36bd2903f2459ecfcaecd61*

# Message hashing



## Hashing is not encryption!

- ➊ You cannot reverse these to get the original message.

The screenshot shows a portion of the Proofpoint website. At the top, there is a navigation bar with the logo "proofpoint.", followed by dropdown menus for "Products", "Solutions", "Partners", "Resources", and "Company". To the right of the menu are a search icon, a globe icon, a "LOGIN" button, and a blue "CONTACT" button. Below the navigation bar, there is a section of text about AES and a list of encryption algorithms:

AES is considered resistant to all attacks, with the exception of brute-force attacks, which attempt to decipher messages using all possible combinations in the 128-, 192- or 256-bit cipher. Still, security experts believe that AES will eventually become the standard for encrypting data in the private sector.

[Twofish encryption algorithm](#)

[Blowfish encryption algorithm](#)

[IDEA encryption algorithm](#)

[MD5 encryption algorithm](#)

[HMAC encryption algorithm](#)

# Message hashing

Home > Download

## Thank you for downloading MuseScore

Your download will start in 0 seconds. Problems with the download? Please use this [direct link](#)

SHA256 Checksum: fa3ca0f8cc5b0e8b0c0bb8ef11e227b9b27b2a5c9da28dab58bafccb0eb657d0



# Message hashing

## Notation:

- $m$  = message, e.g. “Transfer \$30 to Carol”
- $H(m)$  = hash of the message, e.g. “78adcbe2cf8...93ded18”

## Hash Algorithm Requirements

- Different messages should never have the same hash
- Changing the input should greatly change the output
  - $MD5("Test1")$ : fa029a7f2a3ca5a03fe682d3b77c7f0d
  - $MD5("Test2")$ : 856babf68edfd13e2fd019df330e11c5

# Message hashing

**Problem:** What if two hashes **collide**?

$$H(\text{"something good"}) == H(\text{"something bad"})$$

# Message hashing

**Problem:** What if two hashes **collide**?

$$H(\text{"something good"}) == H(\text{"something bad"})$$

- ➊ Eve could swap one message for the other.

# Message hashing

**Problem:** What if two hashes **collide**?

$$H(\text{"something good"}) == H(\text{"something bad"})$$

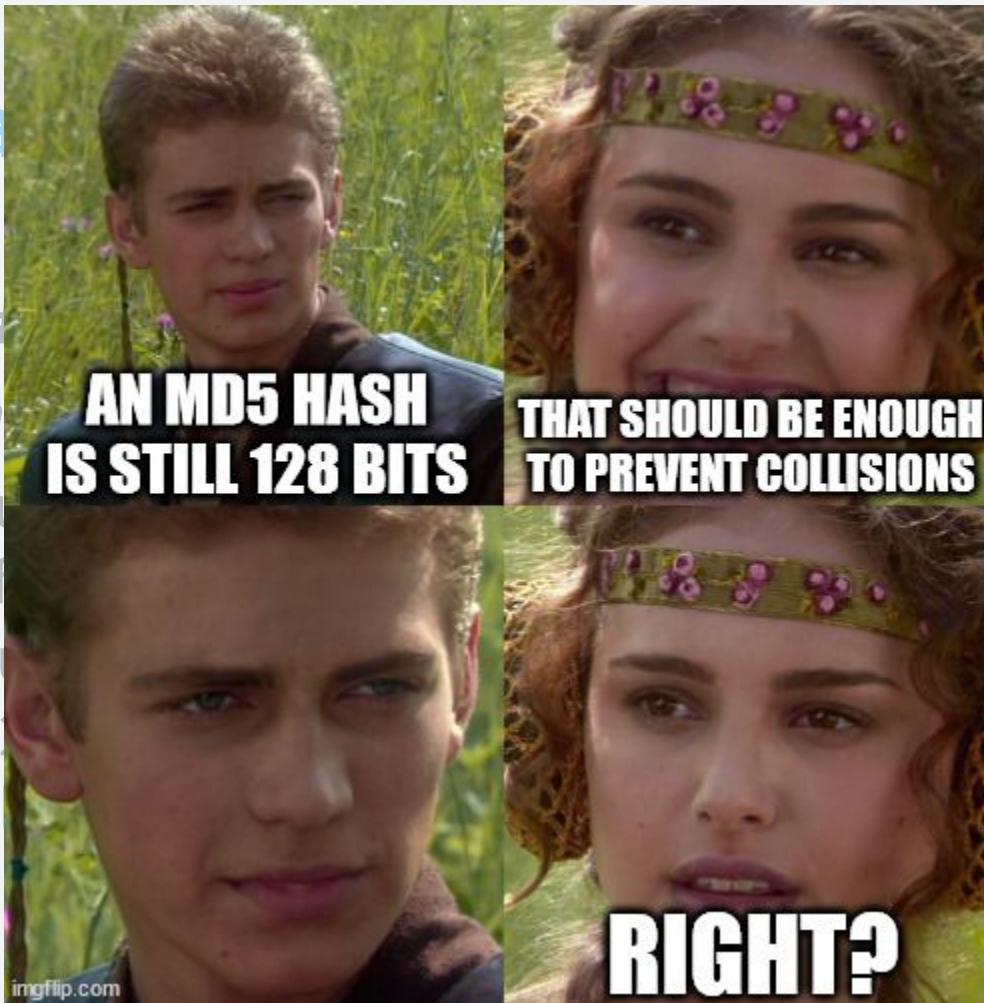
- ➊ Eve could swap one message for the other.
- ➋ Thankfully, MD5 is 128 bits, and SHA1 is 160 bits, and other algorithms are larger. At a minimum, there are 340,282,366,920,938,463,463,374,607,431,768,211,456 possible hash options, and they should rarely collide!

# Messa

## Problem #

$H("something")$

- Eve could
- Thankf
- minimum
- 340,281,000  
possible



bits. At a

8,211,456  
y collide!

# Message hashing

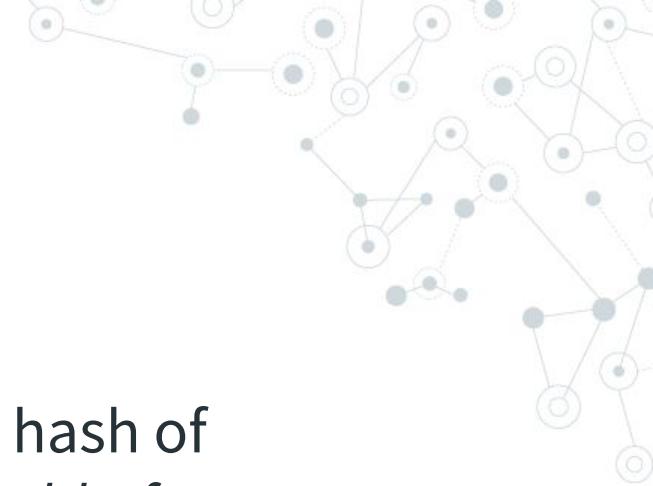
**Problem:** Turns out it is actually possible to generate two hashes that collide!

- The math is a whole PhD thesis, but it can be done

Table 5: Linear part message bit relations for the second block path.

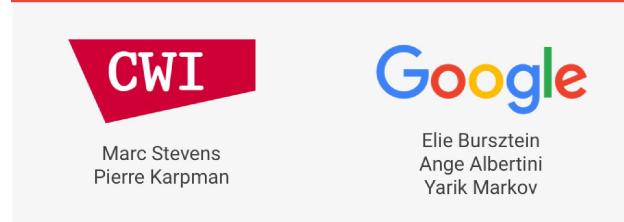
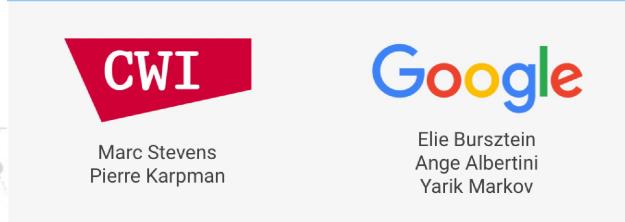
$m_{23}[27] \oplus m_{23}[28] = 1$	$m_{23}[30] \oplus m_{24}[3] = 1$	$m_{23}[30] \oplus m_{28}[28] = 1$
$m_{23}[4] = 0$	$m_{24}[28] = 0$	$m_{24}[29] = 0$
$m_{24}[2] = 0$	$m_{26}[28] \oplus m_{26}[29] = 1$	$m_{27}[29] = 0$
$m_{28}[27] = 0$	$m_{28}[4] \oplus m_{32}[29] = 0$	$m_{36}[4] \oplus m_{44}[28] = 1$
$m_{38}[4] \oplus m_{44}[28] = 0$	$m_{39}[30] \oplus m_{44}[28] = 1$	$m_{40}[3] \oplus m_{44}[28] = 0$
$m_{40}[4] \oplus m_{44}[28] = 1$	$m_{41}[29] \oplus m_{41}[30] = 0$	$m_{42}[28] \oplus m_{44}[28] = 0$
$m_{43}[28] \oplus m_{44}[28] = 0$	$m_{43}[29] \oplus m_{44}[28] = 1$	$m_{43}[4] \oplus m_{47}[29] = 0$
$m_{44}[28] \oplus m_{44}[29] = 1$	$m_{45}[29] \oplus m_{47}[29] = 0$	$m_{46}[29] \oplus m_{47}[29] = 0$
$m_{48}[4] \oplus m_{52}[29] = 0$	$m_{50}[29] \oplus m_{52}[29] = 0$	$m_{51}[29] \oplus m_{52}[29] = 0$
$m_{54}[4] \oplus m_{60}[29] = 1$	$m_{56}[29] \oplus m_{60}[29] = 1$	$m_{56}[4] \oplus m_{60}[29] = 0$
$m_{57}[29] \oplus m_{60}[29] = 1$	$m_{59}[29] \oplus m_{60}[29] = 0$	$m_{67}[0] \oplus m_{72}[30] = 1$
$m_{68}[5] \oplus m_{72}[30] = 0$	$m_{70}[1] \oplus m_{71}[6] = 1$	$m_{71}[0] \oplus m_{76}[30] = 1$
$m_{72}[5] \oplus m_{76}[30] = 0$	$m_{73}[2] \oplus m_{78}[0] = 1$	$m_{74}[1] \oplus m_{75}[6] = 1$
$m_{74}[7] \oplus m_{78}[0] = 0$	$m_{75}[1] \oplus m_{76}[6] = 1$	$m_{76}[0] \oplus m_{76}[1] = 1$
$m_{76}[3] = 1$	$m_{77}[0] \oplus m_{77}[1] = 0$	$m_{77}[0] \oplus m_{77}[2] = 1$
$m_{77}[8] = 0$	$m_{78}[3] = 1$	$m_{78}[7] = 0$
$m_{79}[2] = 0$	$m_{79}[4] = 1$	

# Message hashing



## SHAttered: Breaking SHA1

- Both these PDFs have the same SHA1 hash of  
*38762cf7f55934b34d179ae6a4c80cadccbb7f0a*



# Message hashing

**MD5:** Easily broken at this point

- ◎ Fell out of use around 15 years ago

**SHA1:** Broken, but not as easily

- ◎ Fell out of use around 5 years ago

**SHA256:** Safe... for now

# Message hashing

## Recap

- **Hash algorithms:** Takes a message, returns a fixed-length string based on that message
  - Used to verify **integrity**
  - Common ones are MD5, SHA1, SHA256
- **Hash collision:** Different inputs that map to one output
- MD5 and SHA1 are considered “broken” because attackers can generate collisions  
Most common now is SHA256

# Message hashing

**Problem:** Eve can just change the hash



# Message hashing

**Problem:** Eve can just change the hash

Alice



Bob

*m: Transfer \$30 to Carol*

*H(m): 78adcbe2cf8d0c67b9eb75f3c93ded18*

# Message hashing

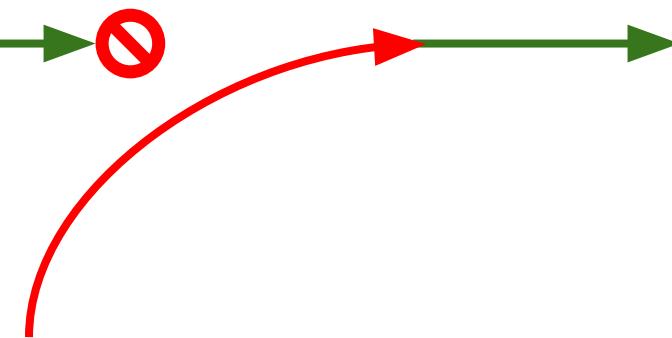
**Problem:** Eve can just change the hash

Alice

*m: Transfer \$30 to Carol*

*H(m): 78adc...18*

Bob



*m: Transfer \$30 to Eve*

*H(m): fb967eb...61*

Eve

# MACs and HMACs

**Message Authentication Code (MAC):** Like a hash, but using a secret

**Hash-based Message Authentication Code (HMAC):** An extension to a hash that turns it into a MAC

**HMAC example:**  $H(\text{secret} + m)$

# MACs and HMACs

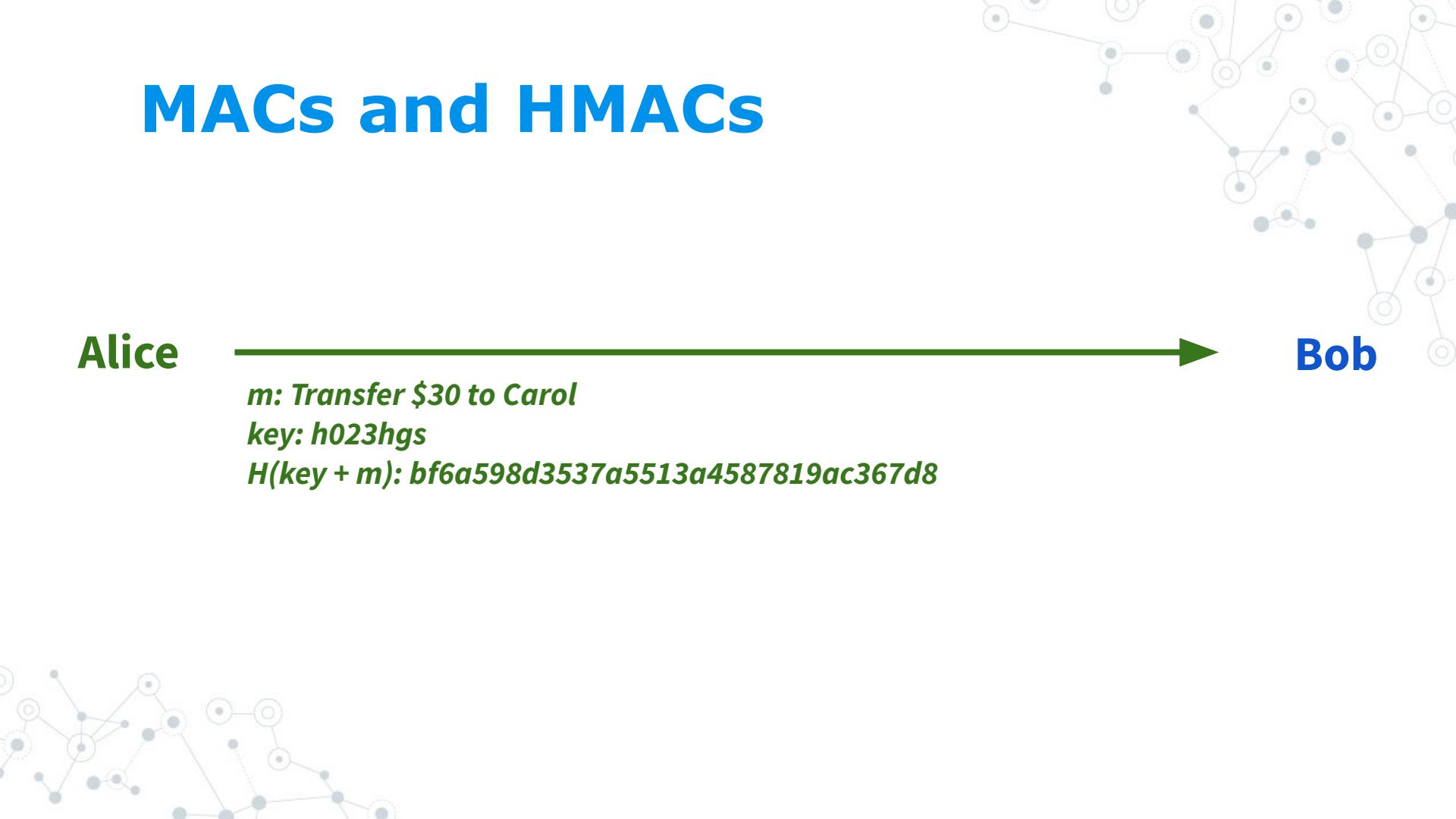
Alice

*m: Transfer \$30 to Carol*

*key: h023hgs*

*H(key + m): bf6a598d3537a5513a4587819ac367d8*

Bob



# MACs and HMACs

Alice

*m: Transfer \$30 to Carol*

*key: h023hgs*

*H(key + m): bf6a598d3537a5513a4587819ac367d8*

Bob

*m: Transfer \$30 to Eve*

*key: ???*

*H(key + m): ???*

Eve

# MACs and HMACs

*Bob can still verify if he knows the key!*

Alice

$H(key + m): bf6a598d3537a5513a4587819ac367d8$

Bob

$m: Transfer \$30 to Carol$   
 $key: h023hgs$   
 $H(key + m): bf6a598d3537a5513a4587819ac367d8$

$m: Transfer \$30 to Eve$   
 $key: ???$   
 $H(key + m): ???$

Eve

*Eve cannot calculate a hash without the key*

# MACs and HMACs

These are standard and widely implemented, do not build them yourself (no matter what StackOverflow says)

```
# Wrong
import hashlib
hashlib.md5(b'Pa$$w0rD' + b'Transfer $30 to Carol').hexdigest()
```

```
# Right
import hmac
hmac.new(b'Pa$$w0rD', b'Transfer $30 to Carol').hexdigest()
```

To compute the SHA256 HMAC of a string in PHP, do this:

```
$signature = hash_hmac('sha256', $request_body, $hmac_key);
```

To compute the SHA256 HMAC of a string in Python, do this:

```
from subprocess import check_output

signature = check_output(
    [
        "php",
        "-r",
        "echo hash_hmac('sha256', $argv[1], $argv[2]);",
        "--",
        request_body,
        hmac_key
    ])
```

Other languages often provide similar support.

# MACs and HMACs

## Recap:

- ◎ **Message Authentication Codes (MACs)**: similar to hashes, but require a secret as well.
  - Prevents attackers from generating their own
- ◎ **Hashed Message Authentication Codes (HMACs)**: the most common form of MACs, created by adding a secret to a hash function like  $MD5(\text{secret} + \text{message})$

# MACs and HMACs

**Problem:** What if an attacker can generate a valid hash without knowing the secret?

# MACs and HMACs

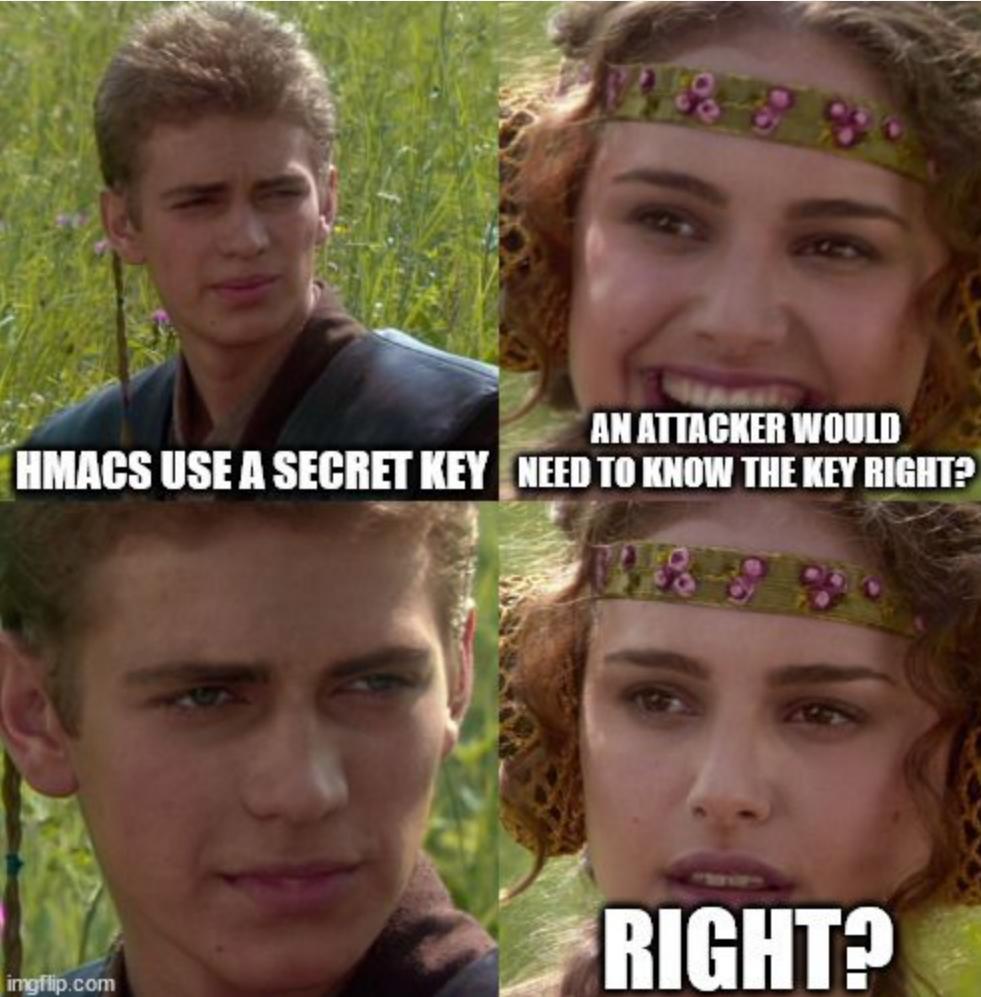
**Problem:** What if an attacker can generate a valid hash without knowing the secret?

- ◎ I mean... no? We have established that is not how it works, right?

# MACs a

Problem: W  
without kno

- ◎ I mean.  
works, r



# Length Extension Attacks



**Problem:** The way some hashes are constructed allows them to be updated, without knowing the original text!

## Alice

*m: Transfer \$30 to Carol*

*key: h023hgs*

*H(key + m): bf6a598d3537a5513a4587819ac367d8*

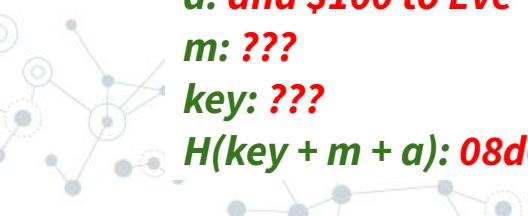
## Eve

*a: and \$100 to Eve*

*m: ???*

*key: ???*

*H(key + m + a): 08d0975bc3f68aa5ef2ed497deaca698*



# Length Extension Attacks



**Problem:** The way some hashes are constructed allows them to be updated, without knowing the original text!

**Alice**

*m: Transfer \$30 to Carol*

*key: h023hgs*

*H(key + m): bf6a598d3537a5513a4587819ac367d8*

**Eve**

*a: and \$100 to Eve*

*m: ???*

*key: ???*

*H(key + m + a): 08d0975bc3f68aa5ef2ed497deaca698*

*Valid hash for*

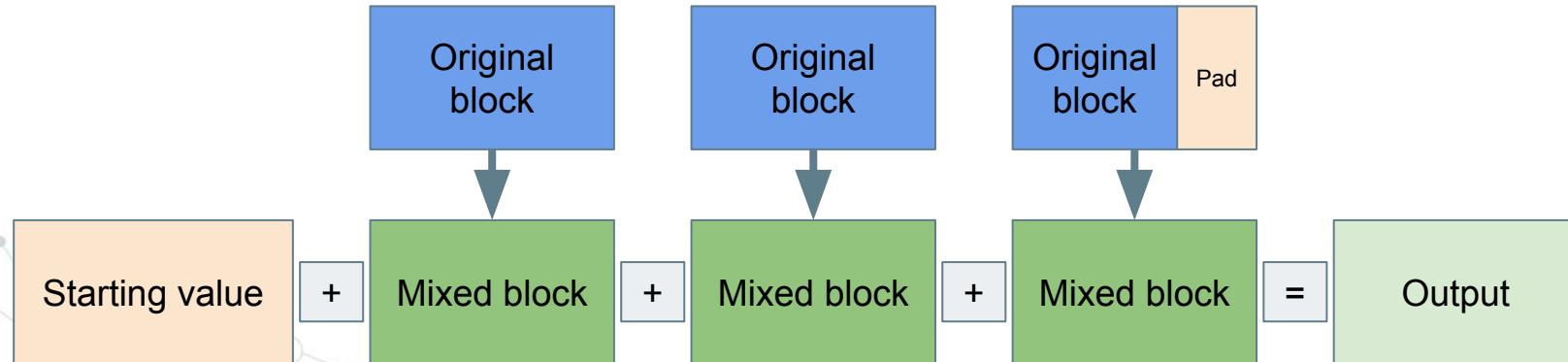
*"h023hgsTransfer \$30 to Carol and \$100 to Eve"*



# Length Extension Attacks

## Merkle-Damgård construction:

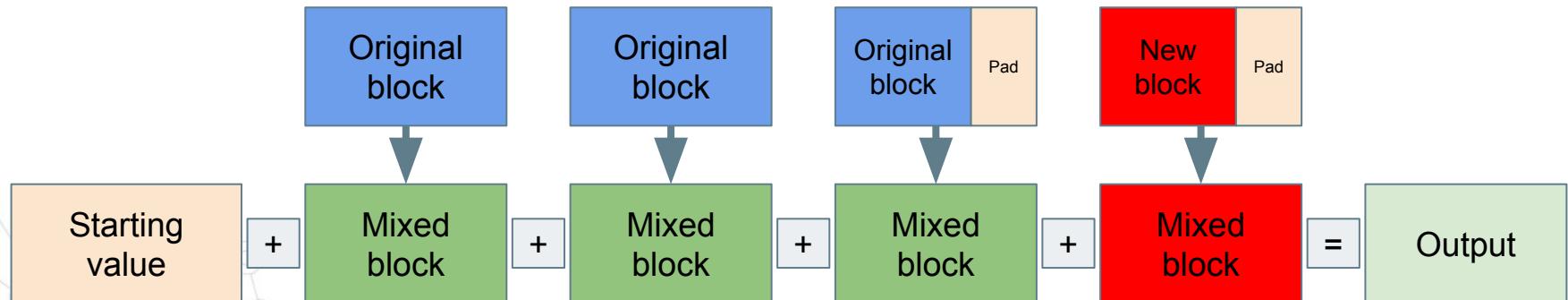
- Split text into fixed-size blocks (pad the end if needed)
- Mix up each block (in a deterministic way)
- Add each block to the previous one



# Length Extension Attacks

**Merkle-Damgård problem:**

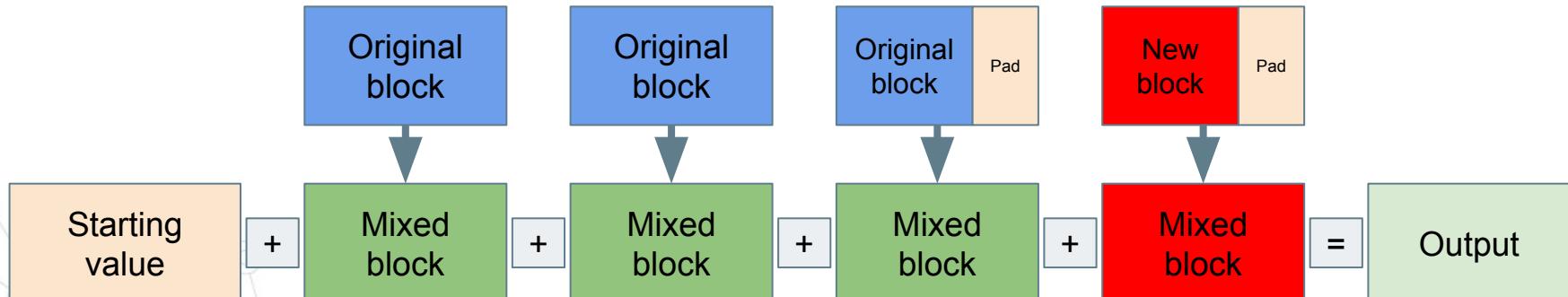
- ◎ You can just add blocks onto the end!



# Length Extension Attacks

## Merkle-Damgård problem:

- You can just add blocks onto the end!
- Given  $H(m)$  but not  $m$ , you can calculate  $H(m + \text{pad} + a)$



# Length Extension Attacks

**Example:** Flickr vulnerability in 2009:

**URL format:**

**http://flickr.com?auth=37ab078bb2286b440e48f892c56ae98  
b&action1=ListImages**

Auth format: MD5(secret + url)

Here: MD5('h34g09ha' + '&action1=ListImages')

# Length Extension Attacks

*Hash verifies the rest of the URL*

**Example:** Flickr vulnerability in 2009:

**URL format:**

`http://flickr.com?auth=37ab078bb2286b440e48f892e56ae98  
b&action1=ListImages&action2=DeleteAllImages`



Auth format: MD5(secret + url)

Here: MD5('h34g09ha' + '&action1=ListImages')

# Length Extension Attacks

**Problem:** Given

`MD5(secret + '&action1=ListImages')`

We can calculate:

`MD5(secret + '&action1=ListImages' + pad + '&action2=DeleteAllImages')`

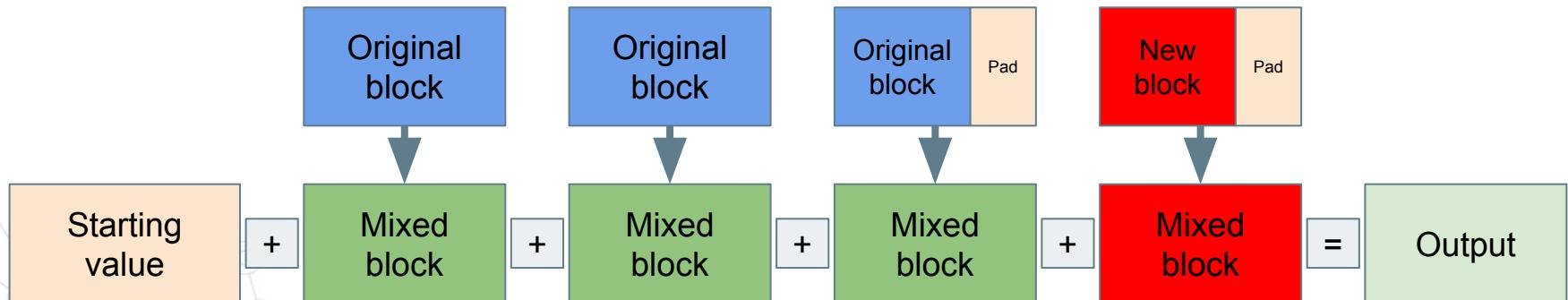
# Length Extension Attacks

**Problem:** Given

`MD5(secret + '&action1=ListImages')`

We can calculate:

`MD5(secret + '&action1=ListImages' + pad + '&action2=DeleteAllImages')`



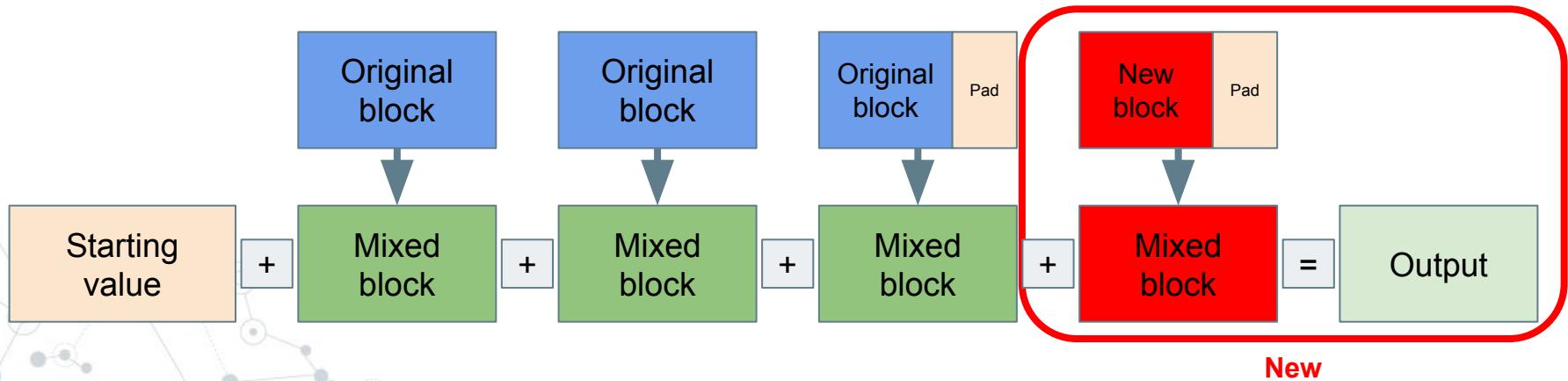
# Length Extension Attacks

**Problem:** Given

`MD5(secret + '&action1=ListImages')`

We can calculate:

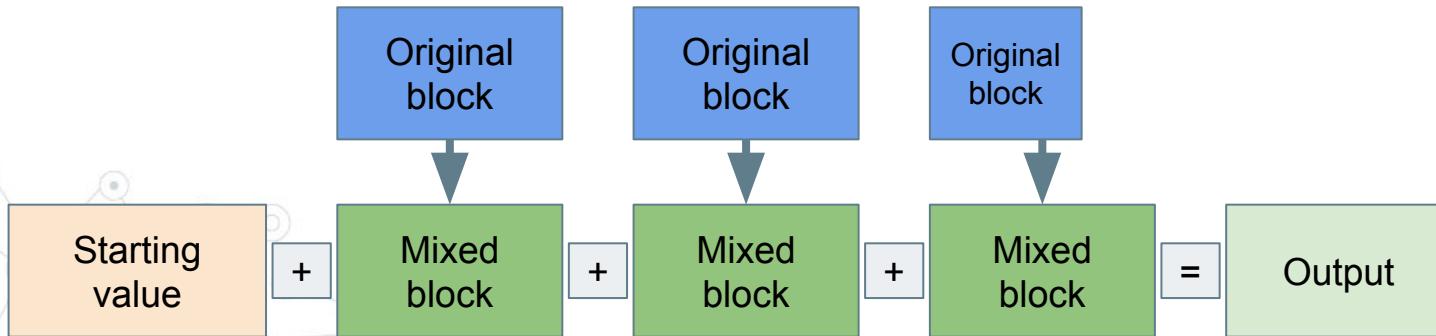
`MD5(secret + '&action1=ListImages' + pad + '&action2=DeleteAllImages')`



# Length Extension Attacks

Steps:

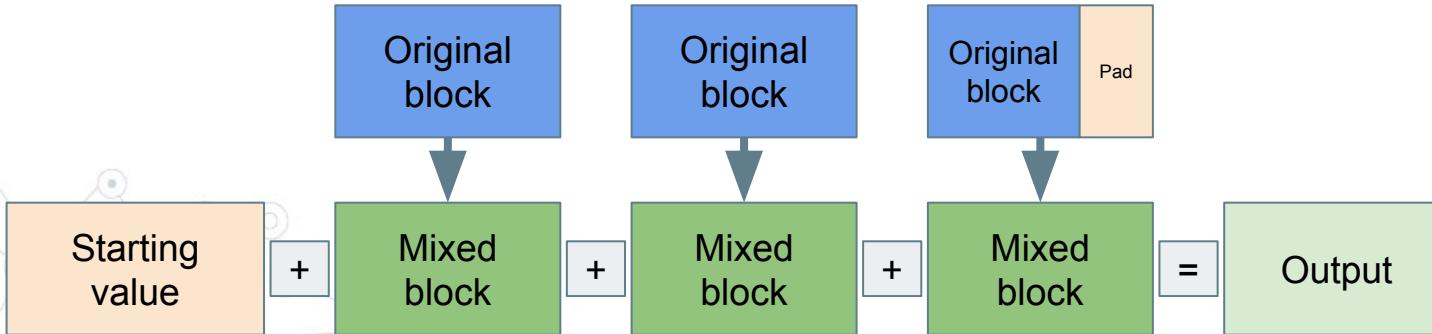
1. Take original hash (37ab078bb2286b440e48f892c56ae98b)



# Length Extension Attacks

Steps:

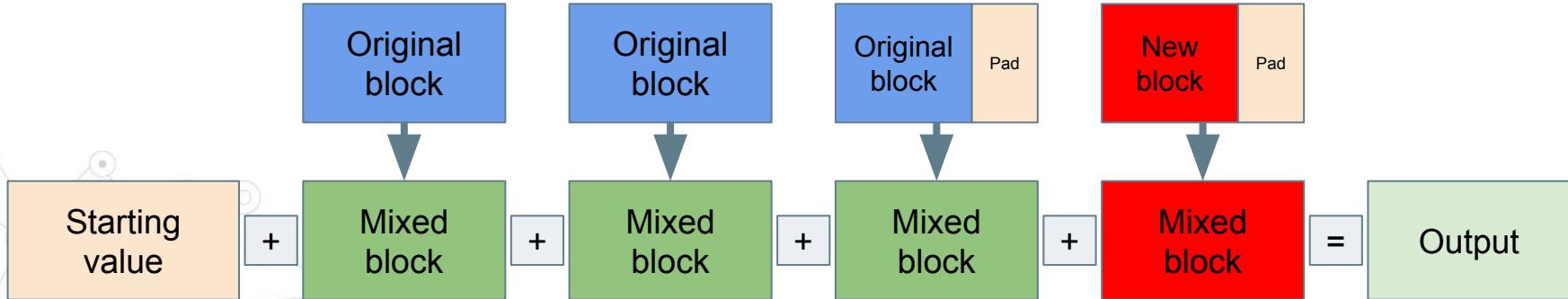
1. Take original hash (37ab078bb2286b440e48f892c56ae98b)
2. Calculate original padding based on estimated length
  - Can be guessed or brute-forced



# Length Extension Attacks

Steps:

1. Take original hash (37ab078bb2286b440e48f892c56ae98b)
2. Calculate original padding based on estimated length
  - Can be guessed or brute-forced
3. Extend the original hash with the padding and new text



# Length Extension Attacks

```
# Step 1: Take the original hash
original_hash = '37ab078bb2286b440e48f892c56ae98b'

# Step 2: Calculate padding (we can guess or brute-force this)
original_length = 124
padding = calculate_padding(original_length)

# Step 3: Extend the original hash
# Restore the state the previous MD5 hash was in when it was created
h = md5(state=original_hash, length=original_length + len(padding))
# Extend it
h.update('&action2>DeleteAllImages')
```

# Length Extension Attacks

**Result:**  $H(\text{original} + \text{pad} + \text{new})$

`http://flickr.com?auth=ca0184d49fb39705344378e5bcd33e89  
&action1=ListImages%80%00%00%00%00%00%00%00%00%00%00%  
0%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%  
%00%00%00%00%00%00%D0%00%00%00%00%00%00%00%00%00%00%&  
action2=DeleteAllImages`

# Length Extension Attacks

## Recap:

- ◎ **Length Extension Attack:** Generating  $H(m + a)$  for some string  $a$ , knowing only  $H(m)$  and not  $m$ .
- ◎ Affects hashes created with the block-based “Merkle–Damgård construction” such as MD5 or SHA1.
  - Note: Does not affect all MACs or HMACs! Most versions are secure!

# Data Security: Day 4

Digital Signatures  
Password storage

# Announcements

So, I underestimated this topic

- Questions may be removed from the end of this week's quiz
- Spread out the schedule to prevent this in the future

# Announcements

**T9Hacks #7:** February 18-19th

Registration link: [Linktr.ee/T9Hacks](https://linktr.ee/T9Hacks)



# Message hashing

Recap from Tuesday:

- ◎ **Hashes:** Fixed-length strings used to verify **integrity**
  - MD5, SHA-1, SHA-256

Alice

Bob

*Message: Transfer \$30 to Carol*

*MD5(message): 78adcbc2cf8d0c67b9eb75f3c93ded18*

# Message hashing

## Professional /

Stable

09 February 2022 at 15:01 UTC

Burp Suite Pro

[Download your copy](#)

SHA256: 7c8f76bf147e00000000000000000000  
MD5: 521e60a869327

The freeware version of IDA v7.7 comes with the following limitations:

- no commercial use is allowed
- cloud-based decompiler lacks certain advanced commands
- lacks support for many processors, file formats, etc...
- comes without technical support

SHA1 checksums:

```
d11fde87bc8550c2f2b91a913a6b3b549c1fd287 arm_idafree77_mac.app.zip
42038657317ebea44954b484a236e7f8cbc7d2fa idafree77_linux.run
195ced3615ea9975953271a68a4b58a8b754f67e idafree77_mac.app.zip
1f815be20a119cc835e7678a32032ab130834d49 idafree77_windows.exe
```

Home > Download

## Thank you for downloading MuseScore

Your download will start in 0 seconds. Problems with the download? Please use this [direct link](#)

SHA256 Checksum: fa3ca0f8cc5b0e8b0c0bb8ef11e227b9b27b2a5c9da28dab58bafccb0eb657d0



IDA Freeware for Linux (76MB)



IDA Freeware for Mac (68MB)



IDA Freeware for Mac ARM (90MB)

[Download Blender 3.0](#)

Windows · Installer · 203MB · ⓘ

Compatible with Windows 11, 10, and 8.1

Blender 3.0 was released on December 3, 2021

md5 sha256

# Message hashing

JavaScript is often imported from third parties

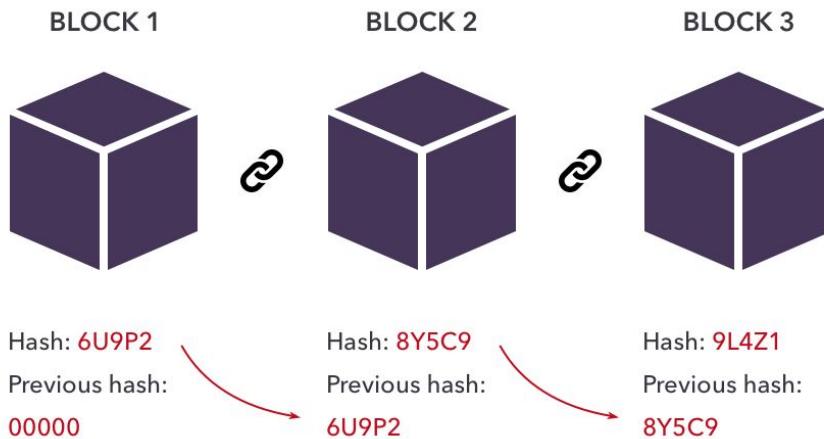
- Import will fail if hashes do not match
- Hash is hard-coded in advance, so it cannot be spoofed

```
<script  
    src="https://code.jquery.com/jquery-1.12.3.min.js"  
    integrity="sha256-aa0DHAgwQW1bFOGXMeX+pC4PZIPsvn2h1sArY0hgXQ="  
    crossorigin="anonymous"></script>
```

# Message hashing

Blockchain uses hashes to verify data

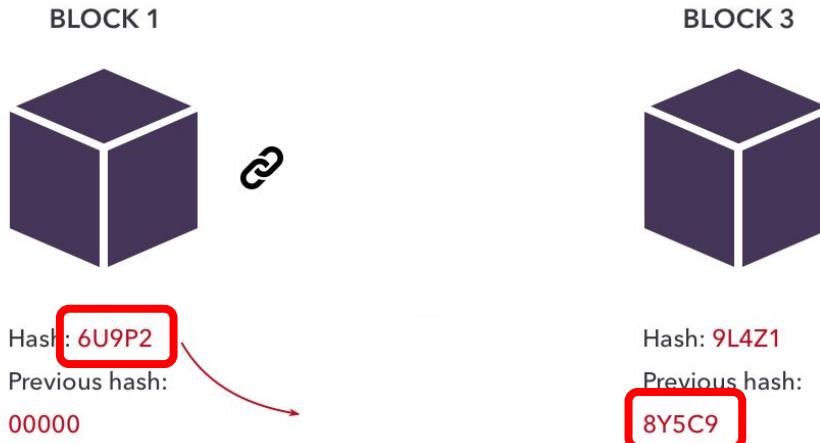
- Each block stores the SHA-256 hash of the previous block
- Blocks cannot be changed without breaking the “chain”



# Message hashing

Blockchain uses hashes to verify data

- Each block stores the SHA-256 hash of the previous block
- Blocks cannot be changed without breaking the “chain”



# Message hashing

**Hash collision:** Two messages with the same hash

*[Collision demo]*

# Message hashing

## Flame attack:

- ◎ Used an MD5 hash collision to send a fake Windows update, which appeared to be the real one

BIZ & IT —

### Flame malware wielded rare “collision” crypto attack against Microsoft

Such real-world exploits are almost unheard of, underscoring Flame's ingenuity.

DAN GOODIN - 6/5/2012, 1:31 AM



<https://arstechnica.com/information-technology/2012/06/flame-wields-rare-collision-crypto-attack/>

# Message hashing



Evolution of MD5 collision research:

- 2012: NSA deploys Flame malware in “unheard of” exploit
- 2014: Can be done in 10 hours on the cloud

BIZ & IT —

## Crypto attack that hijacked Windows Update goes mainstream in Amazon Cloud

Collision attack against widely used MD5 algorithm took 10 hours, cost just 65 cents.

DAN GOODIN - 11/5/2014, 7:00 AM

<https://arstechnica.com/information-technology/2014/11/crypto-attack-that-hijacked-windows-update-goes-mainstream-in-amazon-cloud/>

- 2022: Can be done in fifteen seconds on a laptop



# MACs and HMACs

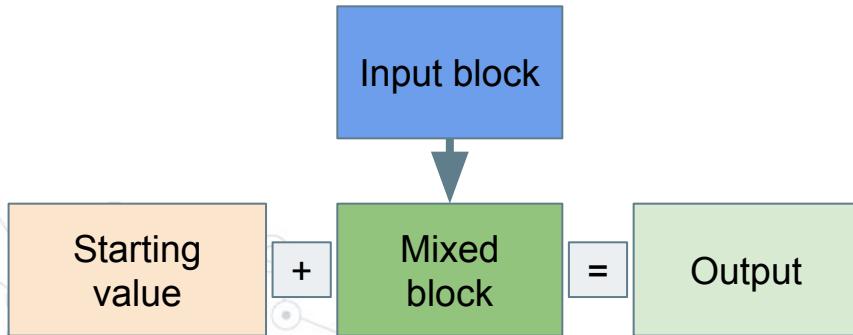
Recap from Tuesday:

- **MACs:** Integrate a secret key to prevent spoofing
- **HMACs:** Hash functions that use a secret key
  - e.g. MD5(key + message)
  - Secure *if done correctly* (the above example is common but it **is not** done correctly)

# Length Extension Attacks

## Recap on hash construction:

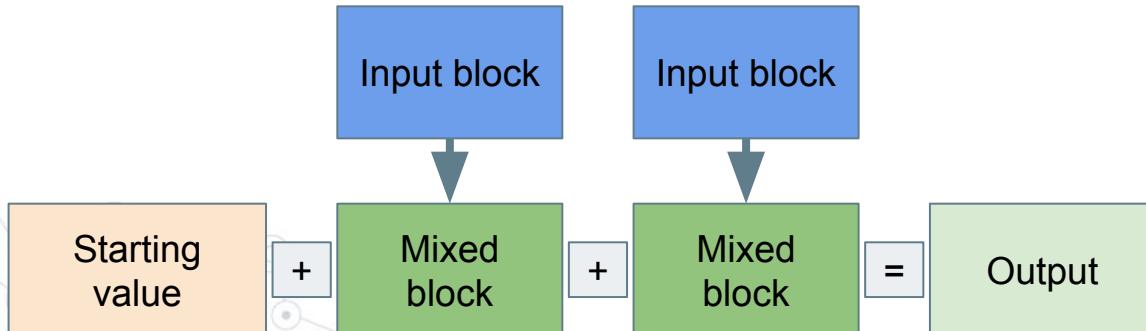
- ◎ Hashes like MD5 or SHA-1 are computed in 512-bit blocks
  - Each block modifies the result of the previous one



# Length Extension Attacks

## Recap on hash construction:

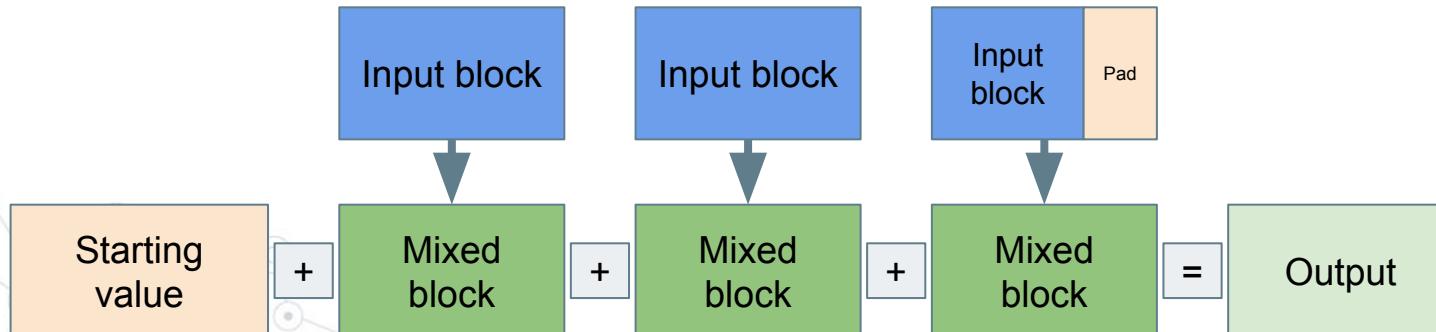
- ◎ Hashes like MD5 or SHA-1 are computed in 512-bit blocks
  - Each block modifies the result of the previous one



# Length Extension Attacks

## Recap on hash construction:

- ◎ Hashes like MD5 or SHA-1 are computed in 512-bit blocks
  - Each block modifies the result of the previous one
  - Last block is padded to be 512 bits



# Length Extension Attacks

**Hash length extension:**

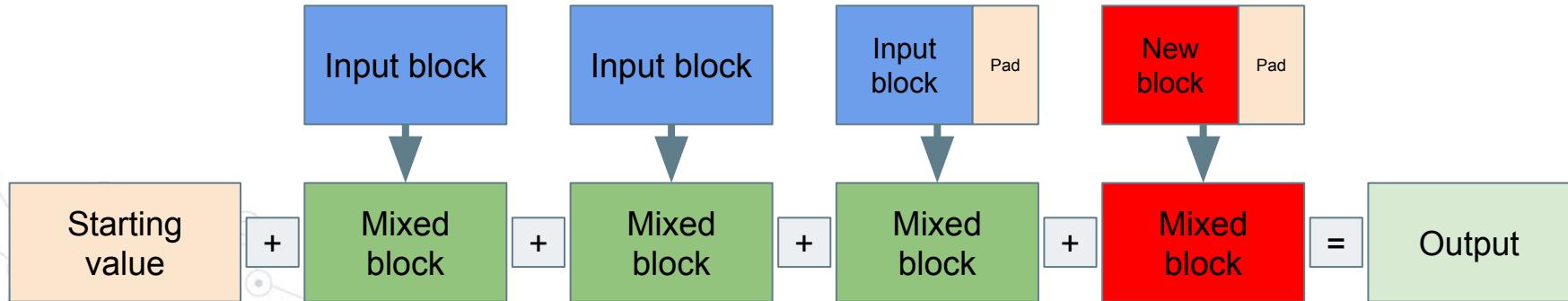
- Attackers can keep adding on new blocks

*Given only:*

MD5( $x$ )

*We can calculate:*

MD5( $x + \text{padding} + \text{suffix}$ )





# Demo



# MACs and HMACs

## Solution:

Better HMAC constructions:

- **MD5(message + key)**
  - Not preferred, but resistant to length extension
- **MD5(key + MD5(key + message))**
  - Use this one

# MACs and HMACs

*Length extension is considered part of the key, not the message*

## Solution:

Better HMAC constructions:

- **MD5(message + key + padding + suffix)**
  - Not preferred, but resistant to length extension
- **MD5(key + MD5(key + message))**
  - Use this one

# MACs and HMACs

*Length extension is considered part of the key, not the message*

## Solution:

Better HMAC constructions:

- **MD5(message + key + padding + suffix)**
  - Not preferred, but resistant to length extension
- **MD5(key + MD5(key + message) + padding + suffix)**
  - Use this one

*Length extension is considered part of the hash*

# Length Extension Attacks

**Q:** Length extension attacks seem weirdly specific. Is it really a big problem?

# Length Extension Attacks

**Q:** Length extension attacks seem weirdly specific. Is it really a big problem?

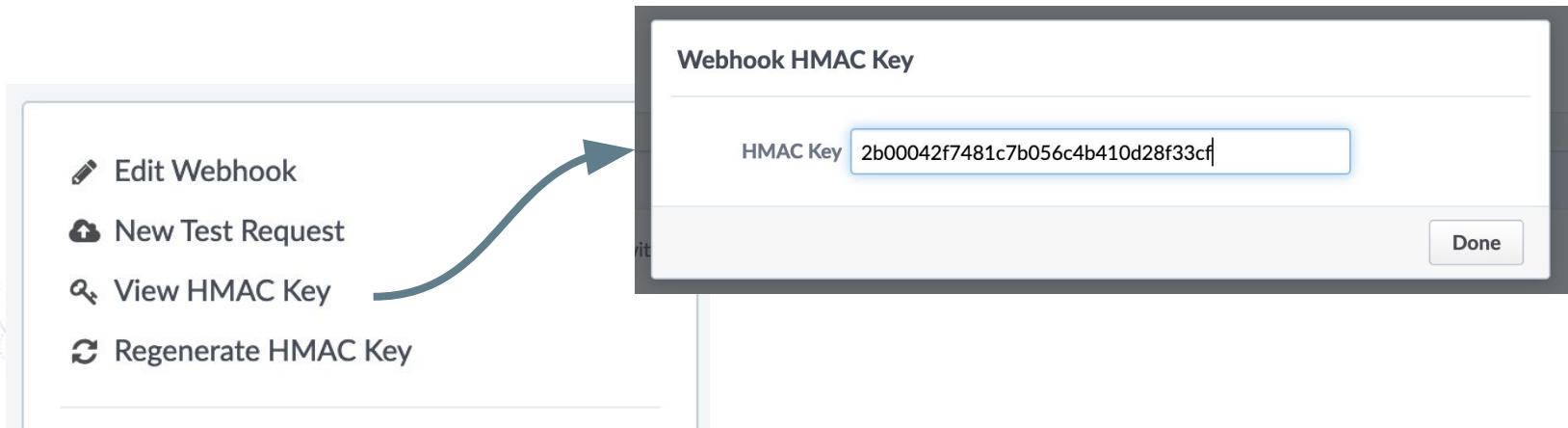
**A:** Not a massive problem, but:

- ◎ It does happen, as  $\text{MD5}(\text{secret} + \text{message})$  is simple and many developers do not know better.
- ◎ It demonstrates a larger problem, which is “rolling your own crypto”.

# Length Extension Attacks

## Recap:

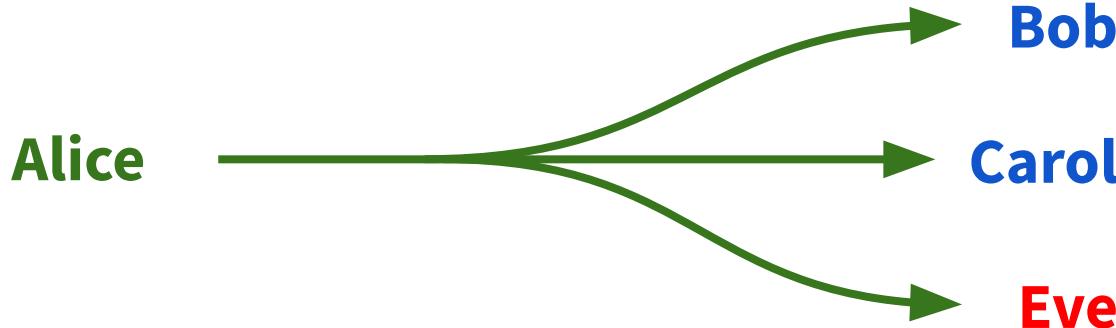
- MACs incorporate a secret key, so they can only be created and verified by people with that key



# Length Extension Attacks

## Problem:

- ◎ ***Everyone*** with the HMAC key can create valid HMACs
  - This is a big problem for public messages, e.g. software downloads

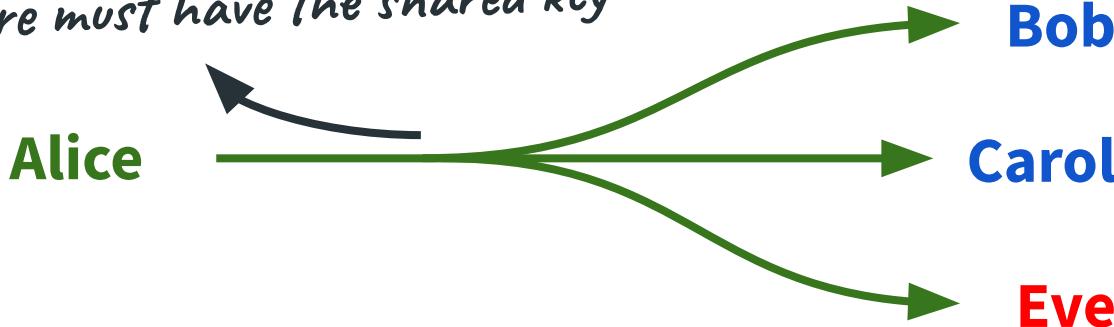


# Length Extension Attacks

## Problem:

- ◎ **Everyone** with the HMAC key can create valid MACs
  - This is a big problem for public messages, e.g. software downloads

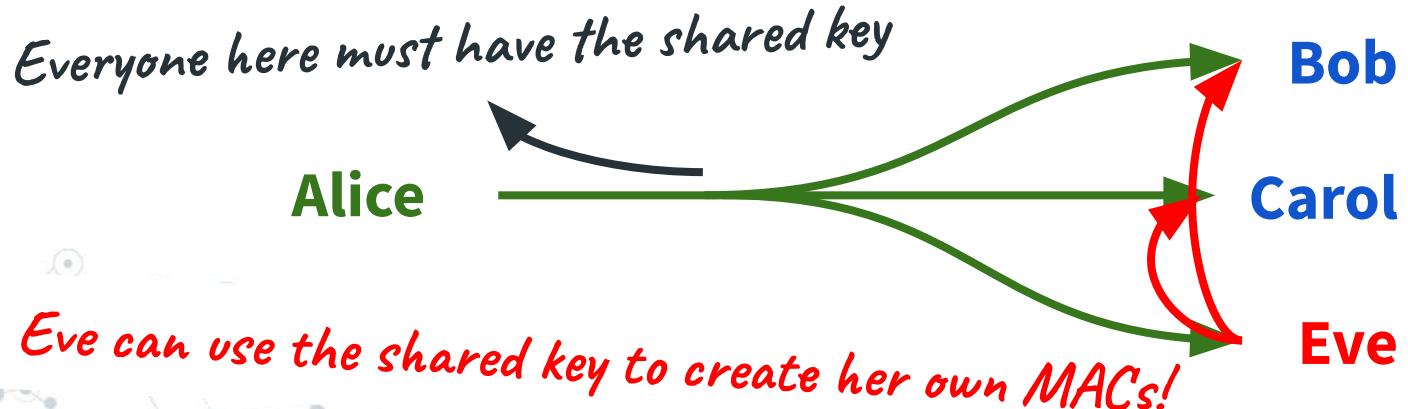
*Everyone here must have the shared key*



# Length Extension Attacks

## Problem:

- ◎ **Everyone** with the HMAC key can create valid MACs
  - This is a big problem for public messages, e.g. software downloads



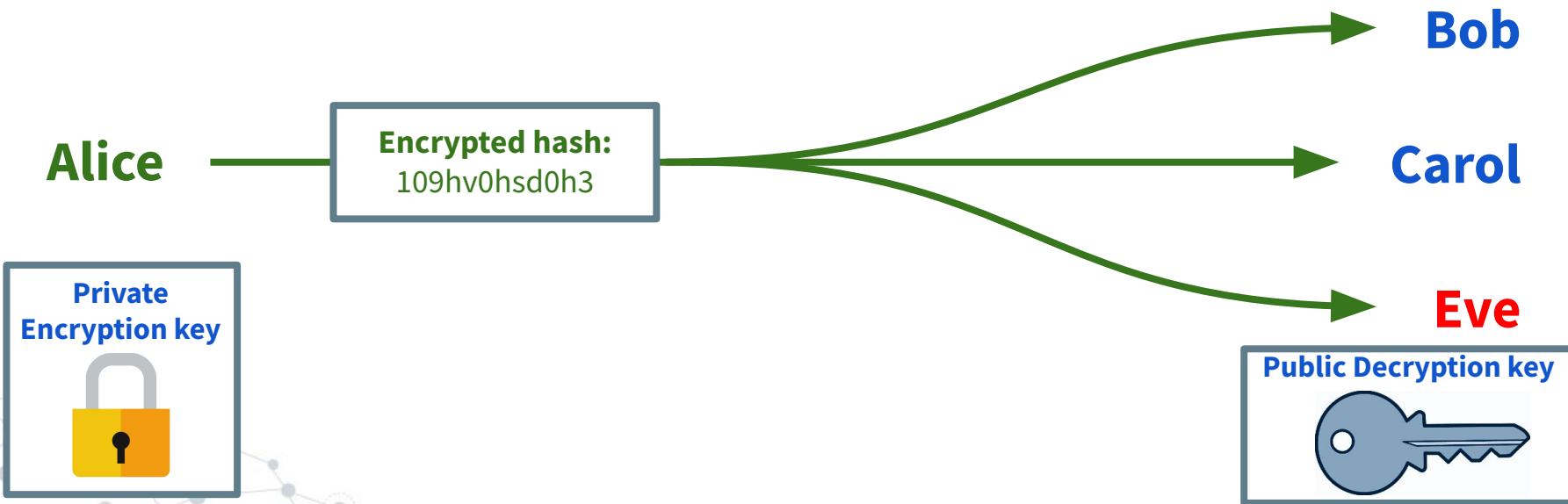
# Digital Signatures

**Digital Signature:** A message hash that uses public / private key pairs to verify the identity of the sender

1. Generate message hash
2. Encrypt hash with private key
3. Only the person with the private key can give out the correct corresponding public key

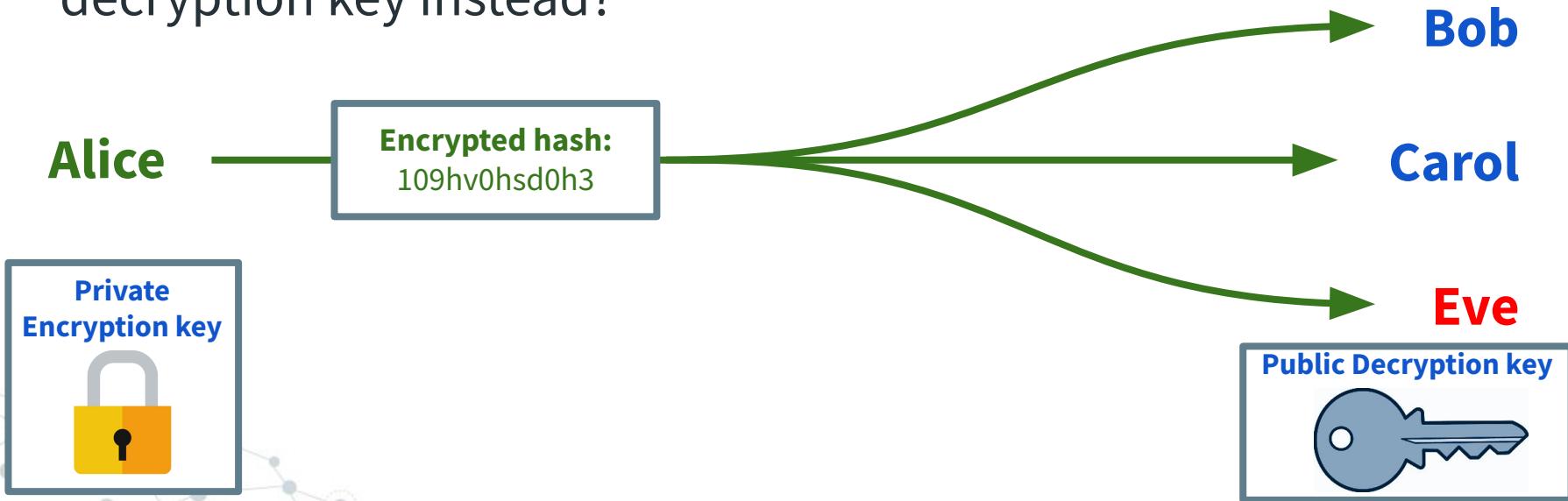
# Digital Signatures

- Only Alice can create signatures with the private key
- Anyone with the public key can decrypt and verify them



# Digital Signatures

Q: How do Bob and Carol get the correct public decryption key? What if **Eve** tricks them into using her public decryption key instead?



# Digital Signatures

**One option:**

Literally meeting face-to-face and  
exchanging public keys (key signing parties)

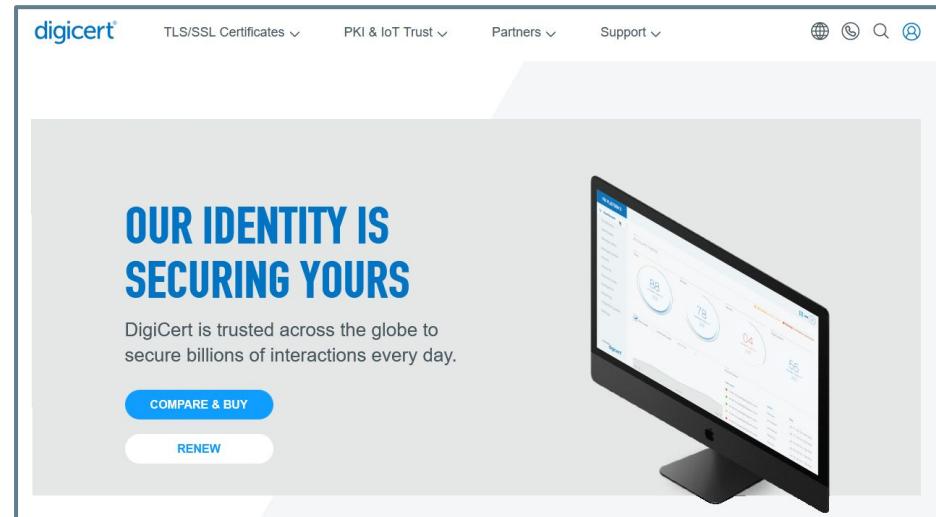


[https://en.wikipedia.org/wiki/Key\\_signing\\_party](https://en.wikipedia.org/wiki/Key_signing_party)

# Digital Signatures

## Another option:

Build an entire industry around verifying the identity of public/private key owners



<https://www.digicert.com/>

# Digital Signatures



<https://www.davidegrayson.com/signing/>

# Digital Signatures

NEWS

## DigiNotar dies from certificate hack caper

'Unlikely many are going to shed tears' over Dutch company's demise, says security researcher



By Gregg Keizer

Senior Reporter, Computerworld | SEP 21, 2011 4:09 PM PST

The Dutch company that was hacked earlier this summer by certificate thieves has gone bust and shut down, its U.S.-based owner said Tuesday.

DigiNotar filed for bankruptcy in a Netherland court on Monday, and its assets will be liquidated by a court-appointed trustee, said Vasco Data Security International, the Chicago company that purchased DigiNotar last January for \$13.1 million.

<https://www.computerworld.com/article/2511297/diginotar-dies-from-certificate-hack-caper.html>

# Message integrity

## Recap

- **Hashes:** Prove message integrity
- **MACs:** Prove message integrity, but can only be created or read by parties with a shared secret
- **Digital signatures:** Prove message integrity, but can only be created by one party and verified by anyone with their public key

# Passwords

Imagine you are building a website with user accounts.  
Each user enters their username and password.

**Q: Should you encrypt those passwords?**

# Passwords



**A: No!**

*Why not?*



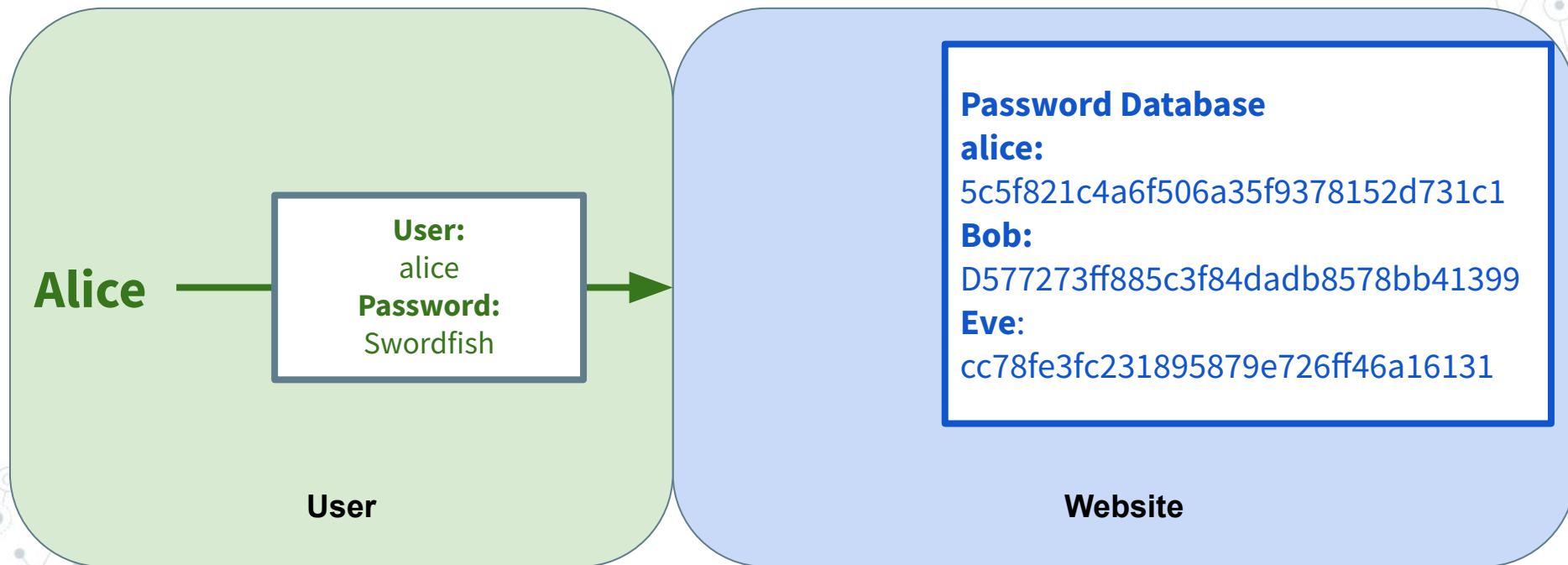
# Passwords

Encrypted passwords can be decrypted. There is *never* a reason to do this!

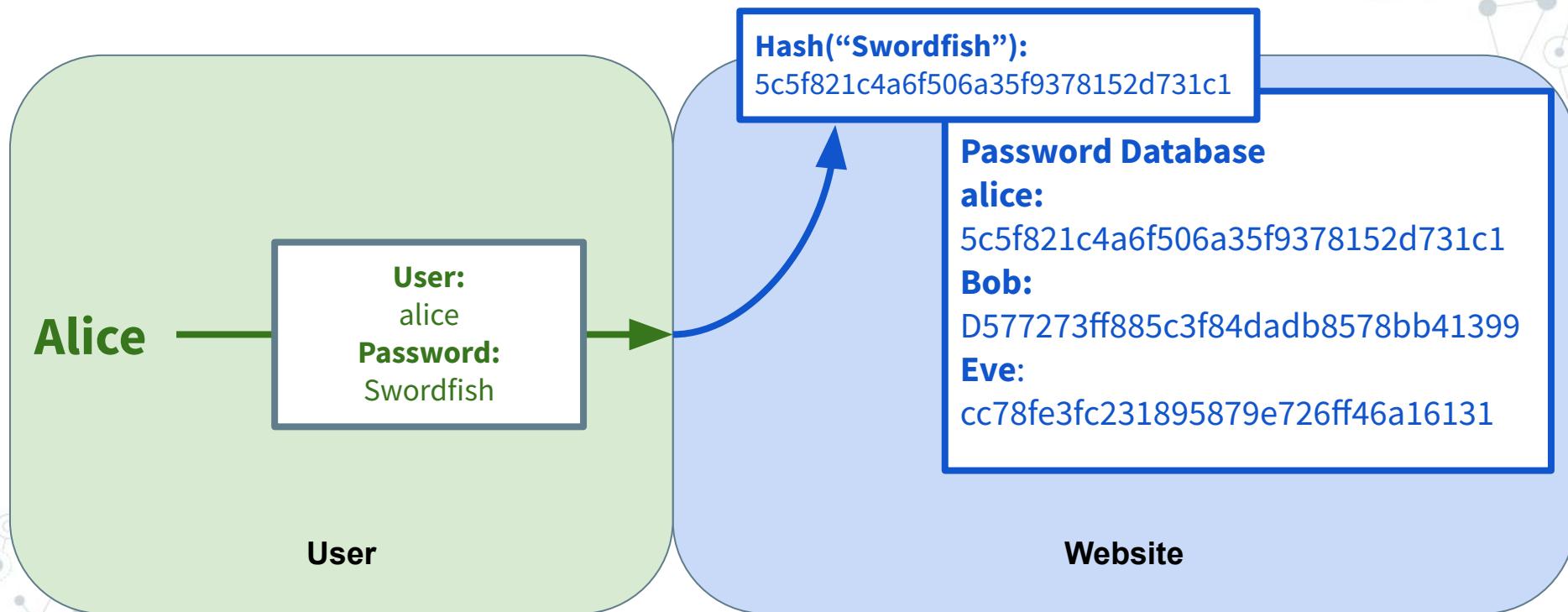
Better solution:

- **Hash** the password when you get it the first time
- *Throw away the original password* and only store the hash
- Hash future password attempts, and compare the hashes

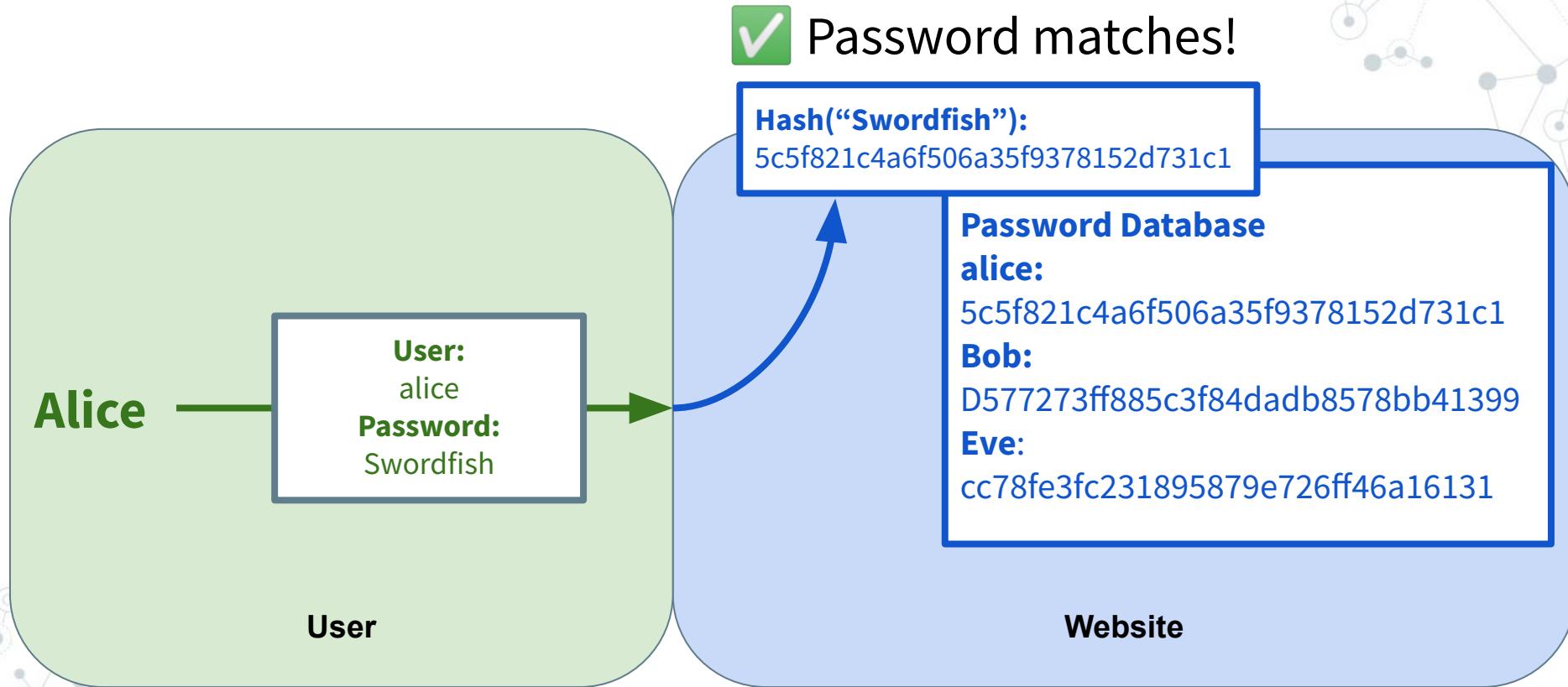
# Passwords



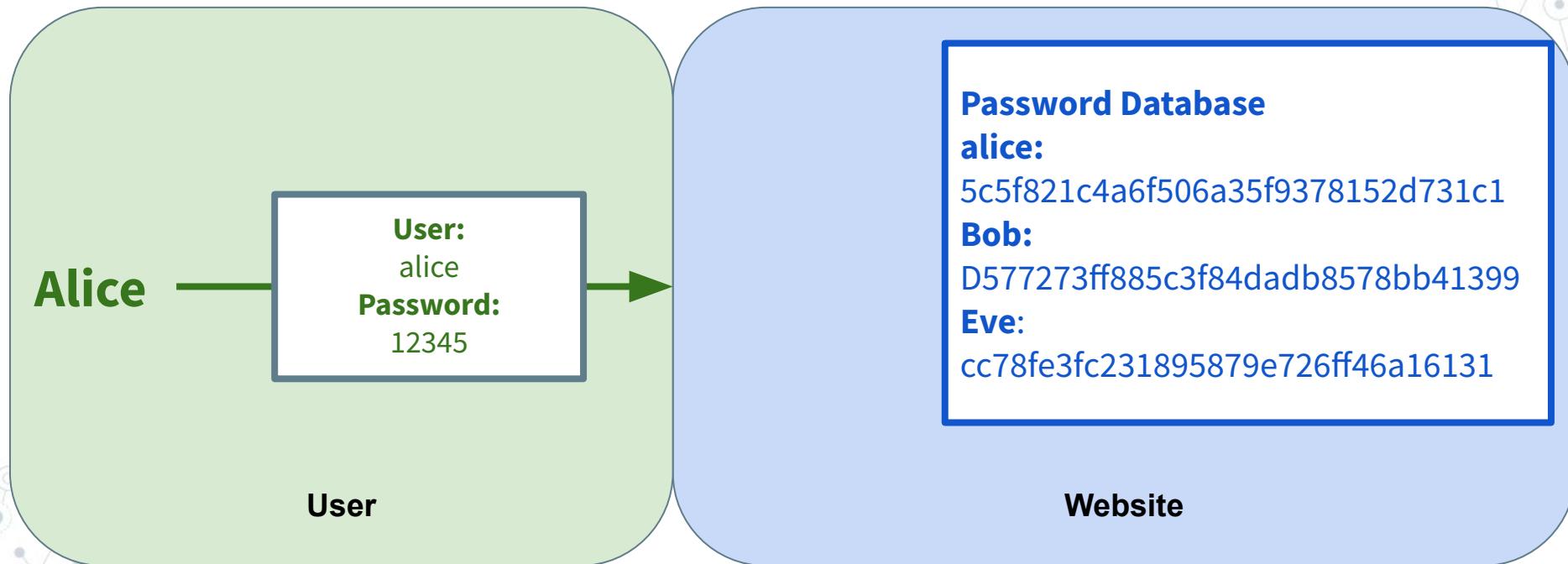
# Passwords



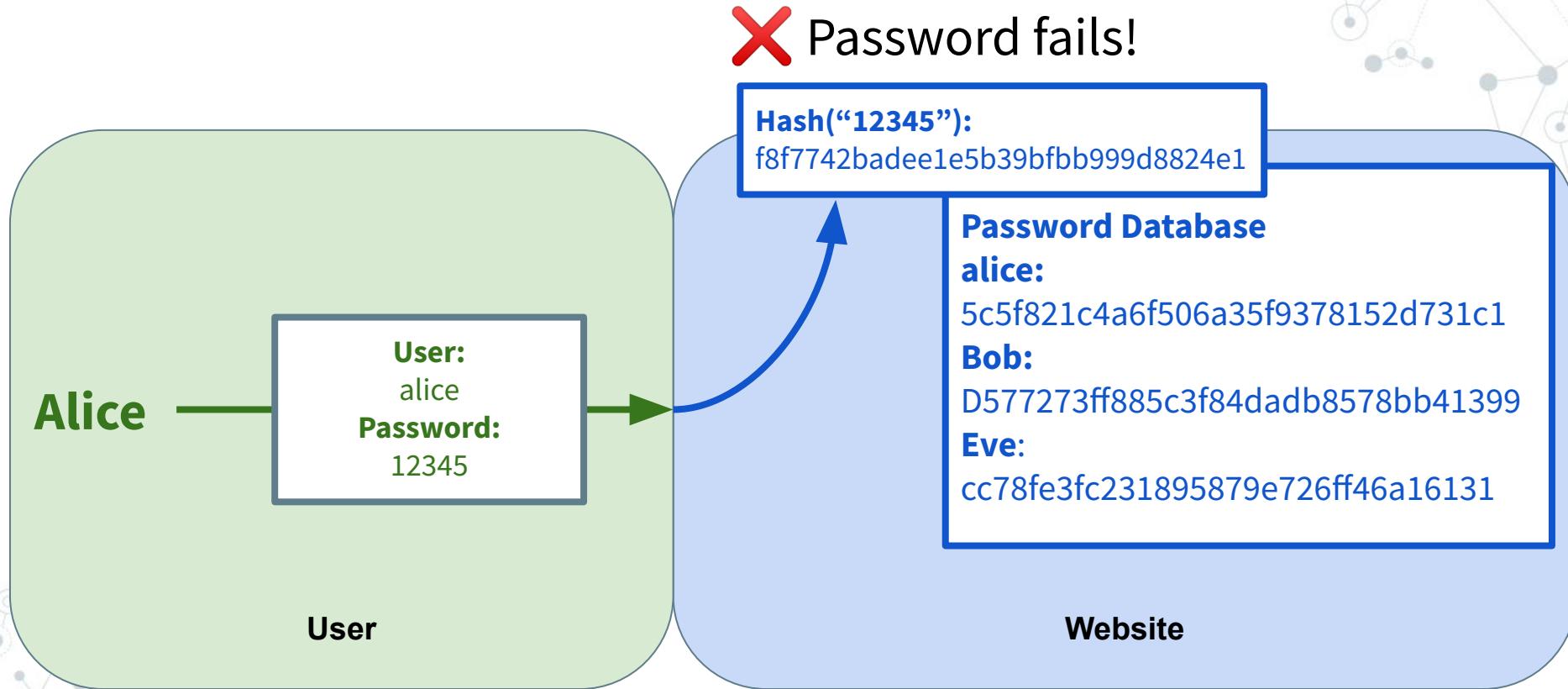
# Passwords



# Passwords



# Passwords



# Passwords

**Benefit:** An attacker with access to the database cannot get the original password!

- ◎ e.g. hacker with root access, or malicious employee

## Password Database

**alice:**

5c5f821c4a6f506a35f9378152d731c1

**Bob:**

D577273ff885c3f84dadb8578bb41399

**Eve:**

cc78fe3fc231895879e726ff46a16131

# Passwords

**Benefit:** An attacker with access to the database cannot get the original password!

- e.g. hacker with root access, or malicious employee
- Companies should **never** be able to email you your password!

## Password Database

**alice:**

5c5f821c4a6f506a35f9378152d731c1

**Bob:**

D577273ff885c3f84dad8578bb41399

**Eve:**

cc78fe3fc231895879e726ff46a16131

# Passwords



See: <https://haveibeenpwned.com/PwnedWebsites>



## Zomato

In May 2017, the restaurant guide website Zomato was hacked resulting in the exposure of almost 17 million accounts. The data was consequently redistributed online and contains email addresses, usernames and salted MD5 hashes of passwords (the password hash was not present on all accounts). This data was provided to HIBP



## Zynga

In September 2019, game developer Zynga (the creator of Words with Friends) suffered a data breach. The incident exposed 173M unique email addresses alongside usernames and passwords stored as salted SHA-1 hashes. The data was provided to HIBP by dehashed.com.



## Zoomcar

In July 2018, the Indian self-drive car rental company Zoomcar suffered a data breach which was subsequently sold on a dark web marketplace in 2020. The breach exposed over 3.5M records including names, email and IP addresses, phone numbers and passwords stored as bcrypt hashes. The data was provided to HIBP by dehashed.com.

# Passwords

**Concern:** Password cracking

- Attackers can still iterate through common passwords

...

```
"Password": 29f33cab54c2a8858885b95d8fbb7ff1  
"12345": d577273ff885c3f84dad8578bb41399  
"letmein": 4aacf9c858c82716ab0034320bd2efe9  
"Swordfish": 5c5f821c4a6f506a35f9378152d731c1
```

**Password Database**

alice:

```
5c5f821c4a6f506a35f9378152d731c1
```

# Passwords

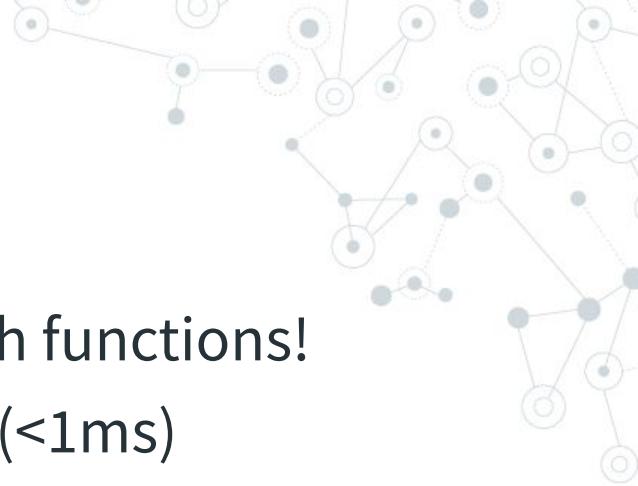
**User-based mitigation:** Use long, random passwords

- ◎ Total keyspace is  $p^n$ , where **p** is the number of possible characters and **n** is the length

# TIME IT TAKES A HACKER TO BRUTE FORCE YOUR PASSWORD

Number of Characters	Numbers Only	Lowercase Letters	Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters, Symbols
4	Instantly	Instantly	Instantly	Instantly	Instantly
5	Instantly	Instantly	Instantly	Instantly	Instantly
6	Instantly	Instantly	Instantly	1 sec	5 secs
7	Instantly	Instantly	25 secs	1 min	6 mins
8	Instantly	5 secs	22 mins	1 hour	8 hours
9	Instantly	2 mins	19 hours	3 days	3 weeks
10	Instantly	58 mins	1 month	7 months	5 years
11	2 secs	1 day	5 years	41 years	400 years
12	25 secs	3 weeks	300 years	2k years	34k years
13	4 mins	1 year	16k years	100k years	2m years
14	41 mins	51 years	800k years	9m years	200m years
15	6 hours	1k years	43m years	600m years	15 bn years
16	2 days	34k years	2bn years	37bn years	1tn years
17	4 weeks	800k years	100bn years	2tn years	93tn years
18	9 months	23m years	6tn years	100 tn years	7qd years

# Passwords



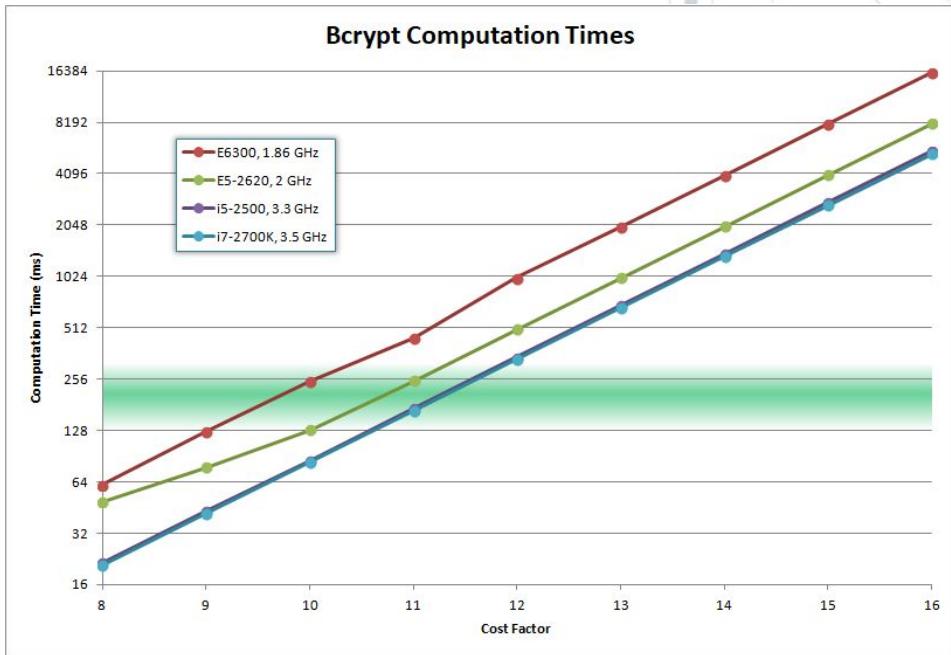
**Website-based mitigation:** Use slow hash functions!

- ◎ Normal hash functions: Goal is speed (<1ms)
  - e.g. MD5, SHA-1, SHA-256
- ◎ Password hash functions: Goal is slowness (>250ms)
  - PBKDF1, bcrypt



# Passwords

Some functions (e.g. bcrypt) even have a “cost factor” to raise computing time by repeating operations!



<https://stackoverflow.com/a/61304956>

# Passwords

See: <https://haveibeenpwned.com/PwnedWebsites>



## Zomato

In May 2017, the restaurant guide website Zomato was hacked resulting in the exposure of almost 17 million accounts. The data was consequently redistributed online and contains email addresses, usernames and salted MD5 hashes of passwords (the password hash was not present on all accounts). This data was provided to HIBP by [dehashed.com](#).



## Zynga

In September 2019, game developer Zynga (the creator of Words with Friends) suffered a data breach. The incident exposed 173M unique email addresses alongside usernames and passwords stored as salted SHA-1 hashes. The data was provided to HIBP by [dehashed.com](#).



## Zoomcar

In July 2018, the Indian self-drive car rental company Zoomcar suffered a data breach which was subsequently sold on a dark web marketplace in 2020. The breach exposed over 3.5M records including names, email and IP addresses, phone numbers and passwords stored as bcrypt hashes. The data was provided to HIBP by [dehashed.com](#).

# Passwords

See: <https://haveibeenpwned.com/>



 Zomato

In May 2017, the restaurant review website suffered a data breach which exposed the email addresses, usernames and hashed passwords of over 17 million users. The data was stored as MD5 hashes of password.

In May 2017, the restaurant review website suffered a data breach which exposed the email addresses, usernames and hashed passwords of over 17 million users. The data was stored as MD5 hashes of password.



Zynga

In September 2019, game developer Zynga (the creator of Words with Friends) suffered a data breach. The incident exposed 173M unique email addresses alongside usernames and passwords stored as salted SHA-1 hashes. The data was provided to HIBP by [dehashed.com](#).

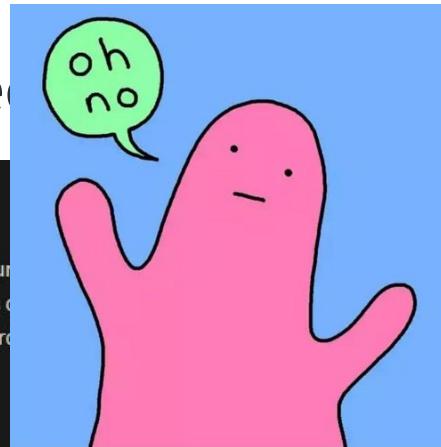


Zoomcar

In July 2018, the Indian self-drive car rental company Zoomcar suffered a data breach which was subsequently sold on a dark web marketplace in 2020. The breach exposed over 3.5M records including names, email and IP addresses, phone numbers and passwords stored as bcrypt hashes. The data was provided to HIBP by [dehashed.com](#).

# Passwords

See: <https://haveibeenpwned.com/>



 Zomato

In May 2017, the restaurant review website suffered a data breach which exposed over 12 million accounts. The data was stored as MD5 hashes of passwords.

In the exposure of almost 17 million accounts (including email addresses, usernames and salted MD5 hashes of passwords). This data was provided to HIBP by dehashed.com.



Zynga

In September 2019, game developer Zynga (the creator of Words with Friends) suffered a data breach. The incident exposed 173M unique email addresses alongside usernames and passwords stored as salted SHA-1 hashes. The data was provided to HIBP by dehashed.com.



Zoomcar

In July 2018, the Indian self-drive car rental company Zoomcar suffered a data breach which was subsequently sold on a dark web marketplace in 2020. The breach exposed over 3.5M records including names, email and IP addresses, phone numbers and passwords stored as bcrypt hashes. The data was provided to HIBP by dehashed.com.

# Recap

- ◎ Passwords can still be brute-forced
  - Long passwords can slow this down
  - Slow hash algorithms are preferred for passwords



# Passwords

**Problem:** Most users do not use random passwords

## Top 100

- |              |              |
|--------------|--------------|
| 1. 123456    | 11. 123123   |
| 2. password  | 12. baseball |
| 3. 12345678  | 13. abc123   |
| 4. qwerty    | 14. football |
| 5. 123456789 | 15. monkey   |
| 6. 12345     | 16. letmein  |
| 7. 1234      | 17. 696969   |
| 8. 111111    | 18. shadow   |
| 9. 1234567   | 19. master   |
| 10. dragon   | 20. 666666   |

[https://en.wikipedia.org/wiki/Wikipedia:10,000\\_most\\_common\\_passwords](https://en.wikipedia.org/wiki/Wikipedia:10,000_most_common_passwords)

# Passwords

**Problem:** Most users do not use random passwords

- Users with identical passwords have identical hashes

*Alice and Bob probably have a very common password!*



## Password Database

**alice:**

5c5f821c4a6f506a35f9378152d731c1

**Bob:**

5c5f821c4a6f506a35f9378152d731c1

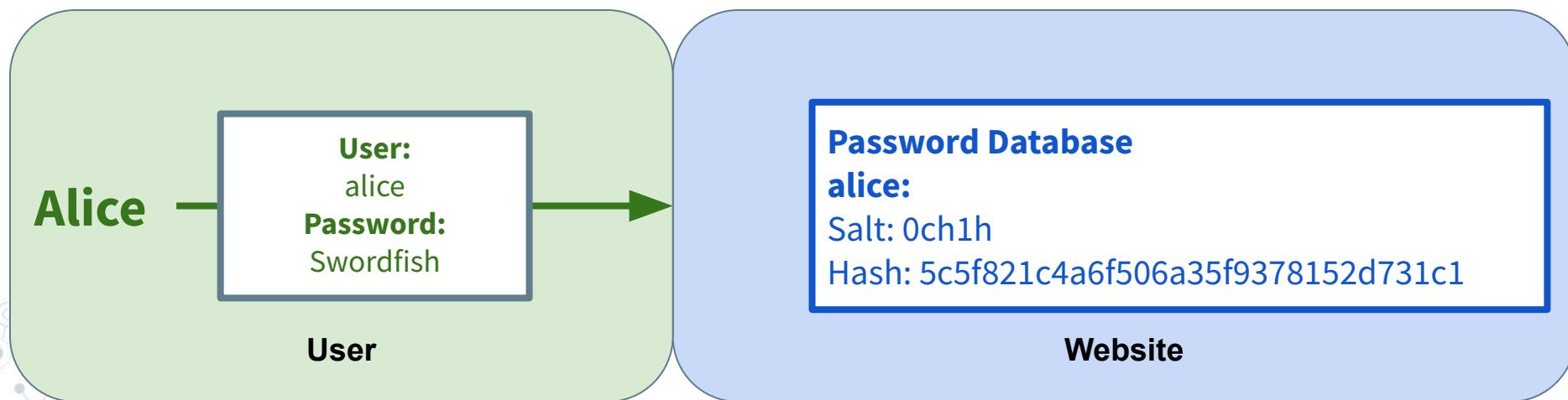
**Eve:**

cc78fe3fc231895879e726ff46a16131

# Passwords

## Solution: Salts

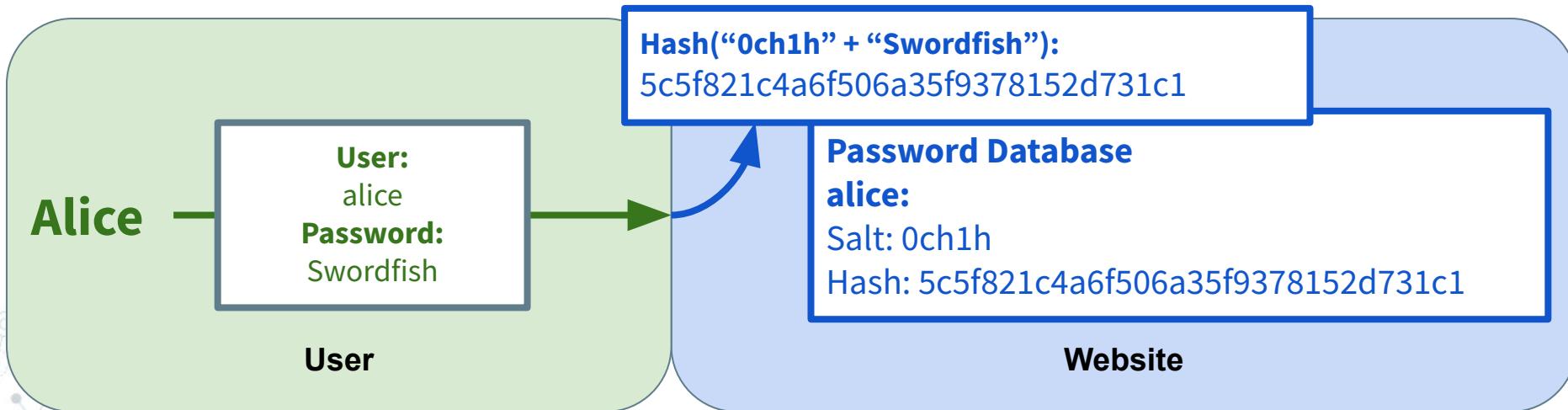
- Pick a random (non-secret) value, a “salt”, for each user
- Add the salt to the password before hashing it



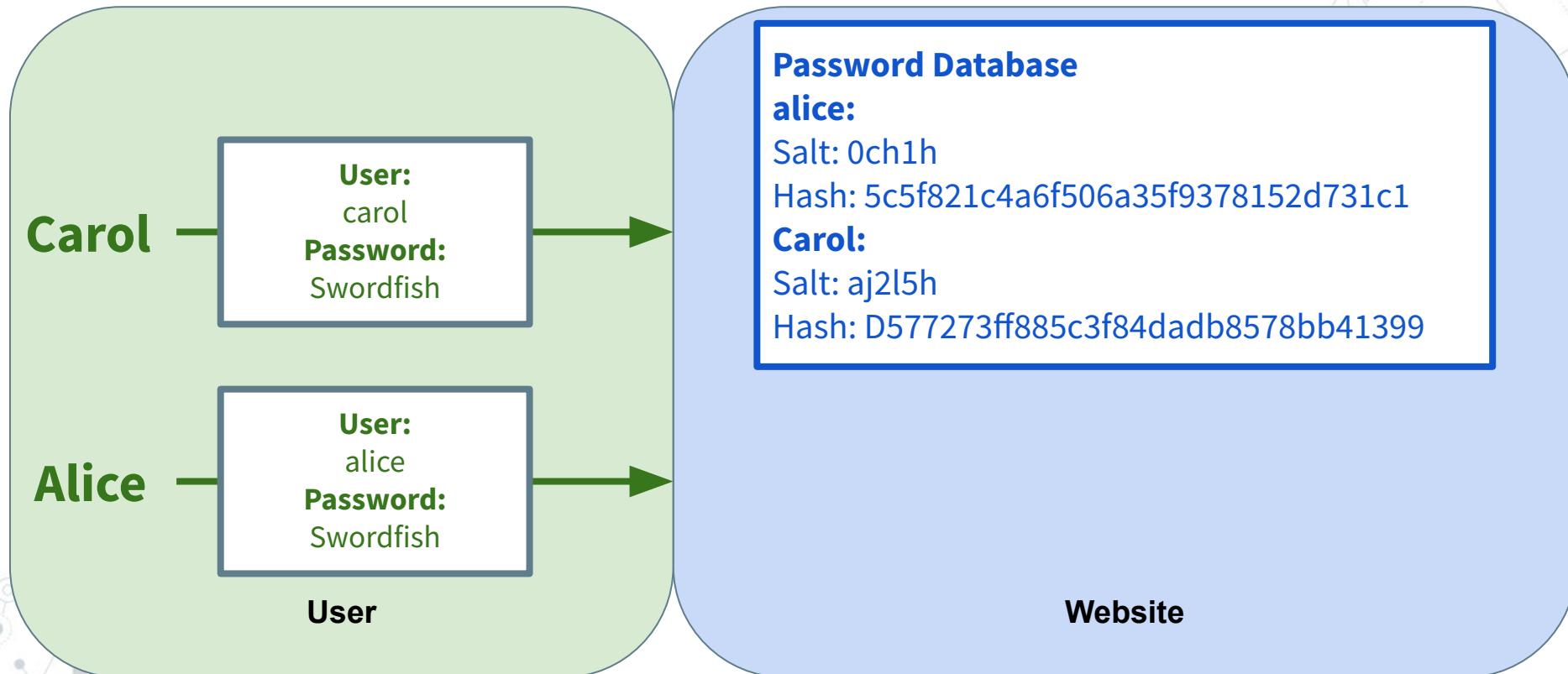
# Passwords

## Solution: Salts

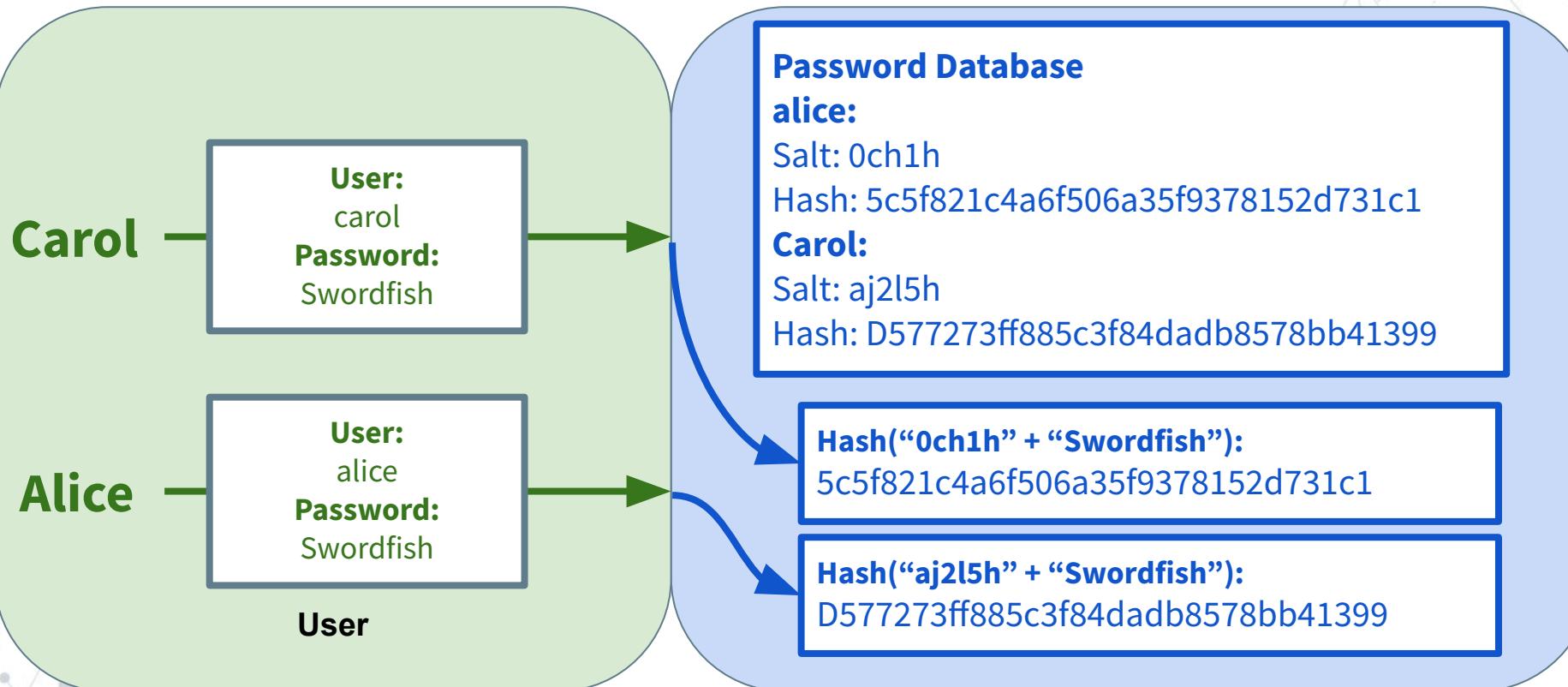
- Pick a random (non-secret) value, a “salt”, for each user
- Add the salt to the password before hashing it



# Passwords



# Passwords



# Passwords

## Salts

- ◎ Randomizes user hashes, even if the password is the same
  - **Does not** slow down the cracking of a single password
  - **Does** slow down the cracking of multiple passwords, as each one must be down individually

### Password Database

**alice:**

Salt: 0ch1h | Hash: 5c5f821c4a6f506a35f9378152d731c1

**Carol:**

Salt: aj2l5h | Hash: D577273ff885c3f84dadb8578bb41399

# Recap

- ◎ **Password hashing:** Prevents attackers from reading passwords, even if they get access to the database
  - Goal is to be slow, not fast
  - Common algorithms: PBKDF1, bcrypt
- ◎ **Password salts:** Random, non-secret values added to each password so each hash is different
  - Often stored alongside the hash
  - Only slows down the cracking of *multiple* passwords



# Questions?



Claudia Pellegrino @c\_pellegrino · Apr 4

Does T-Mobile Austria in fact store customers' passwords in clear text  
@tmobileat? @PWTooStrong @Telekom\_hilft

SeloX @SeloX\_AUT

Replies to @c\_pellegrino @PWTooStrong @Telekom\_hilft

Had the same issue with T-Mobile Austria. Apparently they are saving the password in clear because employees have access to them (you have to tell them your password when you're talking to them on the phone or in a shop) and they are not case sensitive



36



237



468



T-Mobile Austria @tmobileat · Apr 4

Hello Claudia! The customer service agents see the first four characters of your password. We store the whole password, because you need it for the login for [mein.t-mobile.at](#) ^andrea



123



147



117





T-Mobile Austria @tmobileat · Apr 5

Hi @c\_pellegrino, I really do not get why this is a problem. You have so many passwords for every app, for every mail-account and so on. We secure all data very carefully, so there is not a thing to fear. ^Käthe



114



111



97



Eric™ @Korni22 · 14h

Well, what if your infrastructure gets breached and everyone's password is published in plaintext to the whole wide world?



1



23



472



T-Mobile Austria @tmobileat · 14h

@Korni22 What if this doesn't happen because our security is amazingly good?  
^Käthe



158



161



174



Eric™ @Korni22 · 14h

Bad news for you Käthe, nobody's security is that good. No, not even yours. It's not that I say you are 100% getting hacked - what if an employee accesses the database directly?



3



24



733



T-Mobile Austria @tmobileat · 14h

@Korni22 Excuse me? Do you have any idea how telecommunication companies work? Do you know anything about our systems? But I'm glad you have the time to share your view with us. ^Käthe



137



83



84



# Password Cracking

- LinkedIn hack: Zelda puns
- [https://fossbytes.com/unix-co-founder-ken-thompson  
s-bsd-password-cracked/](https://fossbytes.com/unix-co-founder-ken-thompson-s-bsd-password-cracked/)

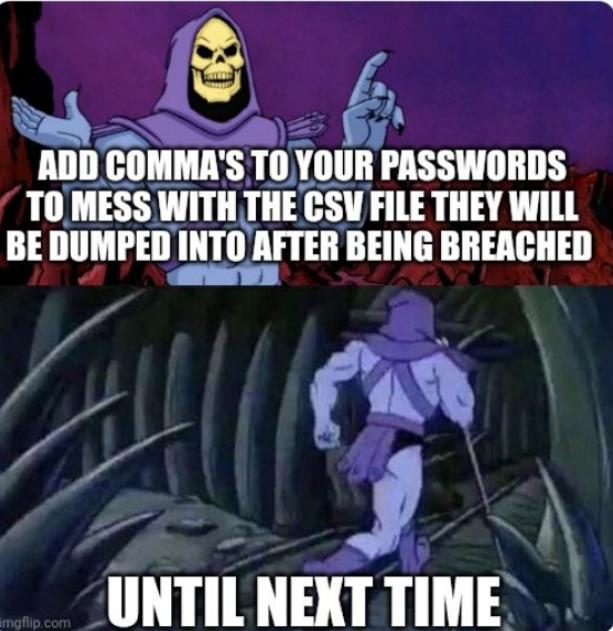
# Password Hashing



Brains93

@Brains933

What attackers don't want you to know



imgflip.com

2:25 AM · Nov 26, 2021 · Twitter for Android

