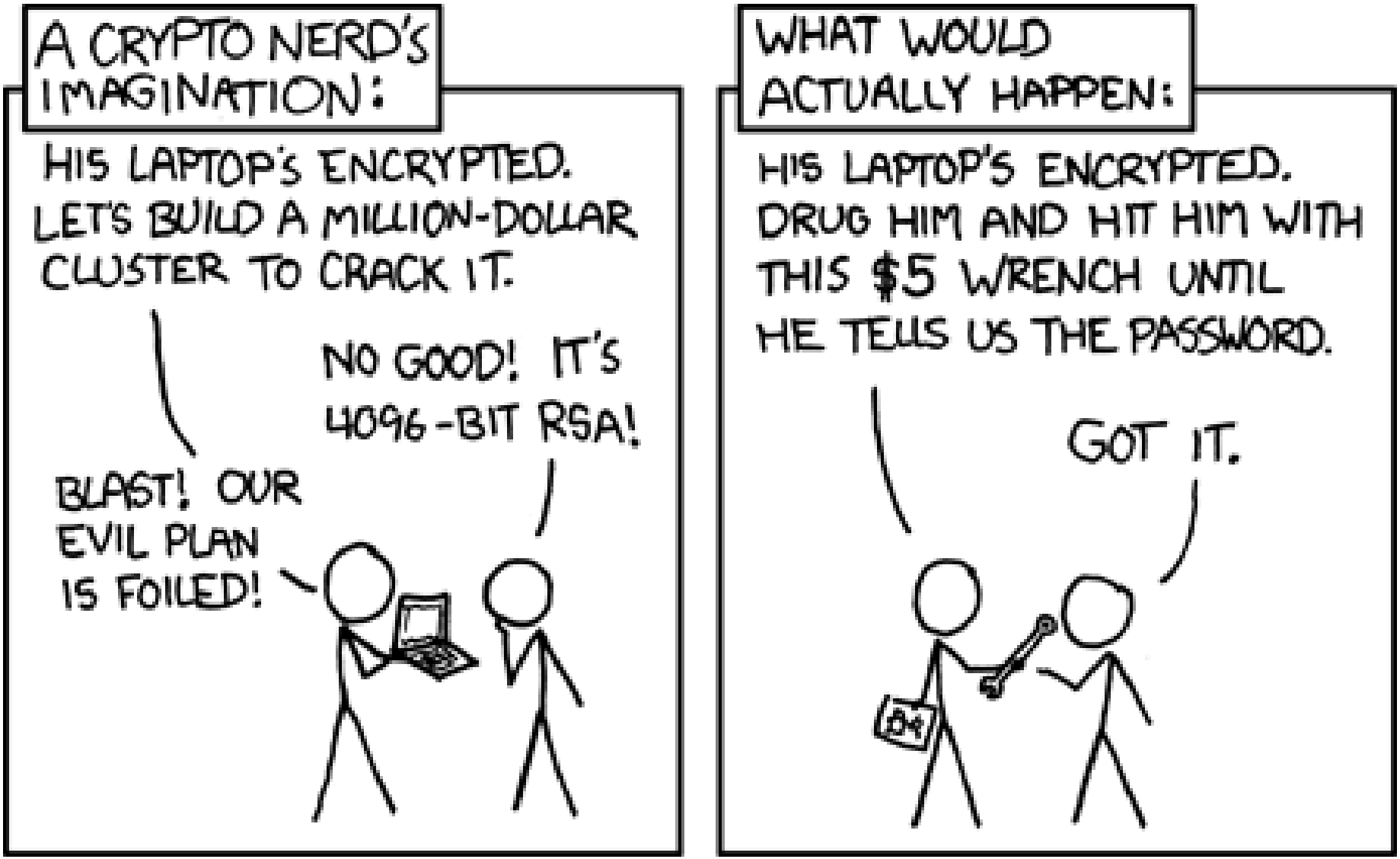


Source: https://xkcd.com/538/

CS 60:
Computer Networks

Secure comms

Review from last class

Ethernet is dominant wired technology at Layer 2

The Internet's core is primarily wired, but wireless is becoming increasingly important for end devices

Wireless modulation choices to impart data on signal:

- Amplitude
- Frequency
- Phase
- QAM (phase and amplitude)

Wireless must deal with

- Attenuation
- Multipath
- Interference

Agenda

1. Encryption
   - Symmetric (AES)
   - Hashing (SHA-2)
   - Asymmetric (RSA)
2. TLS
3. TOR
4. VPN
5. Wi-Fi security
6. Supplemental crypto material

Do not create your own cryptography system!

Use vetted crypto instead!

Discussion

What properties would you like in “secure” communications?

- Confidentiality (no one can read it)
- Integrity (hasn’t been changed)
- Availability (always up)
- Authentication (confirm identities)

Sometimes:

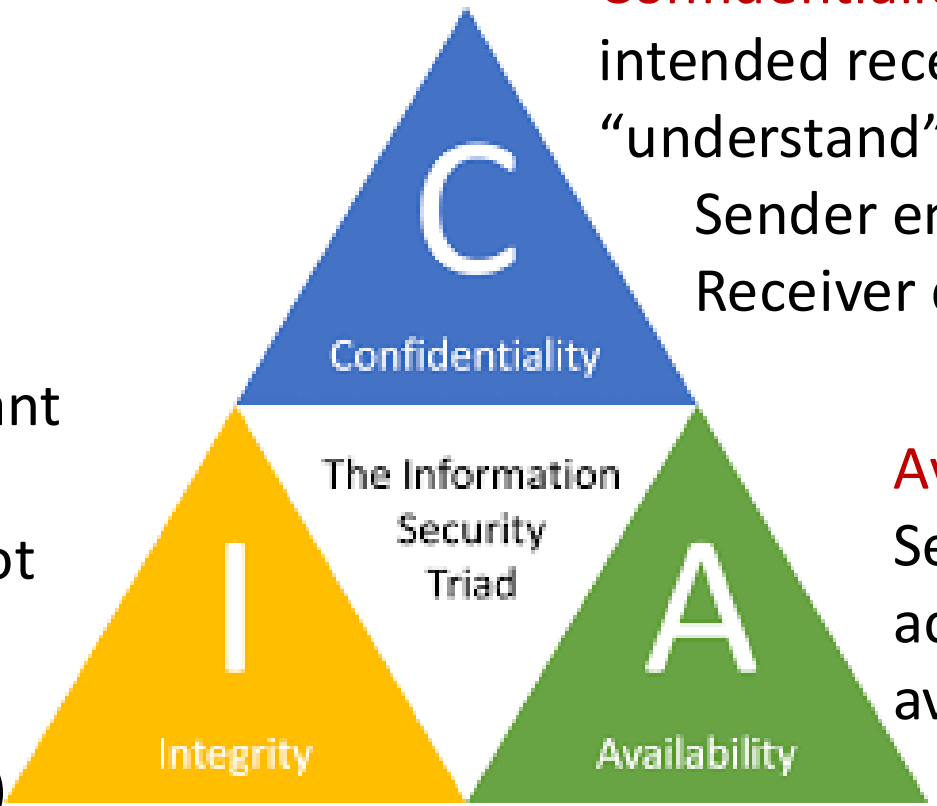
- Anonymity (can’t tell sender or receiver)
- Non-repudiation (can’t take it back if you send it)

CIA triangle: Confidentiality, Integrity, and Availability (Authentication too)

Confidentiality: Only sender and intended receiver should "understand" message contents

Sender encrypts message

Receiver decrypts message



Integrity: Sender and receiver want to ensure message not altered (in transit, or afterwards) without detection

Availability: Services must be accessible and available to users

Authentication: Sender and receiver want to confirm identity of each other

Discussion

What is the difference between security and anonymity?

Who needs anonymity?

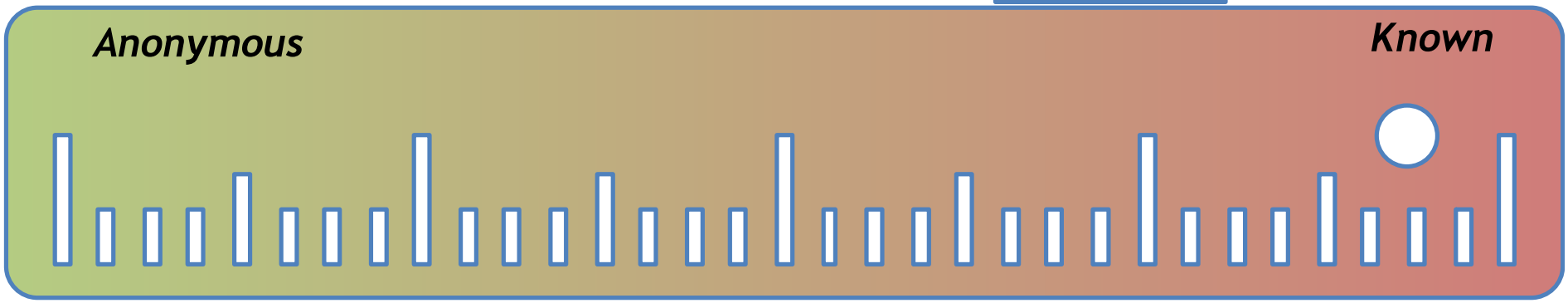
Sometimes you don't want anyone to know you've sent a message or to whom

CS10 SA-3
Steganography
LPI

cannot tell when a message is sent or not sent

non-trivial probability that the sender is not the node in question

sender whose communications can be identified is exposed.



can see that a message is sent but cannot be sure where it came from

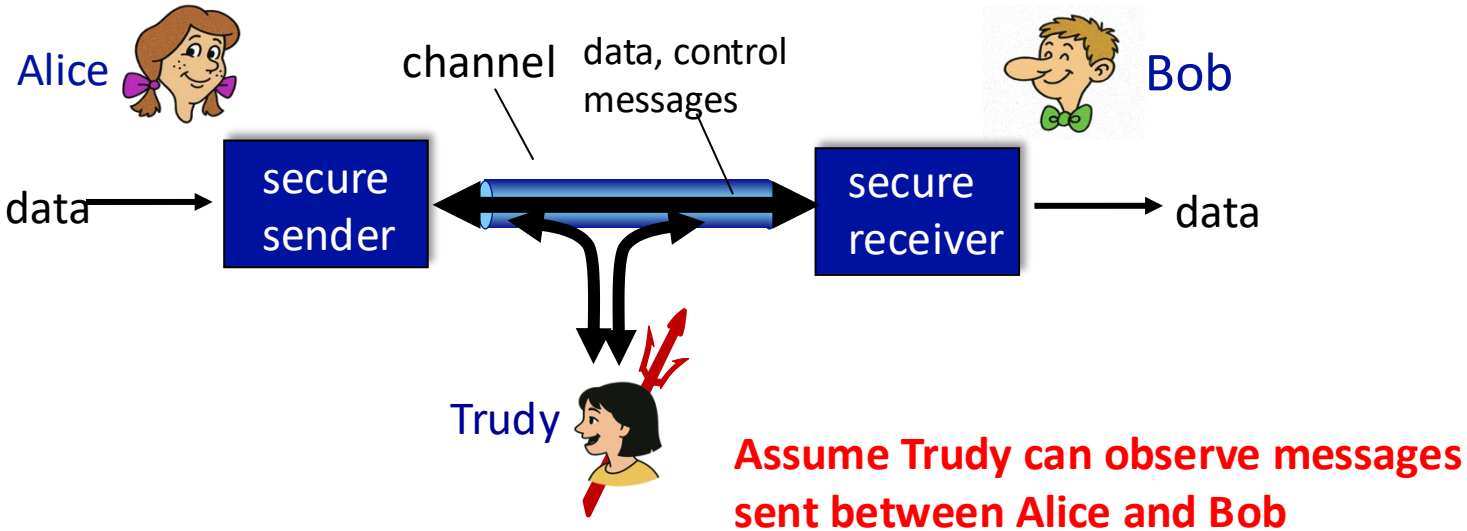
sender is probably innocent if the sender is no more likely than not to be the originator of a message

can demonstrate this exposure to other entities, the sender is provably exposed.

Prof. Palmer lecture notes

Friends and enemies: Alice, Bob, and Trudy (or Eve)

- Well-known in network security world
- Bob, Alice want to communicate “securely” over a channel
- Trudy (intruder) or Eve (eavesdropper) is a bad actor



Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

There are bad actors out there!

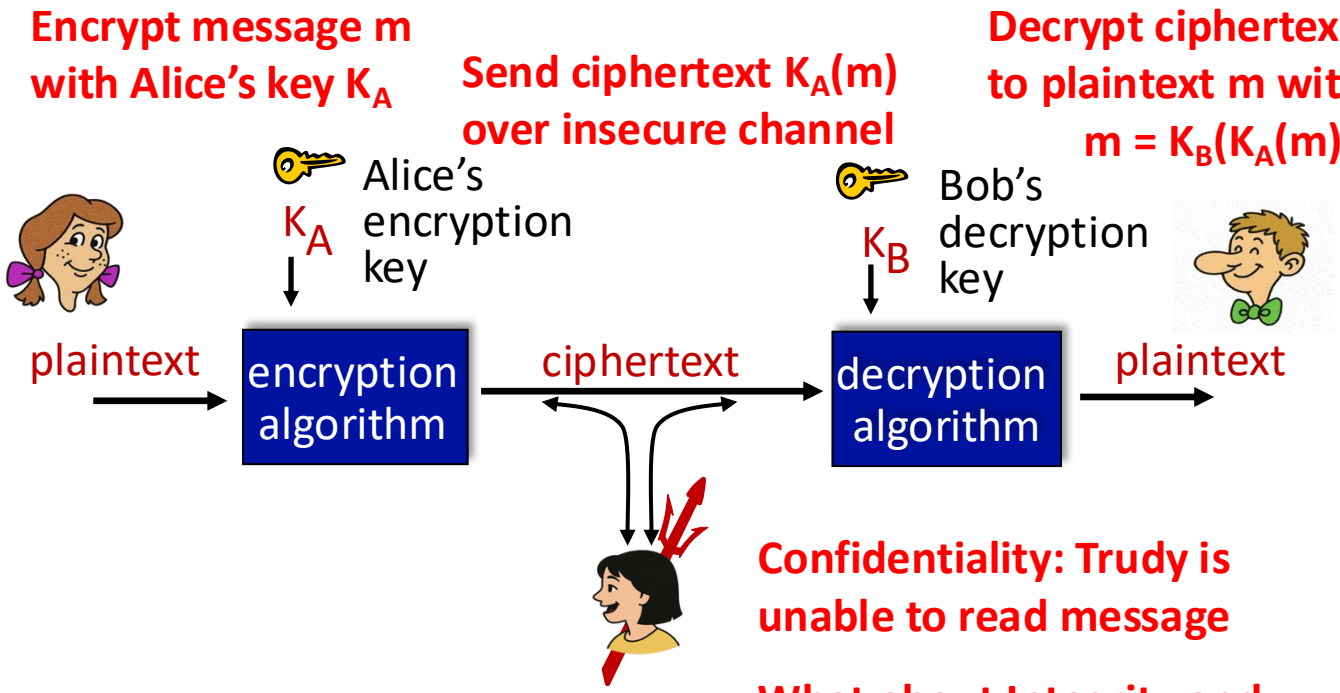
Q: What can a "bad actor" do?

A: A lot!

- Eavesdrop: intercept messages
- Actively insert messages into connection
- Impersonation: can fake (spoof) source address in packet (or any field in packet)
- Hijacking: "take over" ongoing connection by removing sender or receiver, inserting himself in place
- Denial of service: prevent service from being used by others (e.g., by overloading resources)

Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

The language of cryptography



m: plaintext message

$K_A(m)$: ciphertext, encrypted with key K_A

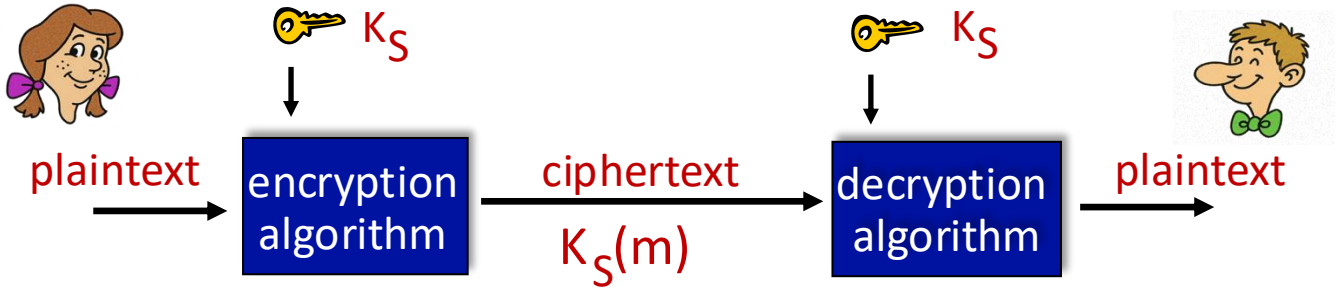
$m = K_B(K_A(m))$

Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

Agenda

1. Encryption
   - Symmetric (AES)
   - Hashing (SHA-2)
   - Asymmetric (RSA)
2. TLS
3. TOR
4. VPN
5. Wi-Fi security
6. Supplemental crypto material

Symmetric key cryptography means Alice and Bob both have the *same* key



Symmetric key crypto: Bob and Alice share same (symmetric) key: K

Q: How do Bob and Alice agree on a key?

Key exchange is difficult if Alice and Bob haven't met

More difficult if they are not connected to the Internet to check crypto

In Lab 4 you implement a way to exchange a key at the Physical Layer

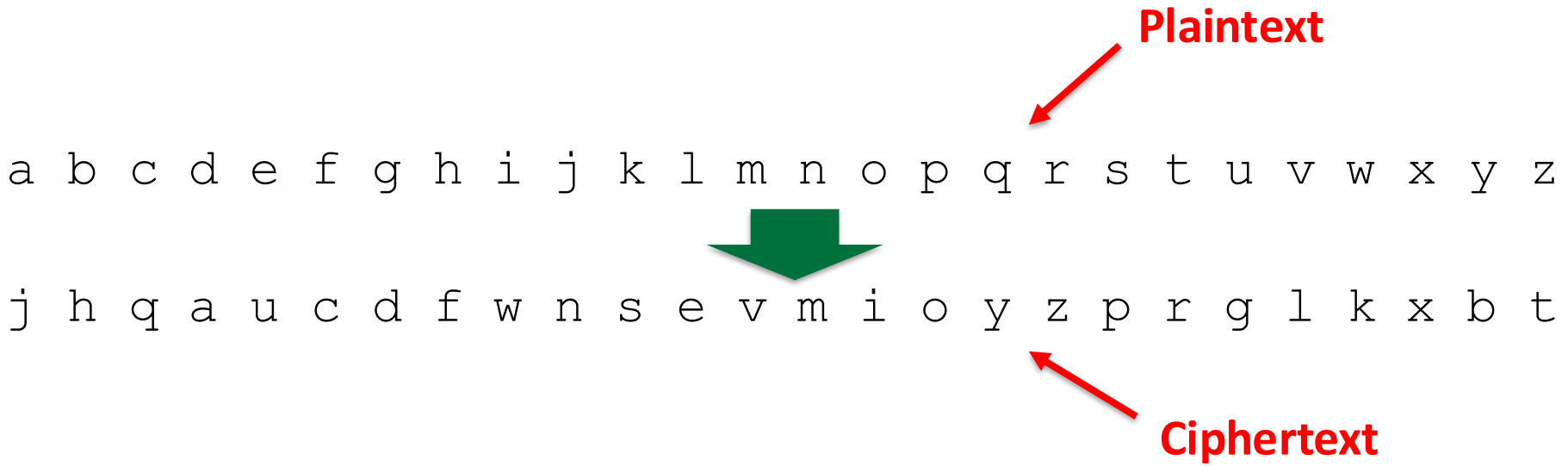
The Advanced Encryption Standard (AES) is a popular symmetric key scheme

- **Assumes both parties have the same key (does not distribute keys)**
- **Details at the end of this slide deck**

Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

With a substitution cipher, each character is replaced with another character

Monoalphabetic substitution cypher



Each character in plaintext has a corresponding character in the cipher

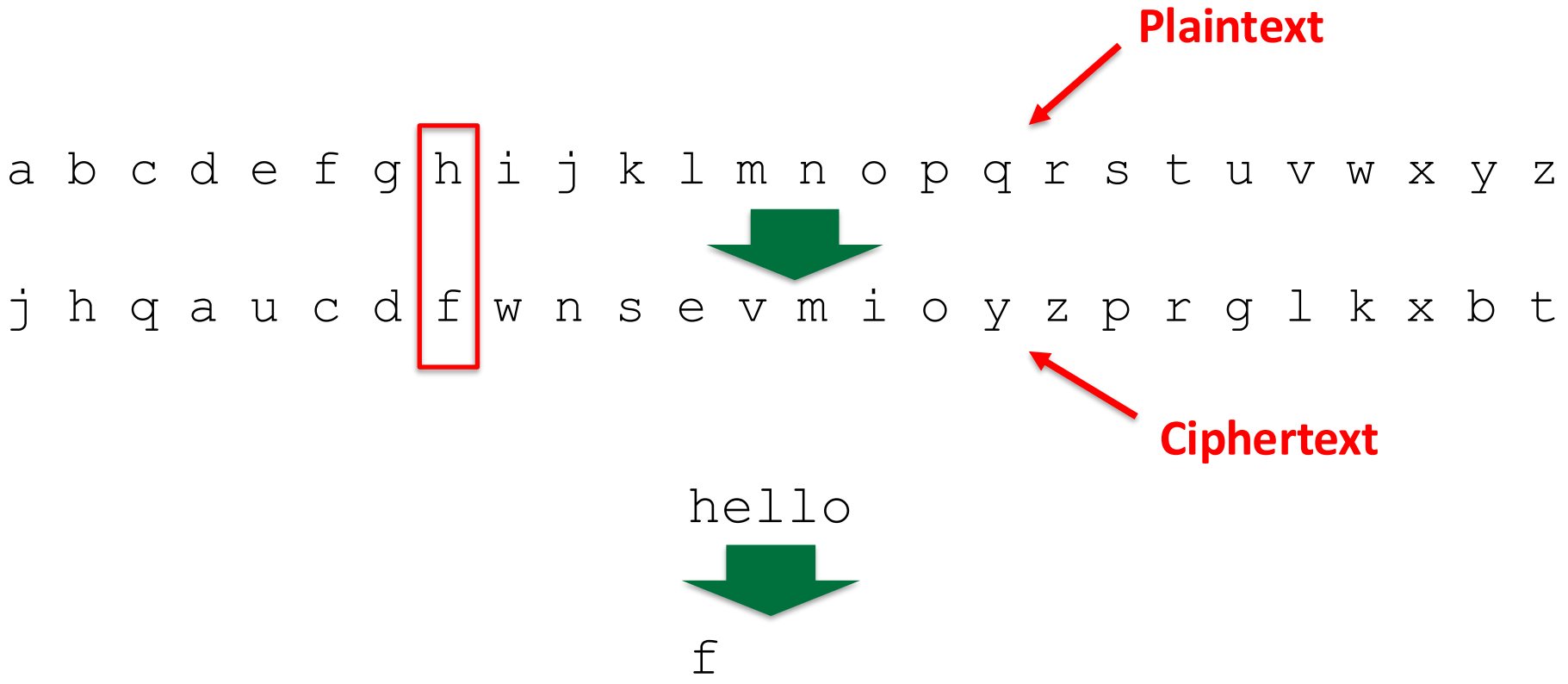
Replace the plaintext character with its matching ciphertext character

The symmetric key is the mapping of letters

Simpler Cesar cipher creates substitution by shift a-z by a fixed amount (amount to shift is the key)

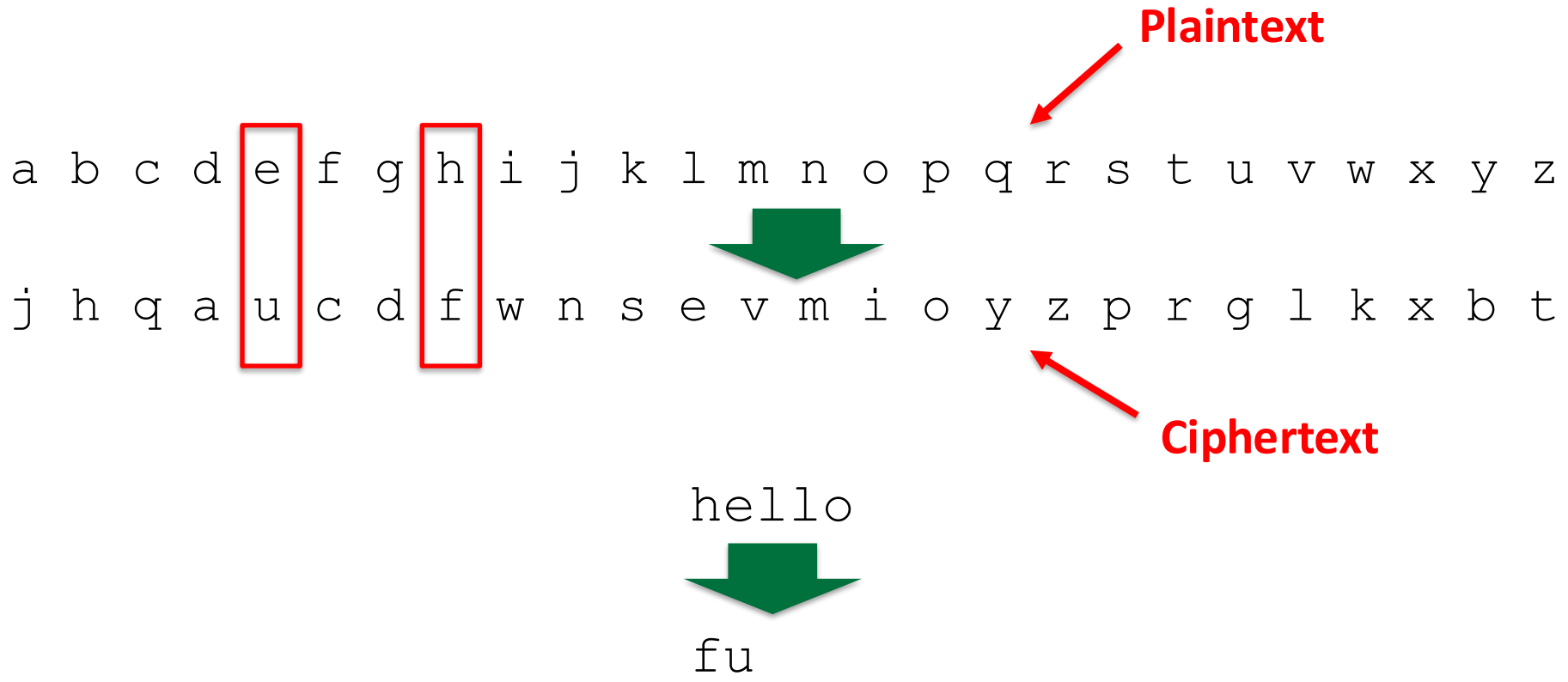
With a substitution cipher, each character is replaced with another character

Monoalphabetic substitution cypher



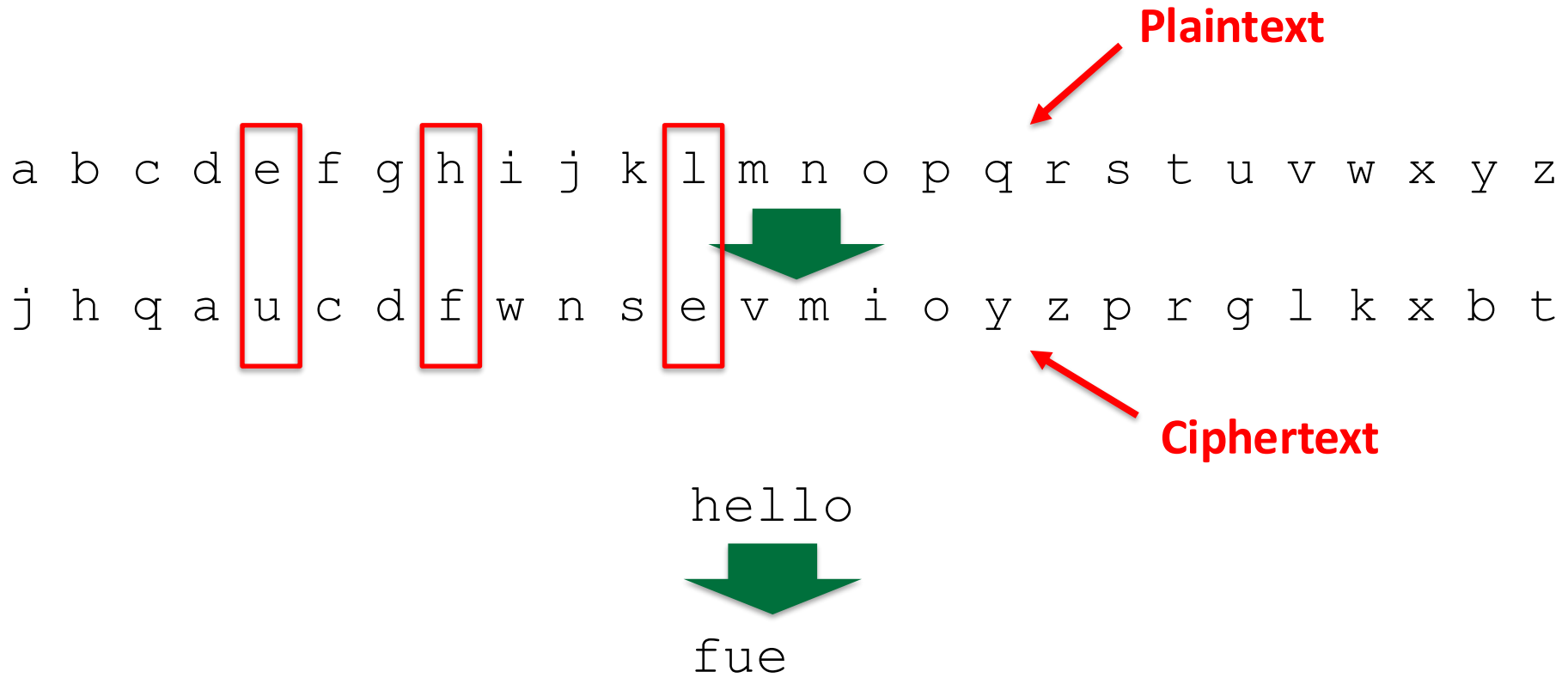
With a substitution cipher, each character is replaced with another character

Monoalphabetic substitution cypher



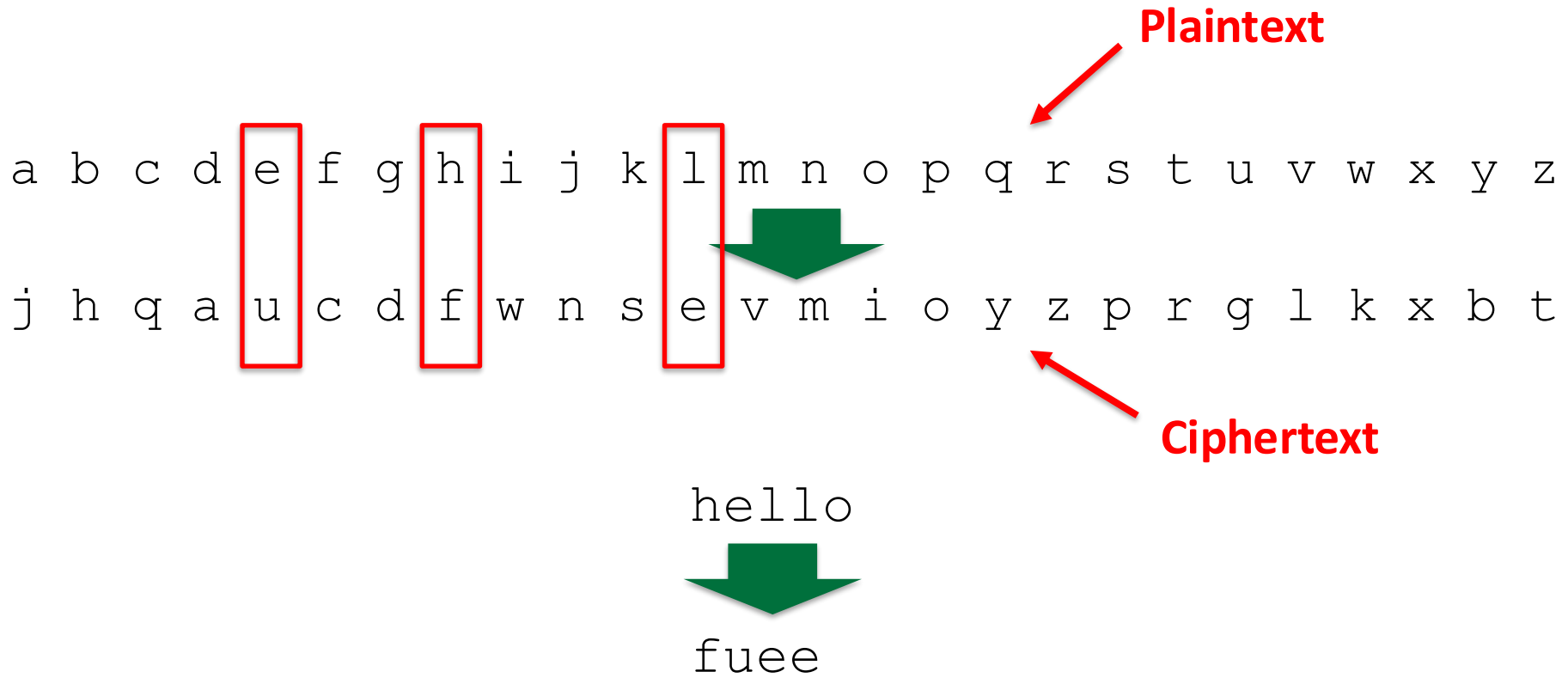
With a substitution cipher, each character is replaced with another character

Monoalphabetic substitution cypher



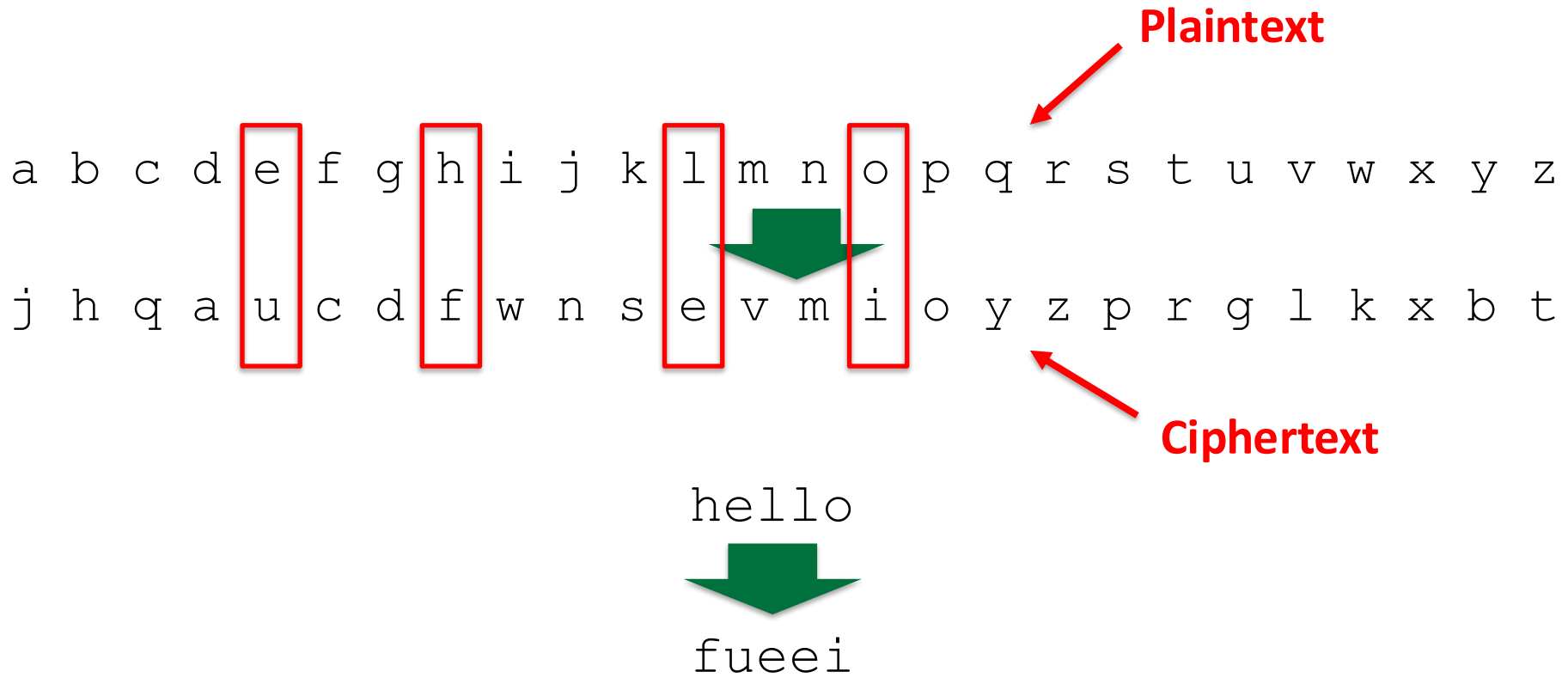
With a substitution cipher, each character is replaced with another character

Monoalphabetic substitution cypher



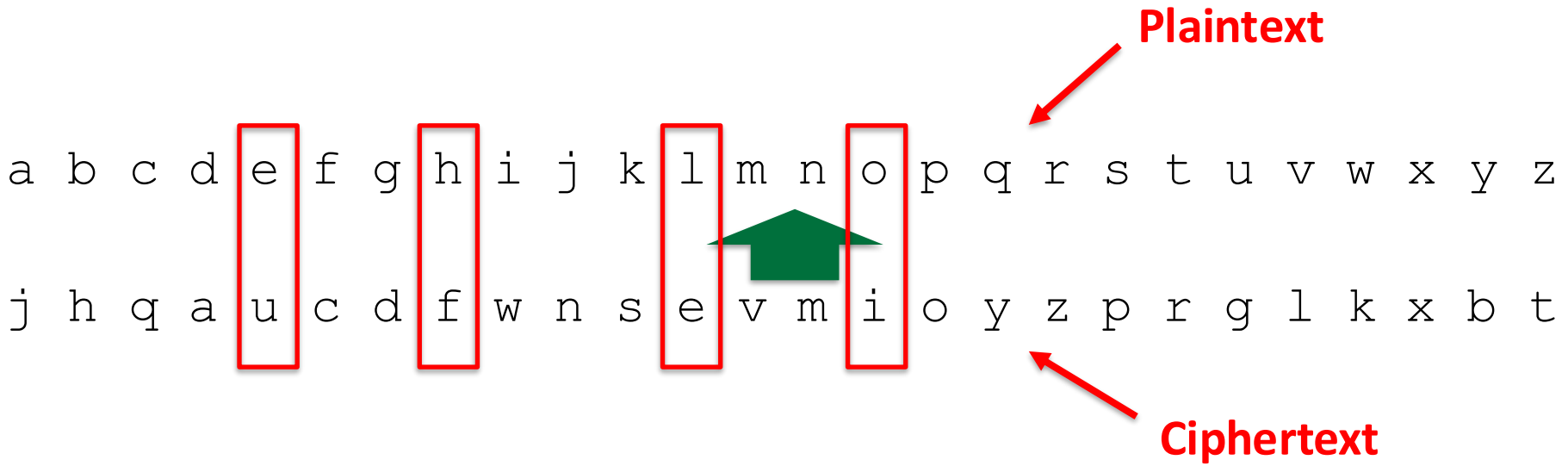
With a substitution cipher, each character is replaced with another character

Monoalphabetic substitution cypher



With a substitution cipher, each character is replaced with another character

Monoalphabetic substitution cypher



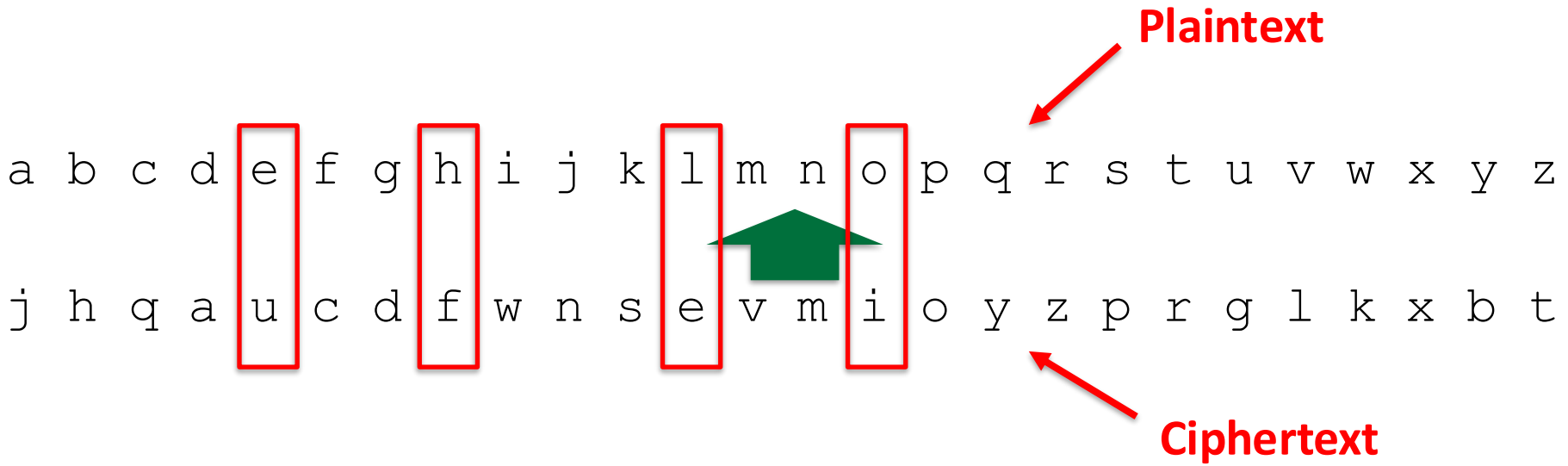
hello

fueei

Decrypt ciphertext by going in the reverse direction

With a substitution cipher, each character is replaced with another character

Monoalphabetic substitution cypher



In simple substitution, plaintext character is always replaced with the *same* ciphertext character ('a' always replaced with 'j')

hello

fueei

Key point: *both* sender and receiver must know the same mapping

26! Possibilities (more if we consider upper case, numbers, and punctuation)

- **400 septillion combinations**
- **Seems strong, right?**

The mapping is the key

Exercise

With a substitution cipher, encryption and decryption are done using the same key

Create key:

`python3 monoalphabetic.py`

- Randomly substitutes characters for a-z
- Example: `jhqaucdfwnsevmioyzprglkxbt`

```
import random

plain = "abcdefghijklmnopqrstuvwxyz"
cypher = random.sample(plain,len(plain))
print(''.join(cypher))
```

This is the key

Encrypt:

```
tr 'a-z' 'jhqaucdfwnsevmioyzprglkxbt' < plain.txt > cipher.txt
```

- Translates characters in `plain.txt` using key
- Saves output in `cipher.txt`
- `cat cipher.txt` to see the result

Works assuming both sender and receiver both know the key

Decrypt:

```
tr 'jhqaucdfwnsevmioyzprglkxbt' 'a-z' < cipher.txt > plain1.txt
```

- Translates characters in `cipher.txt` replacing with a-z
- Saves output in `plain1.txt`

Problem?

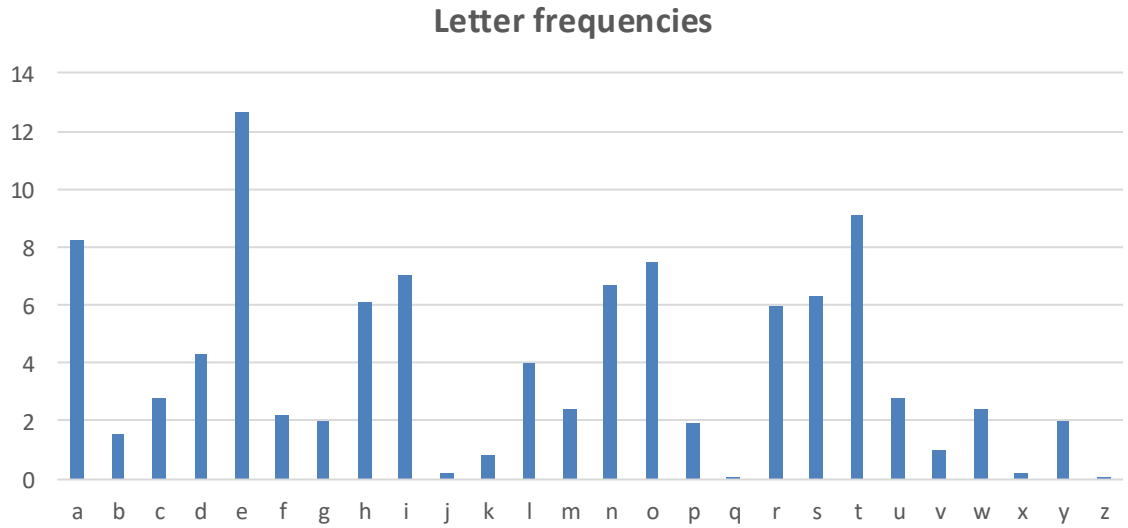
Frequency analysis!

Lab 5 will ask you to break this cipher

We can break substitution ciphers with frequency analysis

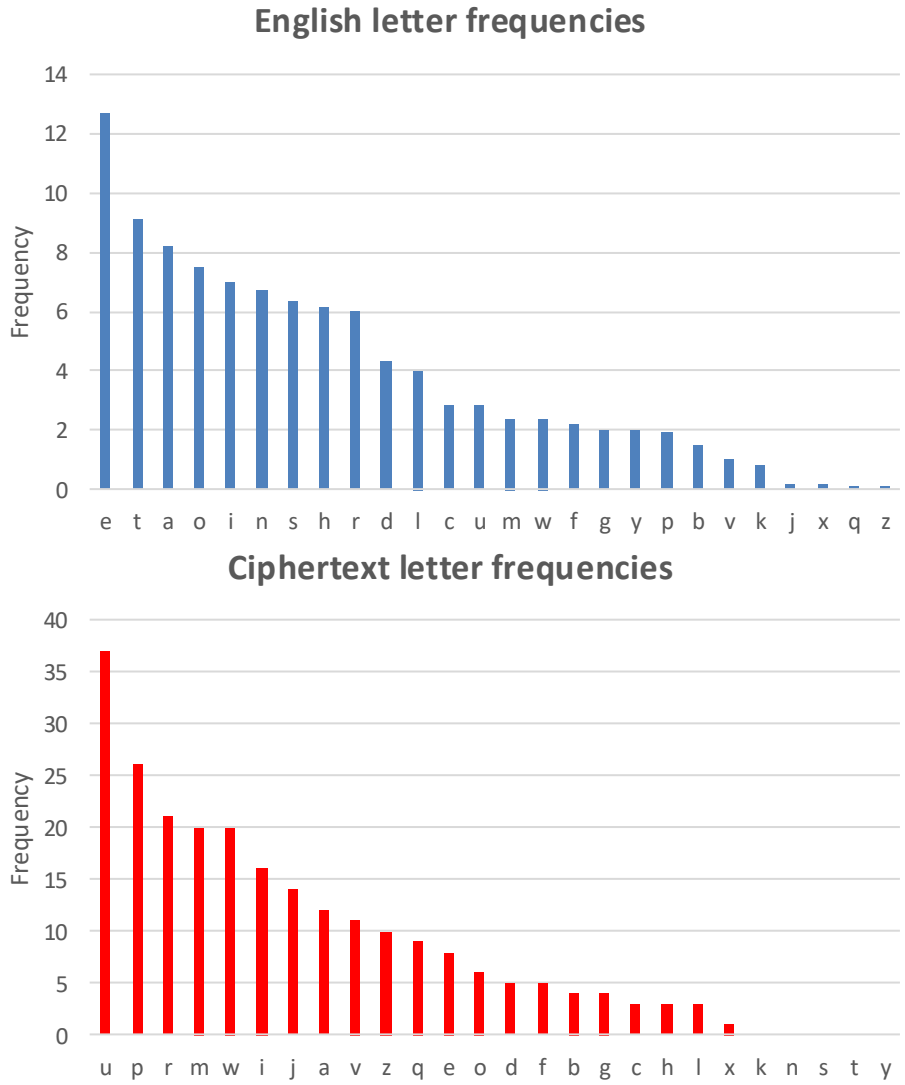
Key ideas:

- Know or guess which language (e.g., English, German, …) used in plaintext
- Look up how frequently different letters used in that language
  - e used in English most frequently
  - q and z not used frequently
- Letter that appears 100 times in plaintext will appear 100 times in ciphertext
- Letters are changed but letter frequencies are not (mapped 1:1)!
- Guess that frequent letters in ciphertext are frequent letters in English



http://www.richkni.co.uk/php/crypta/freq.php

Common letters in plaintext appear more frequently in ciphertext than others



Letter frequencies in English language[1]

```
python3 counts.py cipher.txt
```

gives counts of characters in cipher.txt file

Good chance u=e

tr u E < cipher.txt

Look for THE candidates

Look for AND candidates

…

Assumes the key was: `jhqaucdfwnsevmioyzprglkxbt` as in the example in Symmetric encryption section

[1] http://www.richkni.co.uk/php/crypta/freq.php

Polyalphabetic substitution ciphers are a better solution, but still not ideal

Key ideas:

- Instead of one key, create N keys (say 1000)
- Encode character at position i with i^{th} dictionary (wrap around if i > N)
- Now same character repeated results in different characters

Much more resistant to frequency analysis, but could be a problem if

- Use same set of N keys repeatedly
- Attacker sees large amount of ciphertext

See example in `polyalphabet.py`

Key point: both sender and receiver must know the set of N keys



German Enigma machine from WWII used polyalphabet principle

A transposition cipher rearranges the order of the letters without substitution

Transposition cypher

Message:

"nowrunalonganddontgetintomischiefiamgoingoutxxxx"

Padding

Write message as matrix (here 4 rows with 12 letters each)

```
nowrunalonga
nddontgetint
omischiefiam
goingoutxxxx
```

There *many* variations around this transposition idea

Encrypt by reading down columns

```
nnog odmo wdii
rosn uncg ntho
agiu leet otfx
niix gnax atmx
```

Decrypt by writing ciphertext column by column

```
n
n
o
g
```


Adapted from https://nrich.maths.org/7940

A transposition cipher rearranges the order of the letters without substitution

Transposition cypher

Message:

"nowrunalonganddontgetintomischiefiamgoingoutxxxx"

Padding

Write message as matrix (here 4 rows with 12 letters each)

```
nowrunalonga
nddontgetint
omischiefiam
goingoutxxxx
```

There *many* variations around this transposition idea

Encrypt by reading down columns

```
nnog odmo wdii
rosn uncg ntho
agiu leet otfx
niix gnax atmx
```

Decrypt by writing ciphertext column by column

```
no
nd
om
go
```


Adapted from https://nrich.maths.org/7940

A transposition cipher rearranges the order of the letters without substitution

Transposition cypher

Message:

"nowrunalonganddontgetintomischiefiamgoingoutxxxx"

Padding



Write message as matrix (here 4 rows with 12 letters each)

```
nowrunalonga
nddontgetint
omischiefiam
goingoutxxxx
```

Key point: In a transposition cipher, characters change positions but are not substituted with other characters

There *many* variations around this transposition idea

Encrypt by reading down columns

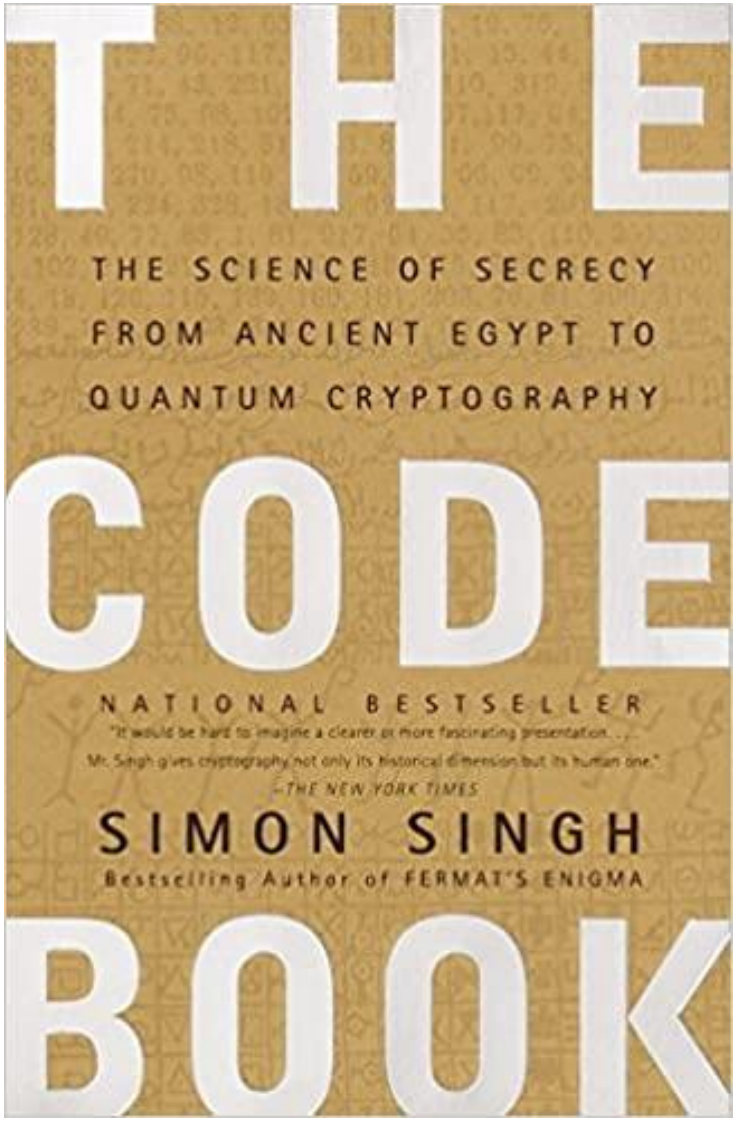
```
nnog odmo wdii
rosn uncg ntho
agiu leet otfx
niix gnax atmx
```

Decrypt by writing ciphertext column by column

```
nowrunalonga
nddontgetint
omischiefiam
goingoutxxxx
```


Adapted from https://nrich.maths.org/7940

Book recommendation



If you are interested in learning more about secret codes used throughout history

The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Computing

By Simon Singh

https://www.amazon.com/Code-Book-Science-Secrecy-Cryptography/dp/0385495323

Covers many other ciphers!

XOR has some nice properties and is often used in cryptography

XOR

Truth table

	0	1
0	0	1
1	1	0

XOR's truth table is like OR but differs if both bits are 1

Returns 1 only if inputs are different, otherwise returns 0

Two possible bit combos yield 1 and two possible bit combos yield 0

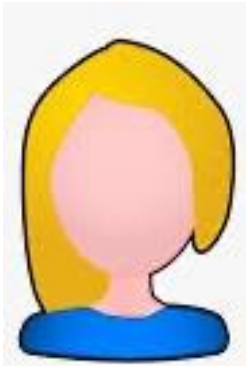
XOR has some nice properties and is often used in cryptography

XOR

Properties

1. $A \oplus 0 = A$
2. $A \oplus A = 0$
3. $(A \oplus B) \oplus C = A \oplus (B \oplus C)$
4. $(B \oplus A) \oplus A = B \oplus 0 = B$

Alice XOR's Message with Key to create Ciphertext



Alice



Bob

Problems?

- **Frequency analysis**
- **Known ciphertext attack**

Alice

```
Key:      10101010
Message:  11000011
Cipher:   01101001
```



Eve

Eve sees cipher, but does not know Key, does not recover message

Alice sends Ciphertext to Bob

Bob knows Key (shared secret)

Bob XOR's Ciphertext and Key to recover Message (property 4 above)

Bob

```
Cipher--> 01101001
Key:      10101010
Message:  11000011
```

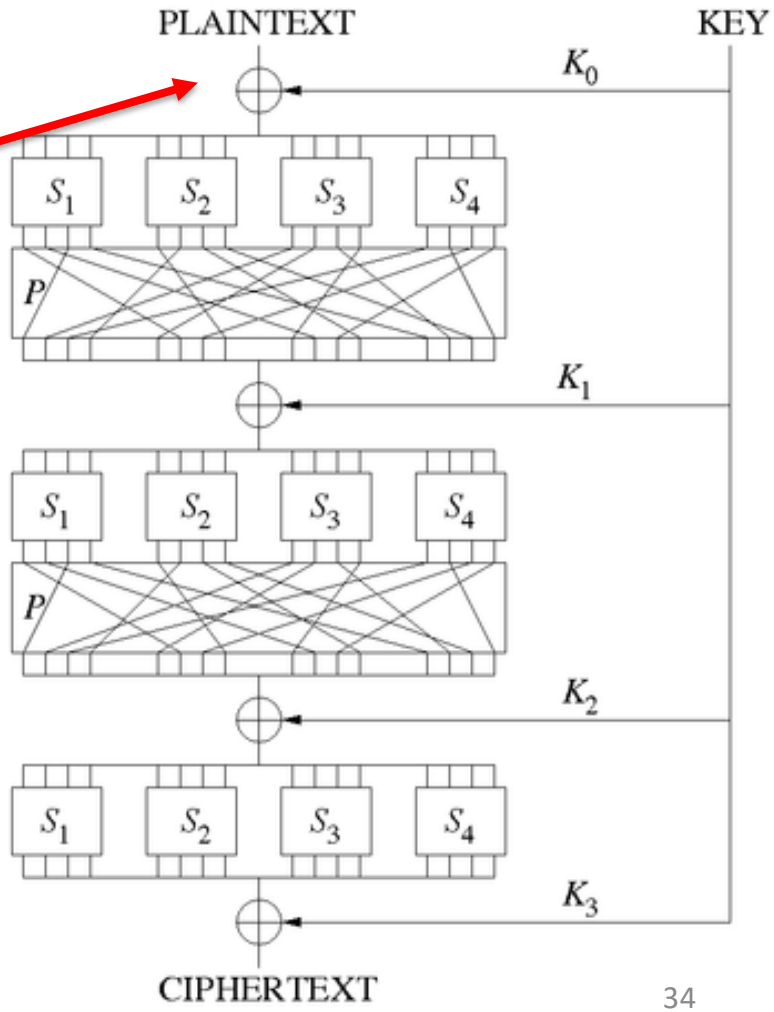
S-P Networks: Intuition for how almost all modern symmetric key cryptography works

Substitution-Permutation Network (S-P network)

Simplified

Key expanded into several "round keys" by key schedule algorithm

Fixed sized block of plaintext XOR'ed with key



Adapted from: Mao, Wenbo. "Modern cryptography: theory and practice." (2004).

S-P Networks: Intuition for how almost all modern symmetric key cryptography works

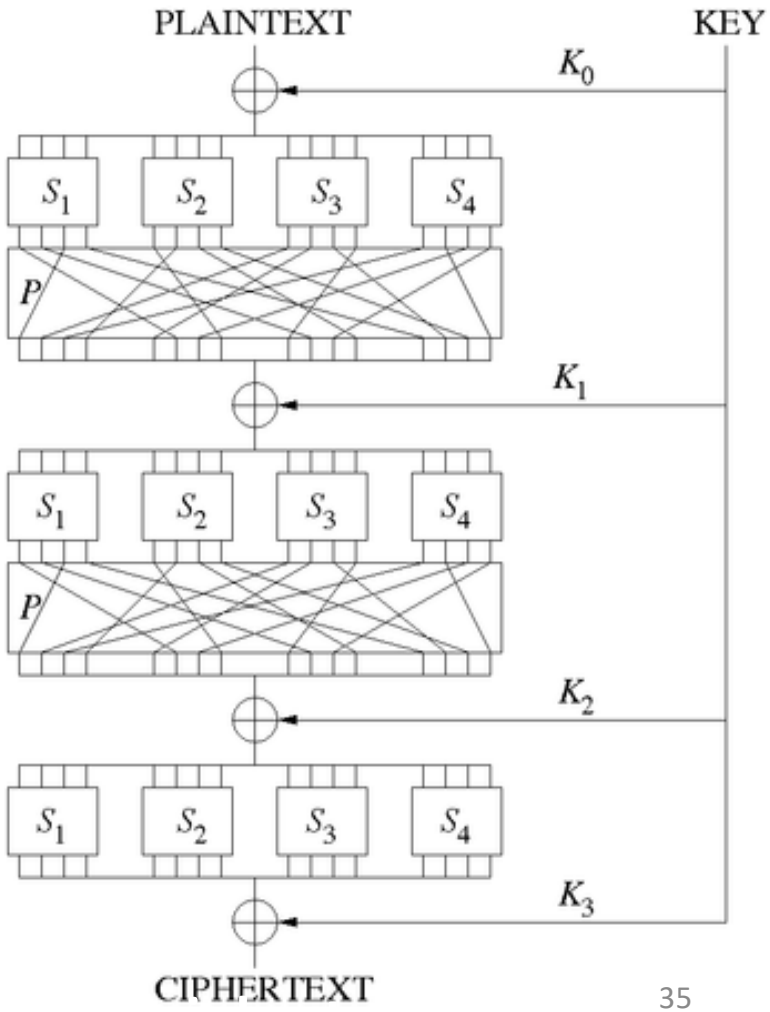
Substitution-Permutation Network (S-P network)

Simplified

Substitution (S-box)

- **Look up table – given inputs bits, output different bits**
- **1-to-1 mapping for invertibility (decryption)**
- **Often done in hardware for speed**
- **Provides *confusion***

Can think of S-box like a *substitution* cipher



Adapted from: Mao, Wenbo. "Modern cryptography: theory and practice." (2004).

S-P Networks: Intuition for how almost all modern symmetric key cryptography works

Substitution-Permutation Network (S-P network)

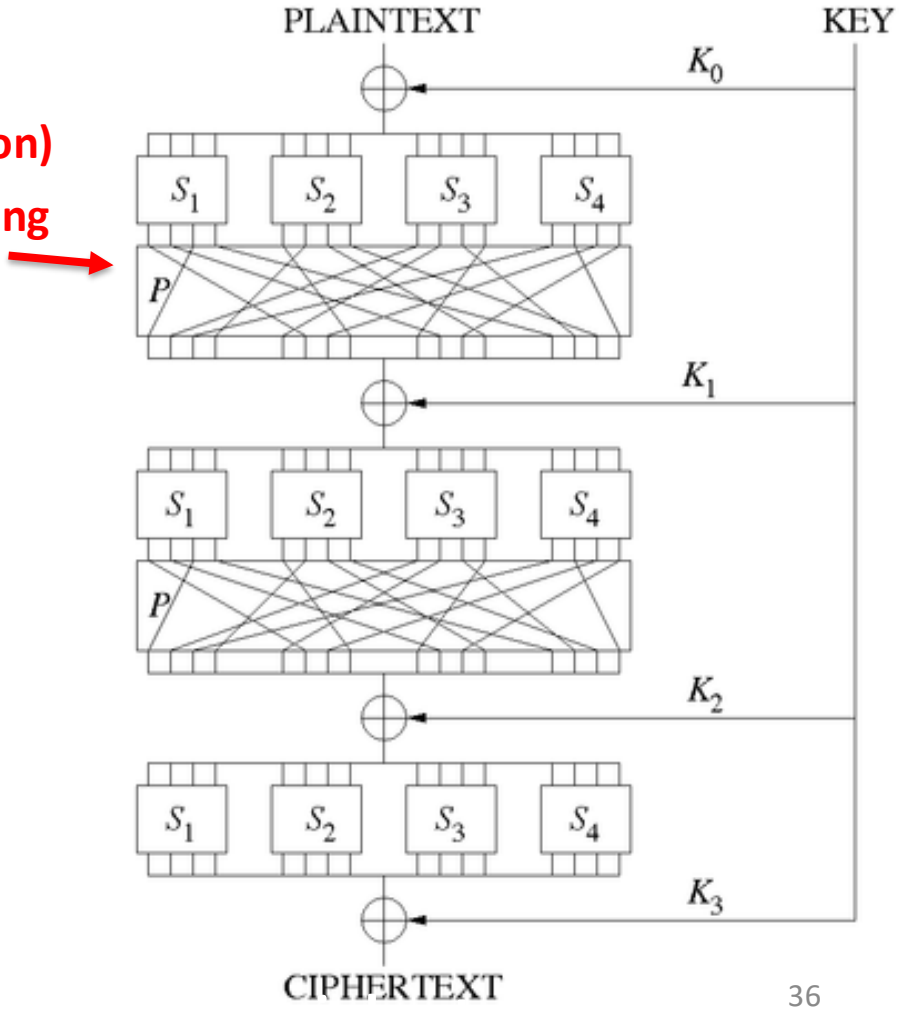
Simplified

Permutation (P-box)

- **Permutes bits (moves bits to new location)**
- **Distributes output its to as many following S-boxes as possible**

Can think of P-box like a *transposition* cipher

Provides *diffusion*



Adapted from: Mao, Wenbo. "Modern cryptography: theory and practice." (2004).

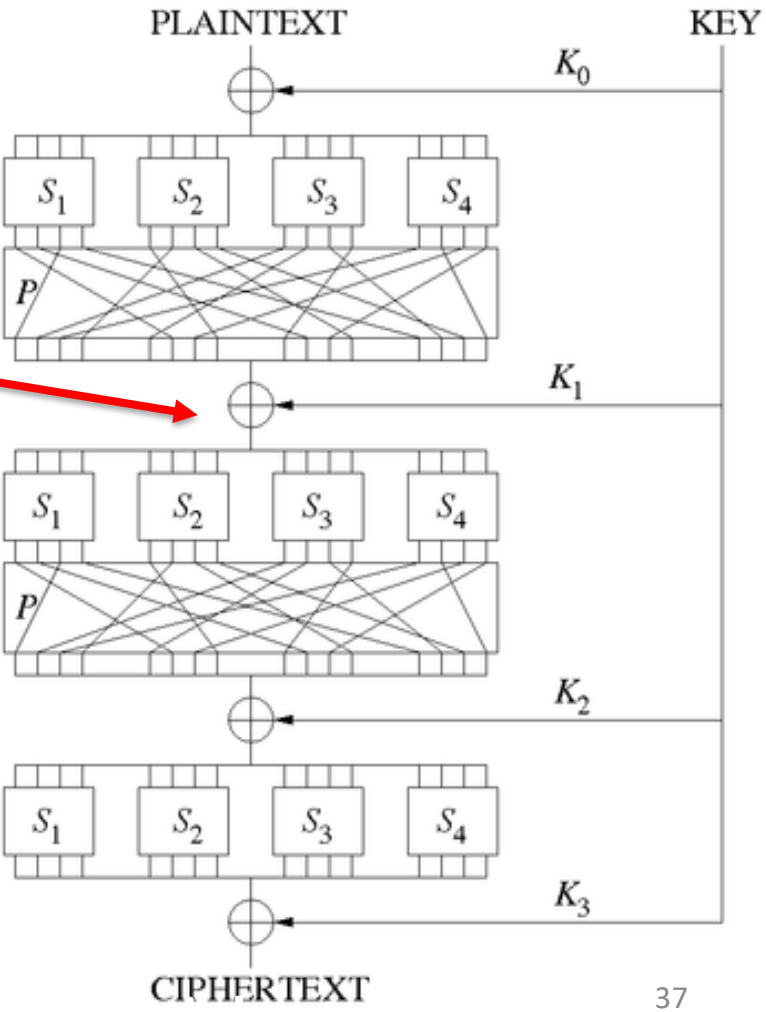
S-P Networks: Intuition for how almost all modern symmetric key cryptography works

Substitution-Permutation Network (S-P network)

Simplified

Output is XOR'ed with next round key

Process repeats for several rounds



Adapted from: Mao, Wenbo. "Modern cryptography: theory and practice." (2004).

S-P Networks: Intuition for how almost all modern symmetric key cryptography works

Substitution-Permutation Network (S-P network)

Simplified

A single S-box or P-box does not provide much cryptographic strength

A S-P network with several rounds provides both confusion and diffusion

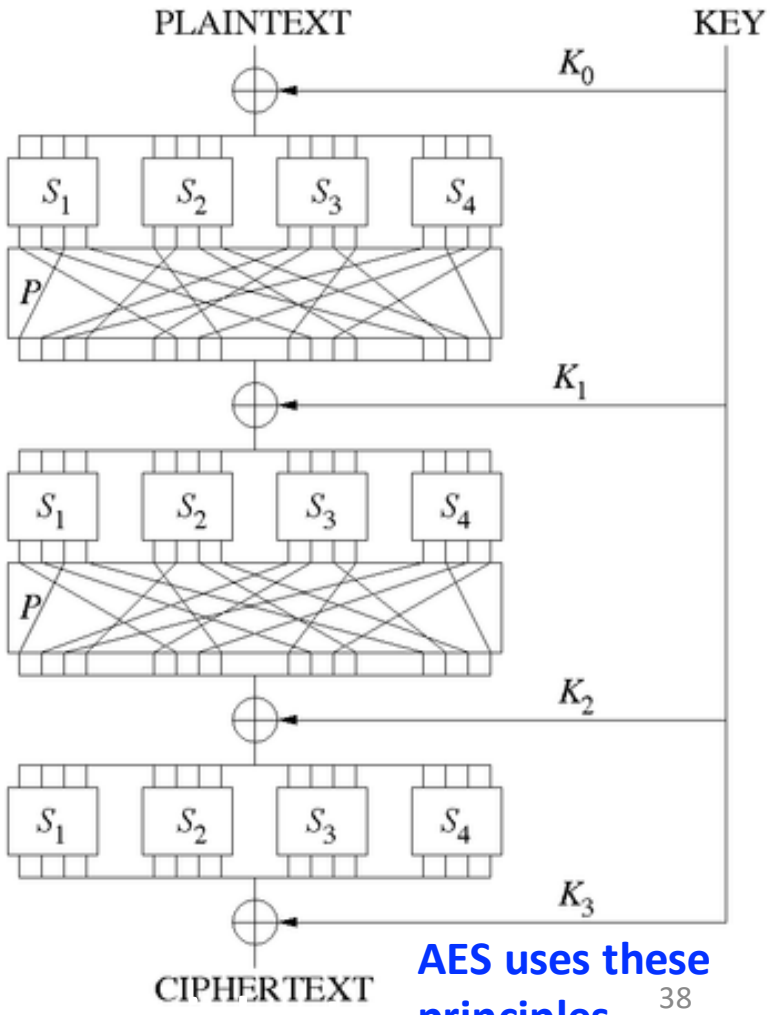
- If one input bit changes, several output bits will change
- Bits are distributed among several S-boxes

Changing one bit of input should change roughly half of the output bits (avalanche effect)

Therefore, each output bit depends on every input bit

If i^{th} input bit flipped, probability j^{th} output bit flips is 50%

Changing one bit of input diffuses over all output bits



Reverse to decrypt

AES uses these principles

Adapted from: Mao, Wenbo. "Modern cryptography: theory and practice." (2004).

Agenda

1. Encryption
   - Symmetric (AES)
   - Hashing (SHA-2)
   - Asymmetric (RSA)
2. TLS
3. TOR
4. VPN
5. Wi-Fi security
6. Supplemental crypto material

Let's start with a game

Rules

You and I will both pick a number, if the sum is:

- Even – I win
- Odd – you win

Protocol

- Tell me your number!
- I will then tell you mine

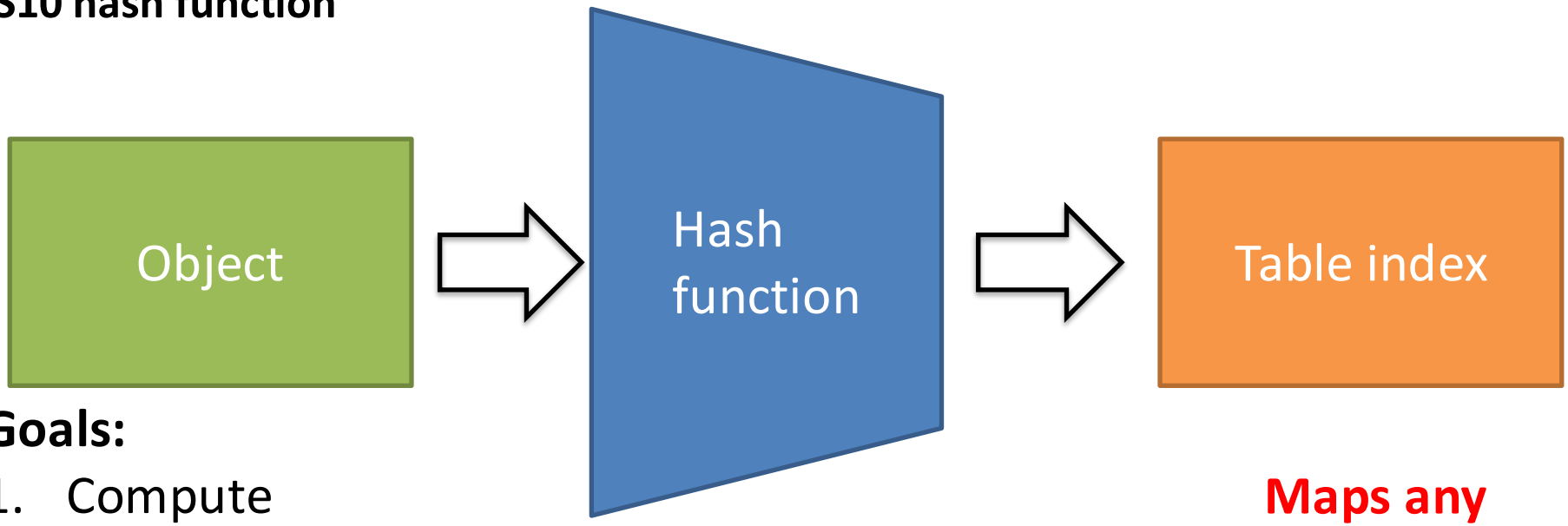
Works if you trust me to be honest, if not…

Hashing can help here

Adapted from Du, Wenliang. *Computer & Internet Security: A Hands-on Approach*. 2019.

Back in CS10 we looked at hashing to find an index in table based on an object

CS10 hash function



Goals:

1. Compute quickly and consistently
2. Spreads keys over the table
3. Small changes give different numbers

Step 1: convert object to integer

- Range –infinity to infinity
- Use hashCode()

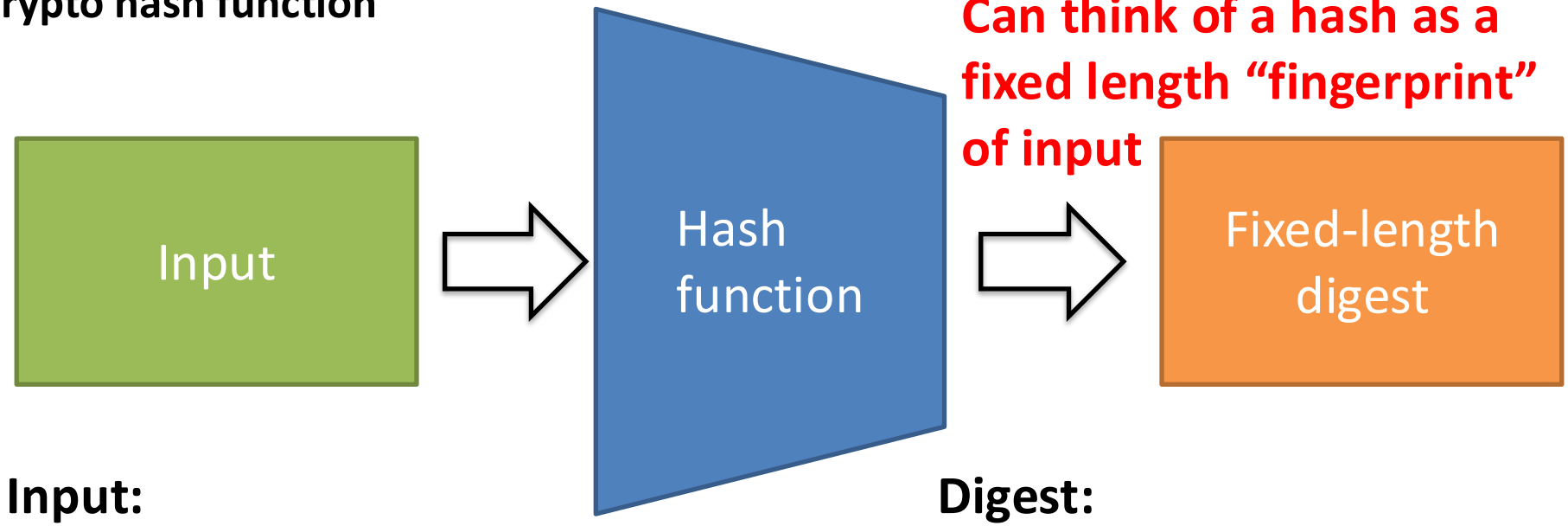
Step 2: constrain to fall within hash table (hashCode %m)

Maps any object into table index from 0..m-1

Output is fixed length

Crypto hash functions add two properties: irreversibility and collision resistance

Crypto hash function



Can think of a hash as a fixed length "fingerprint" of input

Input:
"my secret message"

Digest:
061ef7d71233886440043538e4d91de3

Output is fixed length

Same input always gives same output

Hash function is a ***deterministic*** mathematical function with two additional properties

1. Irreversible – Cannot find plain text in "reasonable" amount of time given only the hash digest output (called a one-way function)
2. Collision resistant – different plaintexts do not produce same hash digest

Pseudorandom - change any bit and get very different result

Let's visit the game with a new protocol

Rules

You and I will both pick a number, if the sum is:

- Even – I win
- Odd – you win

Fair to you:

- **You can confirm a hash of my number matches the commitment, so you know I didn't change my number**
- **Collision resistance means I can't give you a different number whose hash matches commitment**

Protocol

- I hash my number and give you the result (commitment)
- You then tell me your number
- I then tell you my number
- You can confirm I didn't cheat by hashing my number and comparing with commitment

Is this fair to both parties? Why or why not?

Fair to me:

- **You cannot reverse my commitment to get my number due to one-way property**
- **Therefore, you can't pick a winning number because got my number from the commitment**

Problem:
If range of numbers is too small (1 to 10) hash all numbers and match commitment with one of the hashes

Adapted from Du, Wenliang. *Computer & Internet Security: A Hands-on Approach*. 2019.

MD5 and SHA are popular choices, but SHA-2 is currently the most popular choice

MD5 (Message Digest 5)

- Popular choice of hash function
- Designed by Ron Rivest of RSA fame in 1991
- Collision resistance broken in 2004
- Can still use, but not if collision resistance is important

SHA (Secure Hash Algorithm)

- SHA-1
  - 160-bit hash function
  - Not recommended after 2005
  - Collision found in 2017 (two different pdfs gave same hash)
- SHA-2
  - Two popular flavors: SHA-256 and SHA-512 (number = length of hash)
  - No practical exploits… yet
- SHA-3
  - New as of 2015
  - Works differently from MD5 and SHA-2 (in case gets SHA2 broken)

SHA-2 the most common choice today

Don't use MD5 anymore for most purposes

Macs and Linux come with utilities to compute hashes from the command line

#get md5 hash of plain.txt file
#hashes to fixed length of 128 bits (16 bytes)
$ **md5sum plain.txt**
MD5 (plain.txt) = 6cc0536c2c4fae6e79dd7f3b7b338d3d

Key point: Same input into a hashing algorithm always yields the same output (SHA-512 output is the same)

#get SHA-256 hash of plain.txt file
#hashes to fixed length of 256 bits (32 bytes)
$ **shasum -a 256 plain.txt**
8496285a4cbb68b806510621c9f683c9f9d6a52fe9714cbfb354f718f990f6bd plain.txt

Output will vary if using different algorithms (SHA-512 output different from SHA-256 output)

#get SHA-512 hash of plain.txt file
#hashes to fixed length of 512 bits (64 bytes)
$ shasum -a 512 plain.txt
bbb3001c9640b5fa96d5daa40717f07481410b782c4f5bbf2a1c2bd1542cfd5290ffebe04c02496
6a6dfaec029a3f3b6da4ad4c0c796a656a6ca2bc937600351 plain.txt

Hashing is deterministic, no random component!

#can also use openssl
$ **openssl dgst -sha512 plain.txt**
SHA2-512(plain.txt)=
bbb3001c9640b5fa96d5daa40717f07481410b782c4f5bbf2a1c2bd1542cfd5290ffebe04c02496
6a6dfaec029a3f3b6da4ad4c0c796a656a6ca2bc937600351

You can calculate hashes in Python using hashlib

compute_hash.py

```
try:
        #set up hashlib for SHA512
        m = hashlib.sha512()

        #read data file line by line
        f = open(sys.argv[1],'r')
        for line in f:
                #update after each line
                m.update(line.encode("utf-8"))
        f.close()
        #print hash of file
        print(m.hexdigest())
except:
        traceback.print_exc()
```

This Python code reads a file line by line and hashes each line

Result is in m.hexdigest()

To confirm type "sha512sum <filename>" on command line and compare with this program's output

Note: same result if read line by line and hash each line or if read the whole file and then hash the entire file

You can also calculate hashes in C using OpenSSL

compute_hash.c

Compile with: gcc compute_hash.c -o compute_hash -lcrypto
Run with: compute_hash plain.txt
(Some code snipped for space on slide)

```
void main(int argc, char *argv[]) {
        SHA512_CTX ctx;
        u_int8_t results[SHA512_DIGEST_LENGTH];
        FILE * fp;

        SHA512_Init(&ctx); //set up for hashing

        fp = fopen(argv[1], "r"); //open file for reading
        if (fp == NULL) exit(EXIT_FAILURE);

        while ((read = getline(&line, &len, fp)) != -1) { //read lines
                SHA512_Update(&ctx, line, strlen(line)); //update digest
        }

        SHA512_Final(results,&ctx); //finalize hashing

        //print results
        for(i=0;i< SHA512_DIGEST_LENGTH;i++) {
                printf("%02x",results[i]);
        }
        printf("\n");
}
```

Returns the same digest as Python and the command line

Use case 1: Committing to a secret without revealing it

Committing to a secret **Useful for Lab 4**

- Our game
- Hash secret and publish hash
- Disclosing hash does not disclose secret – irreversible (one way)
- Once hash is published, cannot change document without being detected – collision resistance

Example

- You've figured out a way to predict the stock market
- You don't want to tell everyone beforehand (or price goes up)
- Hash your predictions (with random nonce) and post them
- Later people can check your predictions were accurate

Adapted from Du, Wenliang. *Computer & Internet Security: A Hands-on Approach*. 2019.

Use case 2: Integrity verification

Integrity verification (e.g., determine nothing has changed, or if it has!)

- Hash the original object
- To see if the object hash been changed, compare the original hash with a hash of the current version of the object

Hash is equivalent to a duplicate copy because if anything changed in OS file, hashes won't match

Example

- Want to know if OS kernel files have been changed by malware
- Could keep a spare copy of each OS file and check when going to run if it matches the spare copy – wasteful/impractical
- Instead, hash OS kernel files and store the 32-byte hash (not a second copy)
- Adversary might change kernel files, inserting malicious code
- Before running code, hash it and compare with stored version
- If no match, do not run it!

Many files on the Internet post a hash of the original
Compare hash of what you downloaded with posted hash to confirm you got an unaltered copy (is this good enough?)

Adapted from Du, Wenliang. *Computer & Internet Security: A Hands-on Approach*. 2019.

Use case 3: Password verification

Password verification

- Do not store password in plaintext
- Store hashed version of password
- Use ***salt*** to defeat dictionary/rainbow tables
- To authenticate: enter password, hash it, compare with stored

User name

Example

- In Linux passwords stored in /etc/shadow
- seed:6wDRrWCQz$IsBXp9.9wz9SGrF.nbihpoN5w.zQx02sht4cTY8qI7YKh00wN/sfYvDeCAcEo2QYzCfpZoaEVJ8sbCT7hkxXY/:17372:0:99999:7:::

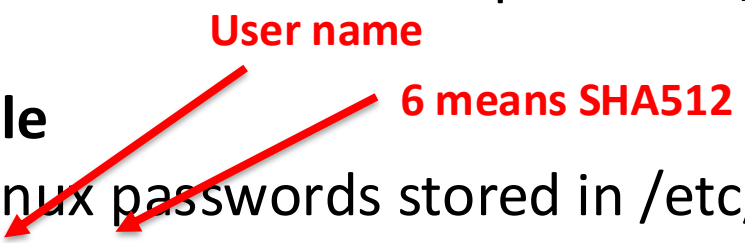
Adapted from Du, Wenliang. *Computer & Internet Security: A Hands-on Approach*. 2019.

Use case 3: Password verification

Password verification

- Do not store password in plaintext
- Store hashed version of password
- Use salt to defeat dictionary/rainbow tables
- To authenticate: enter password, hash it, compare with stored

Example



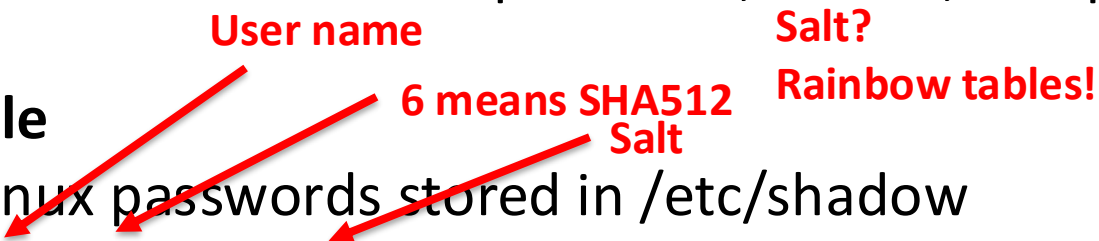
- In Linux passwords stored in /etc/shadow
- seed:6wDRrWCQz$IsBXp9.9wz9SGrF.nbihpoN5w.zQx02sht4cTY8qI7YKh00wN/sfYvDeCAcEo2QYzCfpZoaEVJ8sbCT7hkxXY/:17372:0:99999:7:::

Adapted from Du, Wenliang. *Computer & Internet Security: A Hands-on Approach*. 2019.

Use case 3: Password verification

Password verification

- Do not store password in plaintext
- Store hashed version of password
- Use salt to defeat dictionary/rainbow tables
- To authenticate: enter password, hash it, compare with stored



Example

- In Linux passwords stored in /etc/shadow
- seed:6wDRrWCQz$IsBXp9.9wz9SGrF.nbihpoN5w.zQx02sht4c TY8qI7YKh00wN/sfYvDeCAcEo2QYzCfpZoaEVJ8sbCT7hkxXY/:17372:0:99999:7:::

Adapted from Du, Wenliang. *Computer & Internet Security: A Hands-on Approach*. 2019.

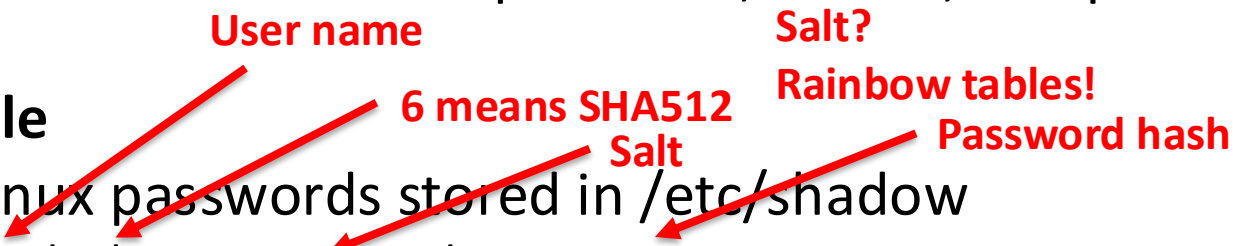
Use case 3: Password verification

Password verification

- Do not store password in plaintext
- Store hashed version of password
- Use salt to defeat dictionary/rainbow tables
- To authenticate: enter password, hash it, compare with stored

Example

- In Linux passwords stored in /etc/shadow
- seed:6wDRrWCQz$IsBXp9.9wz9SGrF.nbihpoN5w.zQx02sht4cTY8qI7YKh00wN/sfYvDeCAcEo2QYzCfpZoaEVJ8sbCT7hkxXY/:17372:0:99999:7:::

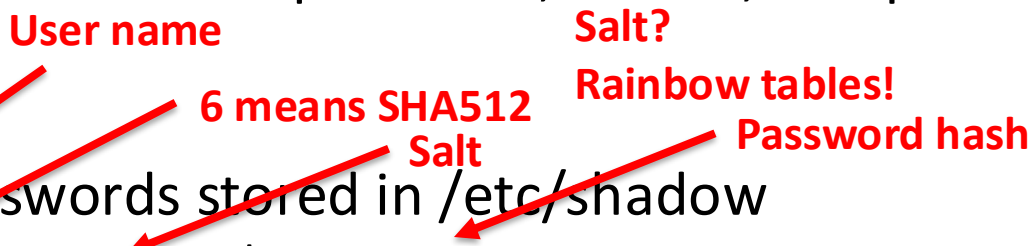


Adapted from Du, Wenliang. *Computer & Internet Security: A Hands-on Approach*. 2019.

Use case 3: Password verification

Password verification

- Do not store password in plaintext
- Store hashed version of password
- Use salt to defeat dictionary/rainbow tables
- To authenticate: enter password, hash it, compare with stored



Example

- In Linux passwords stored in /etc/shadow
- seed:6wDRrWCQz$IsBXp9.9wz9SGrF.nbihpoN5w.zQx02sht4cTY8qI7YKh00wN/sfYvDeCAcEo2QYzCfpZoaEVJ8sbCT7hkxXY/:17372:0:99999:7:::

```
$ sudo cat /etc/shadow #list all passwords on system (requires sudo)
…
seed:$6$wDRrWCQz$IsBXp9.9wz9SGrF.nbihpoN5w.zQx02sht4cTY8qI7YKh00wN/
```

Do not store passwords in plaintext!

Stored hashed passwords instead

Use case 4: Trusted timestamping

RFC 3161

Trusted timestamping

- Want to be able to prove a document existed at a point in time (called Long Term Validation – LTV)
- Also want non-repudiation (cannot later deny something, repudiate means to deny)

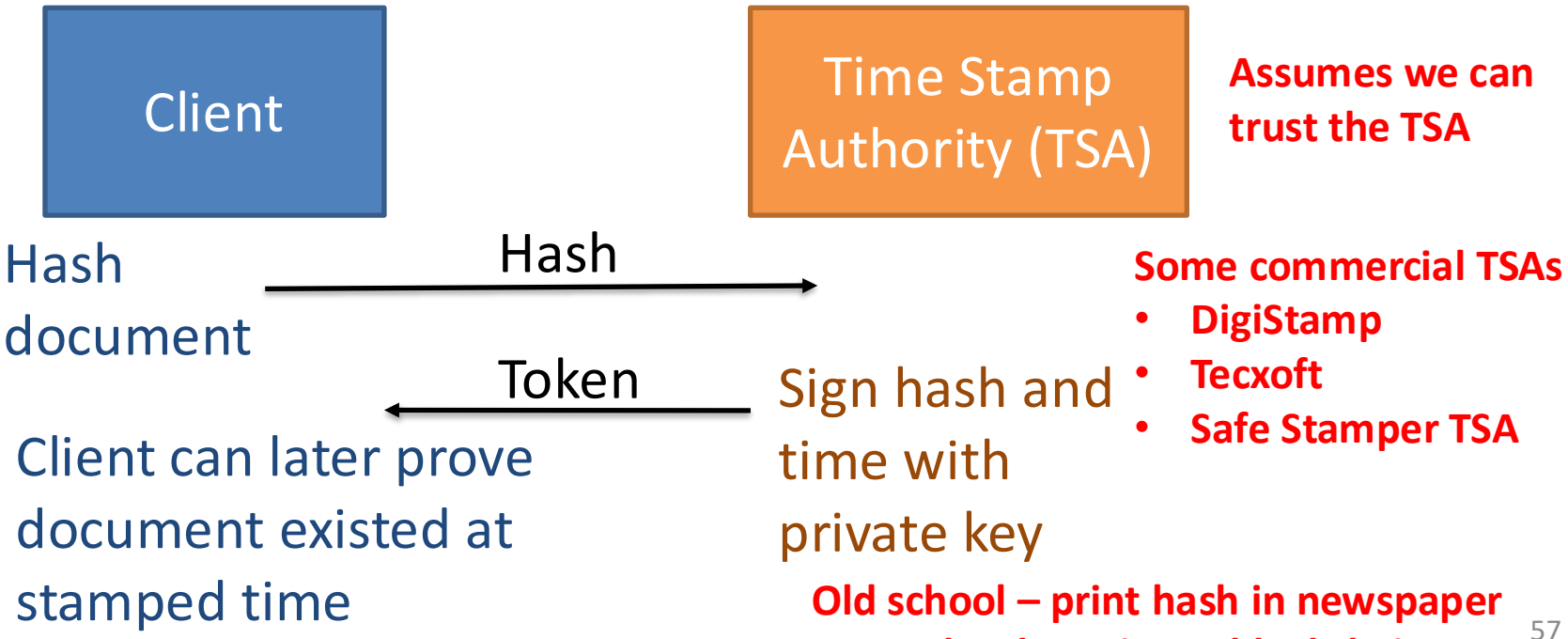
Adapted from: https://www.forensicnotes.com/timestamping-authority-tsa/

Use case 4: Trusted timestamping

RFC 3161

Trusted timestamping

- Want to be able to prove a document existed at a point in time (called Long Term Validation – LTV)
- Also want non-repudiation (cannot later deny something, repudiate means to deny)



Adapted from: https://www.forensicnotes.com/timestamping-authority-tsa/

Discussion

Scenario:

- You are pitching a start up idea to a sketchy venture capitalist
- You'd like to be able to prove you have produced your business plan *before* you meet with them
- What can you do?

  - **Could try to get an NDA, but reputable VCs won't sign one**
  - **Hash the business plan**
    - **Store the hash in bitcoin block chain (where it is timestamped)**
    - **If later you need to prove you had the idea, produce the business plan and have people check the hash written into the block chain**

Agenda

1. Encryption
   - Symmetric (AES)
   - Hashing (SHA-2)
   - Asymmetric (RSA)
2. TLS
3. TOR
4. VPN
5. Wi-Fi security
6. Supplemental crypto material

Asymmetric cryptography uses two keys, one public and one private

Symmetric key crypto:

- Requires sender and receiver to know the shared secret key
- Q: How to agree on key in first place (particularly if never "met")?

Asymmetric crypto:



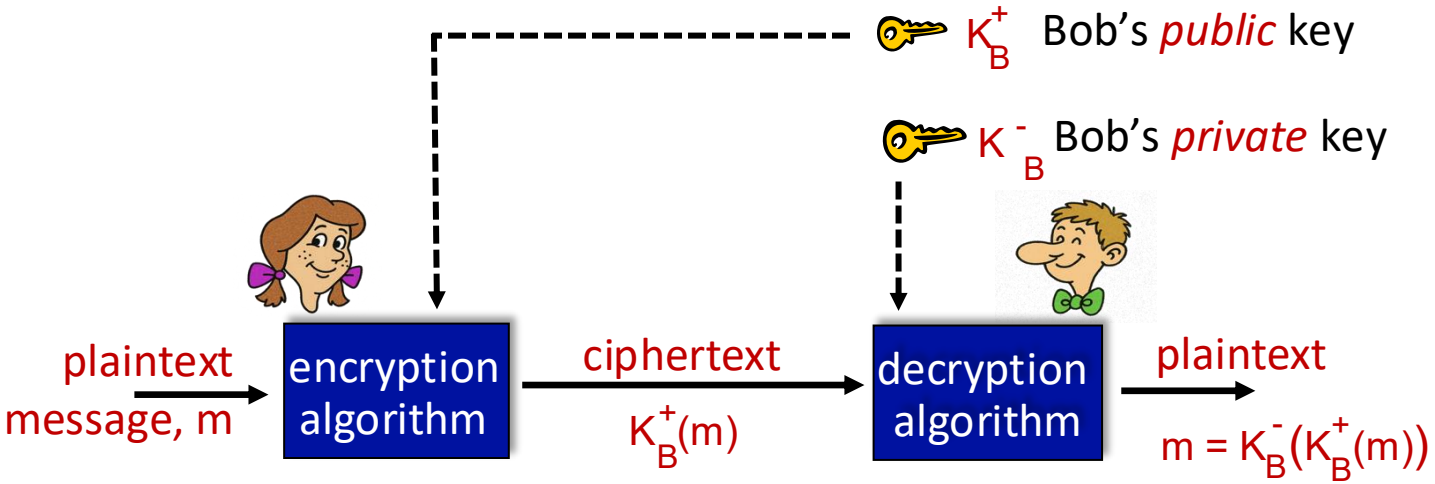
- *Radically* different approach [Diffie-Hellman76, RSA78]
- Sender, receiver do *not* share secret key
- *Public* encryption key known to *all*
- *Private* decryption key known only to receiver

Never reveal your *private* key

But you do reveal your *public* key

Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

Asymmetric cryptography uses two keys, one public and one private



Wow - public key cryptography revolutionized 2000-year-old (previously only symmetric key) cryptography!

- Similar ideas emerged at roughly same time, independently in US and UK (classified)

Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

RSA is a widely used implementation of asymmetric encryption

Requirements:

1. Need $K_B^+(\cdot)$ and $K_B^-(\cdot)$ such that

$$K_B^-(K_B^+(m)) = m$$

2. Given public key K_B^+, it should be impossible to compute private key K_B^-

RSA: Rivest, Shamir, Adleman algorithm is commonly used

See supplemental slides for details on how to compute key

K^+ public key
K^- private key

Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

Importantly, in RSA, order of encryption does not matter

The following property will be *very* useful later:

$$\underbrace{K_B^-(K_B^+(m))}_{\text{use public key first, followed by private key}} = m = \underbrace{K_B^+(K_B^-(m))}_{\text{use private key first, followed by public key}}$$

Result is the same!

Order of encryption doesn't matter

Encrypt with public key first, then encrypt with private key
same as
Encrypt with private key first, then encrypt with public key

Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

Asymmetric encryption can provide integrity, authentication, and confidentiality

Symmetric encryption can provide Confidentiality

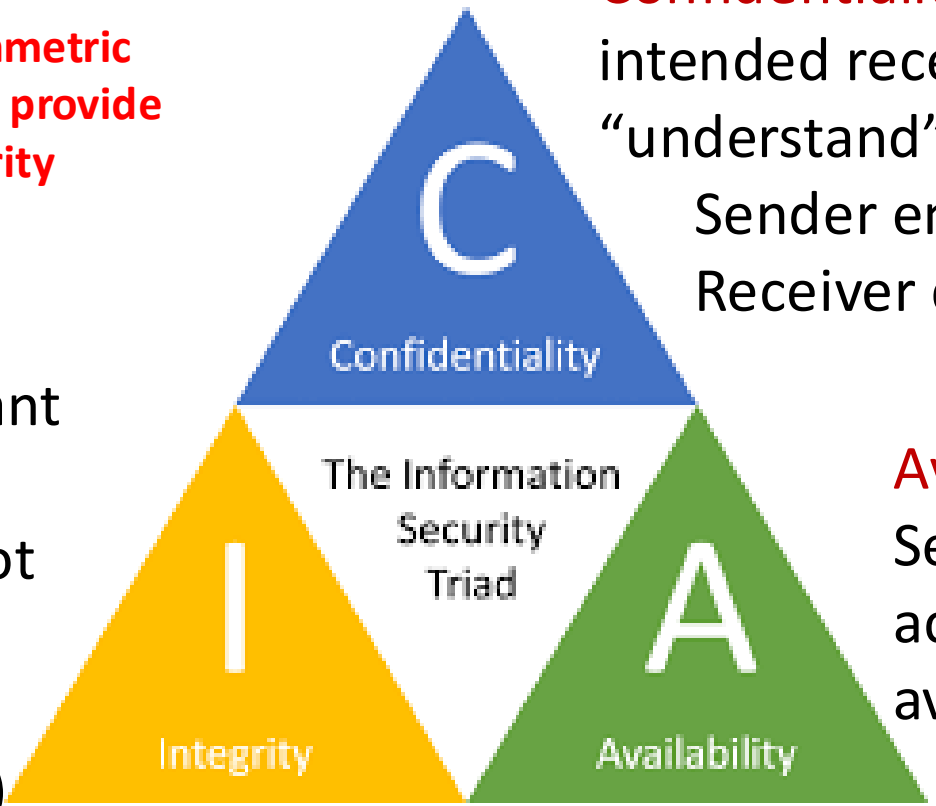
Confidentiality: Only sender and intended receiver should "understand" message contents

Sender encrypts message

Receiver decrypts message

Hash and asymmetric encryption can provide message Integrity

Integrity: Sender and receiver want to ensure message not altered (in transit, or afterwards) without detection



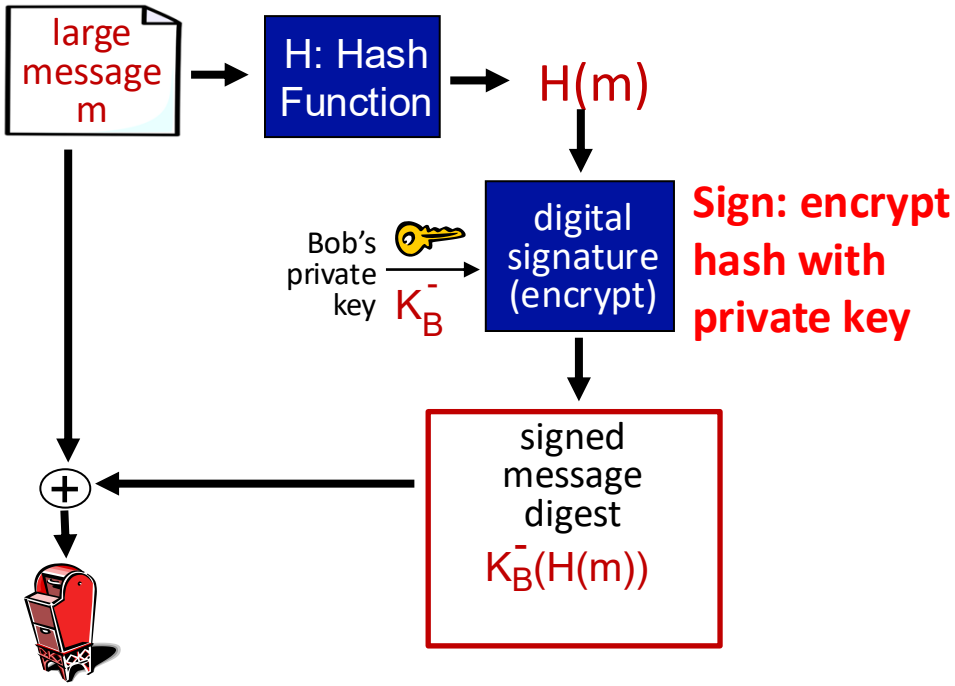
Availability: Services must be accessible and available to users

Authentication: Sender and receiver want to confirm identity of each other

Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

Data integrity: sign a hash to detect if any bits were altered

Bob sends digitally signed message:



Bob:

Hashes message m

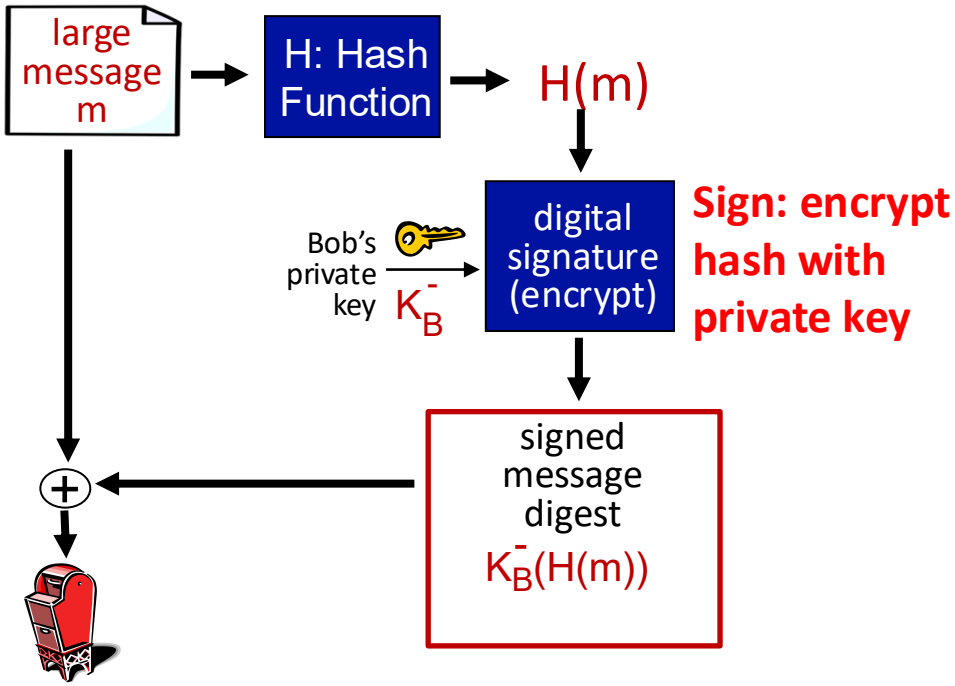
Encrypts hash with *private key*, $K_B^-(H(m))$ -> called a digital signature

Sends m in plaintext *and* encrypted hash to Alice

Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

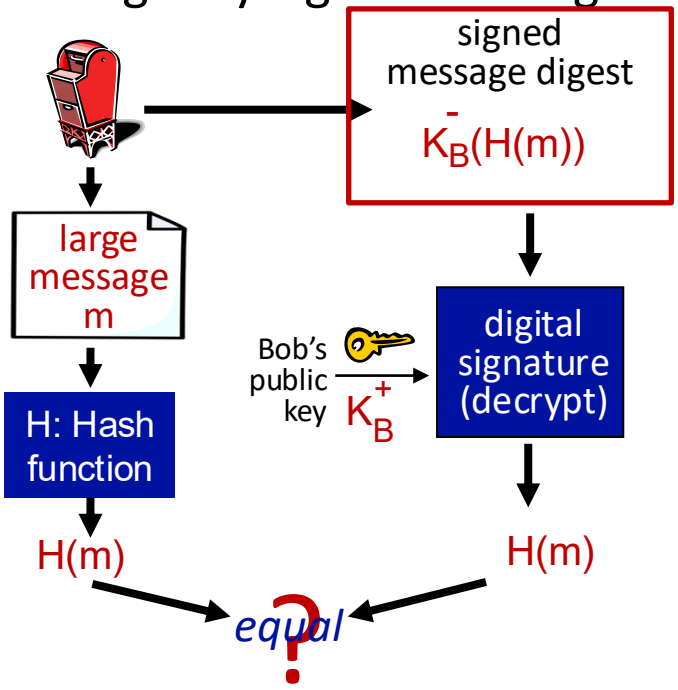
Data integrity: sign a hash to detect if any bits were altered

Bob sends digitally signed message:



Sign: encrypt hash with private key

Alice verifies signature, integrity of digitally signed message:



Alice:

Receives plaintext m and Bob's digital signature

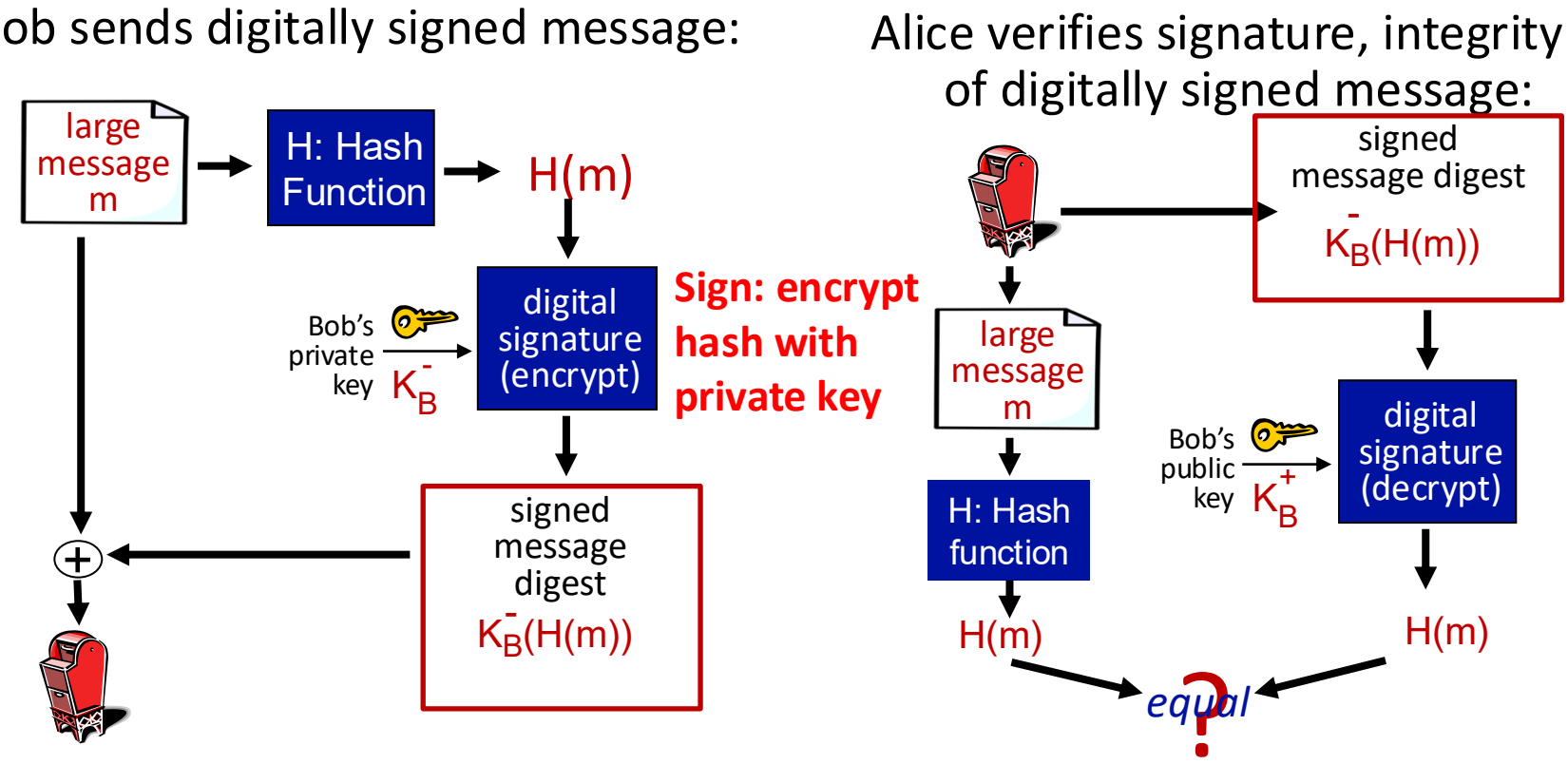
Encrypts signature with Bob's public key (reverses private key encryption) to get H(m)

Hashes plaintext message m to get H(m)

Compares both H(m)'s to see if equal

Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

Data integrity: sign a hash to detect if any bits were altered



If hashes are equal, Alice knows:

Authentication: the message MUST have come from someone with Bob's private key, but was it really Bob? Can't be sure just yet…

Integrity: the message was not altered

Confidentiality? Not yet! Adversary can see plain text message *m*

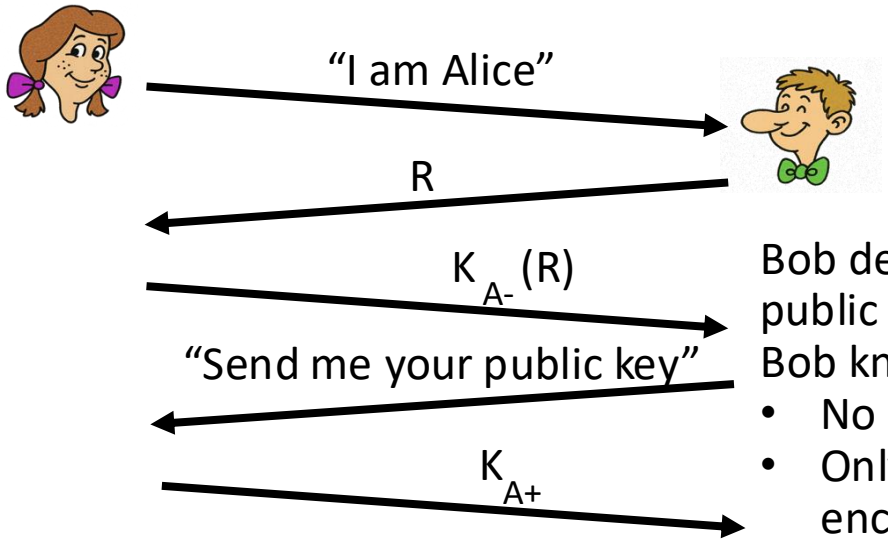
Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

Even with asymmetric keys we cannot ensure authentication

To prevent replay attacks, Bob sends Alice nonce, R

- Alice must return R, encrypted with her private key

nonce: number (R) used only once-in-a-lifetime



Bob decrypts K_{A-} with Alice's public key

Bob knows:

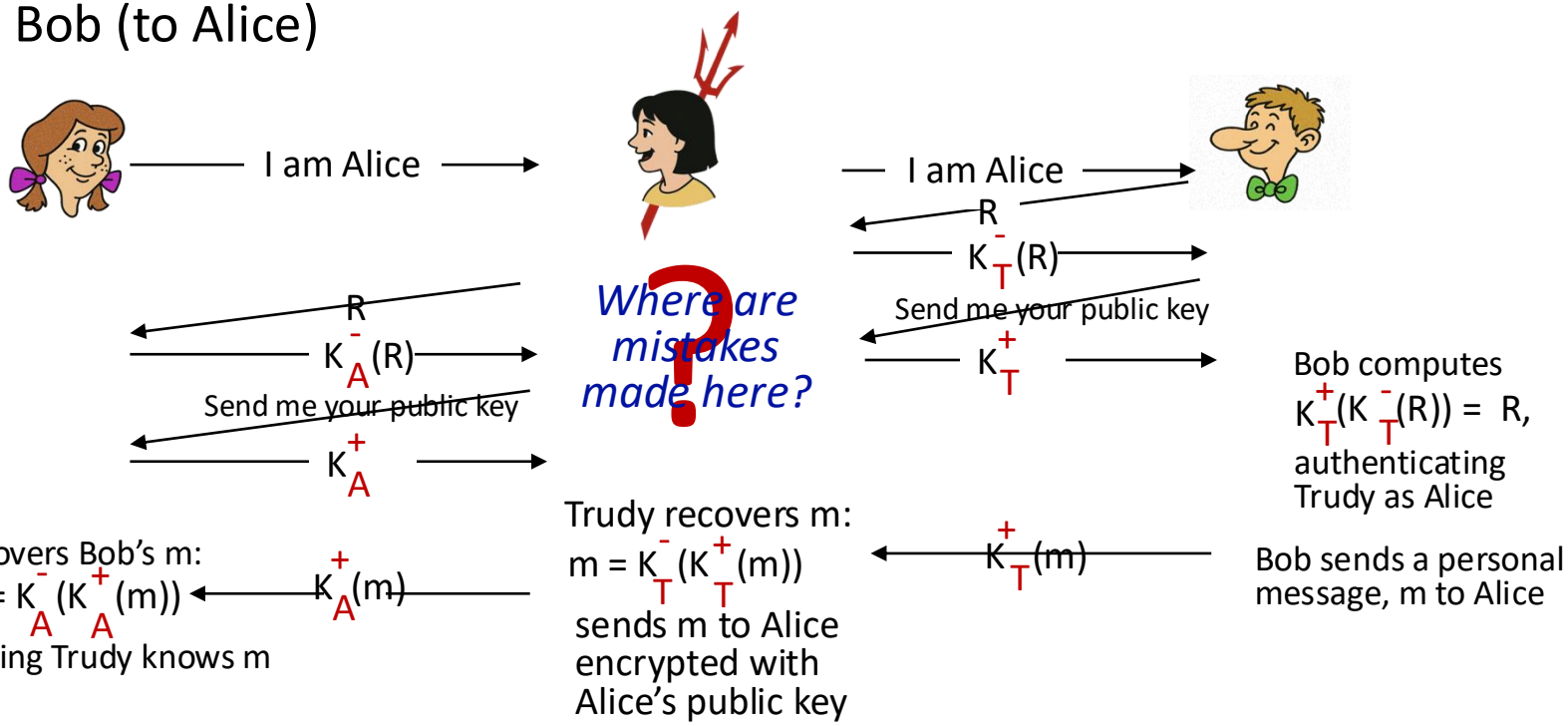
- No play back (nonce)
- Only Alice knows key to encrypt nonce, so it must be Alice!

Looks good, but are we sure we are really talking to Alice?

Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

Even with asymmetric keys we cannot ensure authentication

Man (or woman) in the middle attack: Trudy poses as Alice (to Bob) and as Bob (to Alice)



We have not solved authentication yet!

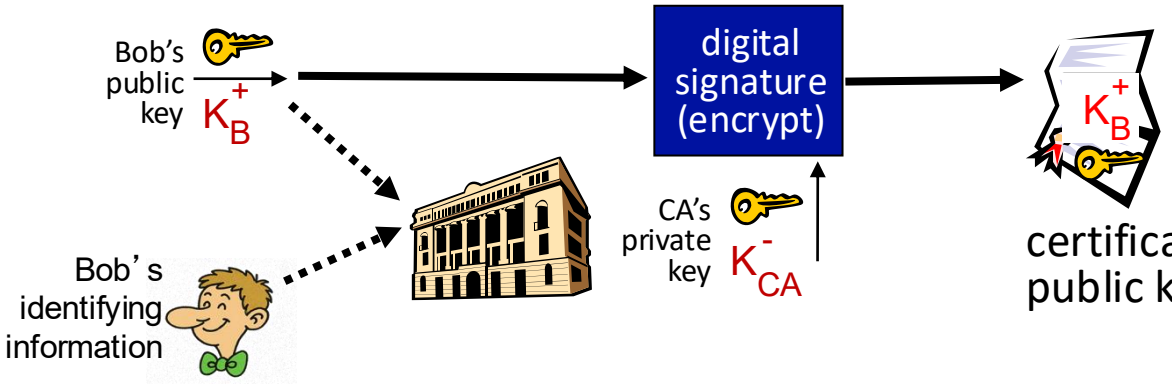
Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

Public key Certification Authorities (CA) can ensure authentication

- Certification authority (CA): binds public key to particular entity, E
- Entity (person, website, router) registers its public key and provides "proof of identity" to CA
  - CA creates certificate binding identity E to E's public key
  - Certificate containing E's public key digitally signed by CA: CA says "this is E's public key"

CA sees the file appear on the server

CA certifies Bob's public key by signing it with CA's private key



If Bob sends public key to anyone

Anyone can check it matches the CA's

certificate for Bob's public key, signed by CA

Use CA's public key to recover Bob's public key from CA signature

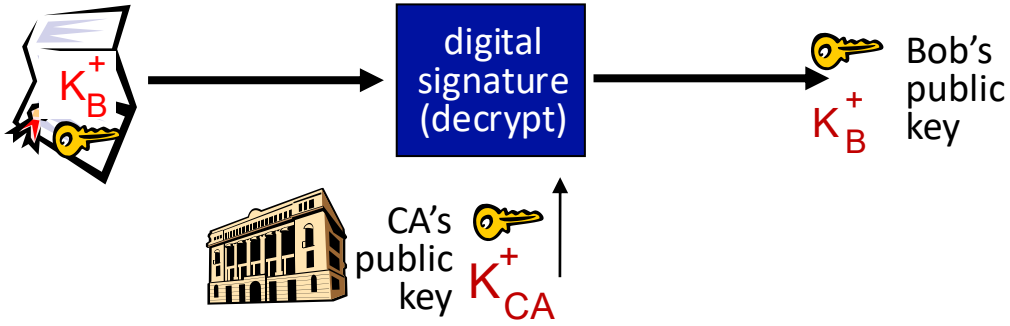
Bob must prove his identity to CA

Server: CA asks Bob to put a file on the server

Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

Public key Certification Authorities (CA) can ensure authentication

- When Alice wants Bob's public key:
  - Gets Bob's certificate (from Bob or elsewhere)
  - Apply CA's public key to Bob's certificate, get Bob's public key



Authentication: If Alice trusts the CA, and if Bob's key matches decrypted CA key, she can be sure she is communicating with Bob and not an imposter!

Bob never revealed his private key!

Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

In practice: create session keys and then uses symmetric encryption

- Asymmetric encryption is computationally intensive
- DES (symmetric) is at least 100 times faster than RSA (asymmetric)
- Use public key crypto to establish secure connection, then establish second key – symmetric session key – for encrypting data

Session key, K_S

- Bob and Alice use asymmetric encryption with a CA to exchange a symmetric session key K_S
- Once both have K_S, they use symmetric key cryptography (AES) to protect communications

Confidentiality – symmetric encryption from derived session keys ensures no one can read messages
Authentication – each side knows they are talking to the real other side (not MiTM) thanks to the CA's signature on the public key certificate (if you trust the CA at least)
Integrity – can use MACs to know the message was not altered

Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

Agenda

1. Encryption
   - Symmetric (AES)
   - Hashing (SHA-2)
   - Asymmetric (RSA)
2. TLS
3. TOR
4. VPN
5. Wi-Fi security
6. Supplemental crypto material

Transport-Layer Security (TLS)

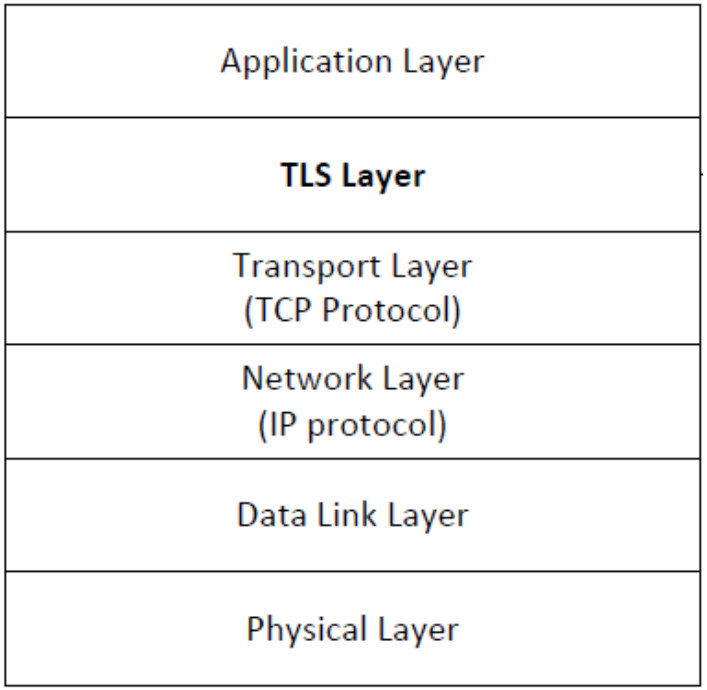
- Widely deployed security protocol above the transport layer
  - Supported by almost all browsers, web servers: https (port 443)
- Provides:
  - Confidentiality: via *symmetric encryption*
  - Integrity: via *cryptographic hashing*
  - Authentication: via *public key cryptography*
- History:
  - Early research, implementation: secure network programming, secure sockets
  - Secure socket layer (SSL) deprecated [2015]
  - TLS 1.3: RFC 8446 [2018]

You will still hear people say SSL when they mean TLS!

Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

TLS sits between the Transport and Application layers

- Unprotected data is given to TLS by Application layer
- TLS handles encryption, decryption and integrity checks
- TLS gives protected data to Transport layer



Are IP and MAC addresses hidden?
No!
Why?
Needed for routing and switching

Is Layer 7 message hidden?
Yes!

Adapted from *Computer & Internet Security* by Du

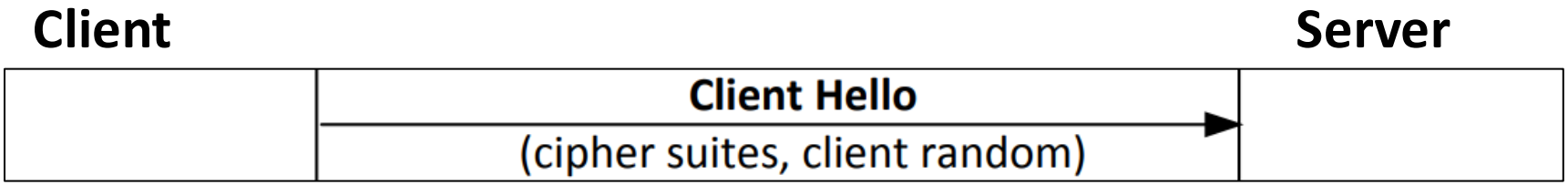
TLS involves a handshake between client and server to agree upon parameters

Before a client and server can communicate securely, several things need to be set up first:

- Encryption algorithm and key
- MAC algorithm (Message Authentication Code)
- Algorithm for key exchange

These cryptographic parameters need to be agreed upon by both the client and the server, otherwise connection is refused

TLS involves a handshake between client and server to agree upon parameters

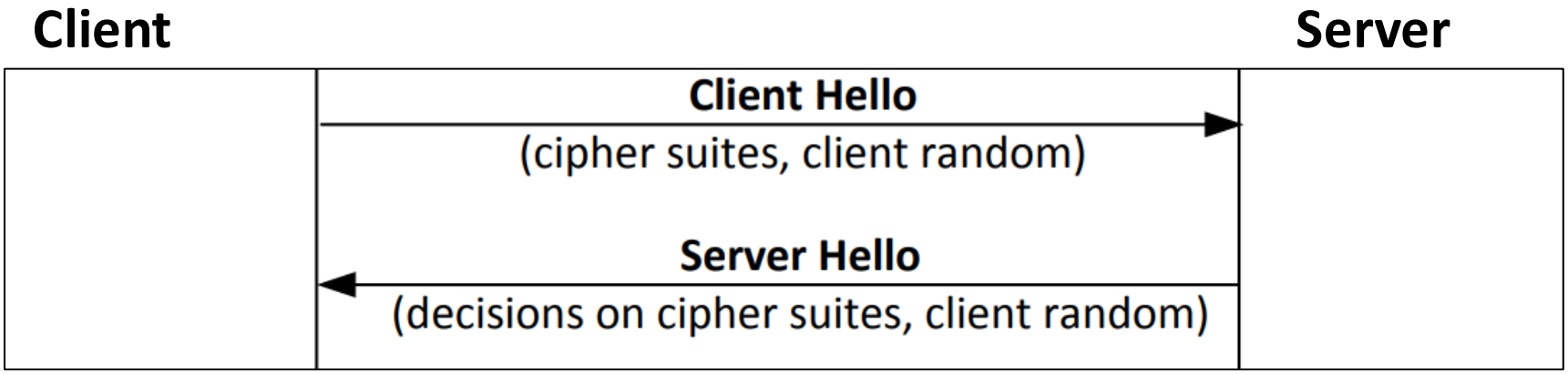


Client sends "Client Hello" message to server with:

- **List of ciphers that it can use (e.g., AES)**
- **Random nonce (to prevent replay attacks)**
- **Max TLS version it can support (e.g., TLS 1.2, 1.3)**

Adapted from https://www.youtube.com/watch?v=86cQJ0MMses

TLS involves a handshake between client and server to agree upon parameters

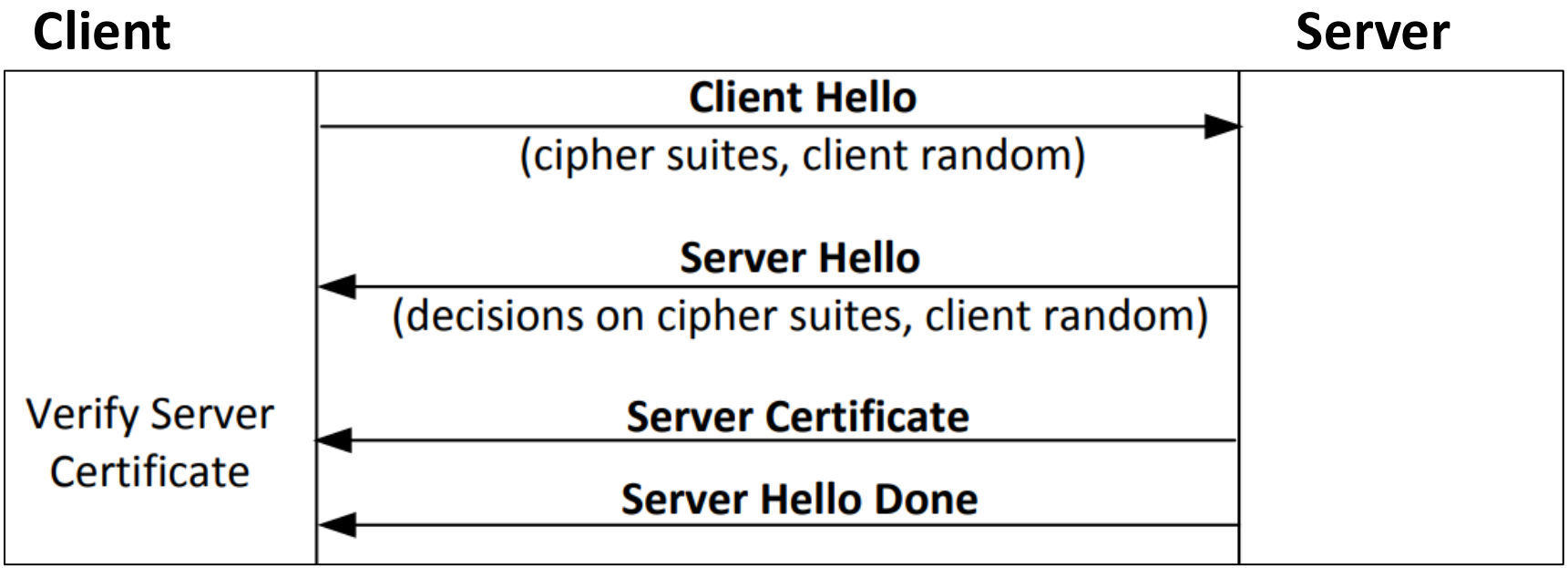


Server responds with "Server Hello" message to client with:

- **A decision on what cipher to use**
- **Random nonce (to prevent replay attacks)**
- **TLS version to use**

Adapted from https://www.youtube.com/watch?v=86cQJ0MMses

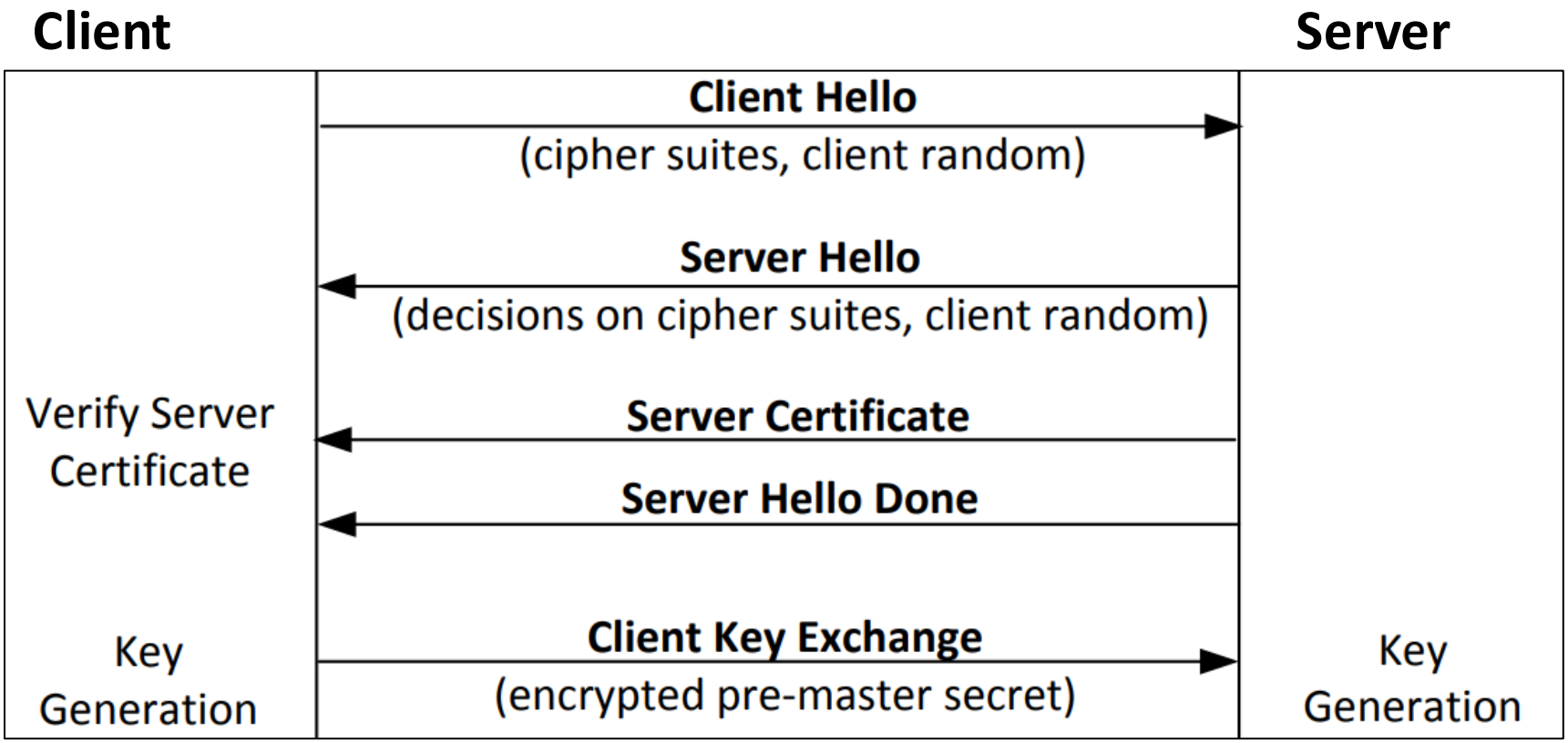
TLS involves a handshake between client and server to agree upon parameters



Server sends its certificate (includes public key signed by CA)
Client verifies certificate to prevent MiTM attacks
Client now knows server is the intended server
Hello Done indicates the first portion of handshake is complete
In some use cases, the client also sends a certification to server (typically not done on the web)

Adapted from https://www.youtube.com/watch?v=86cQJ0MMses

TLS involves a handshake between client and server to agree upon parameters

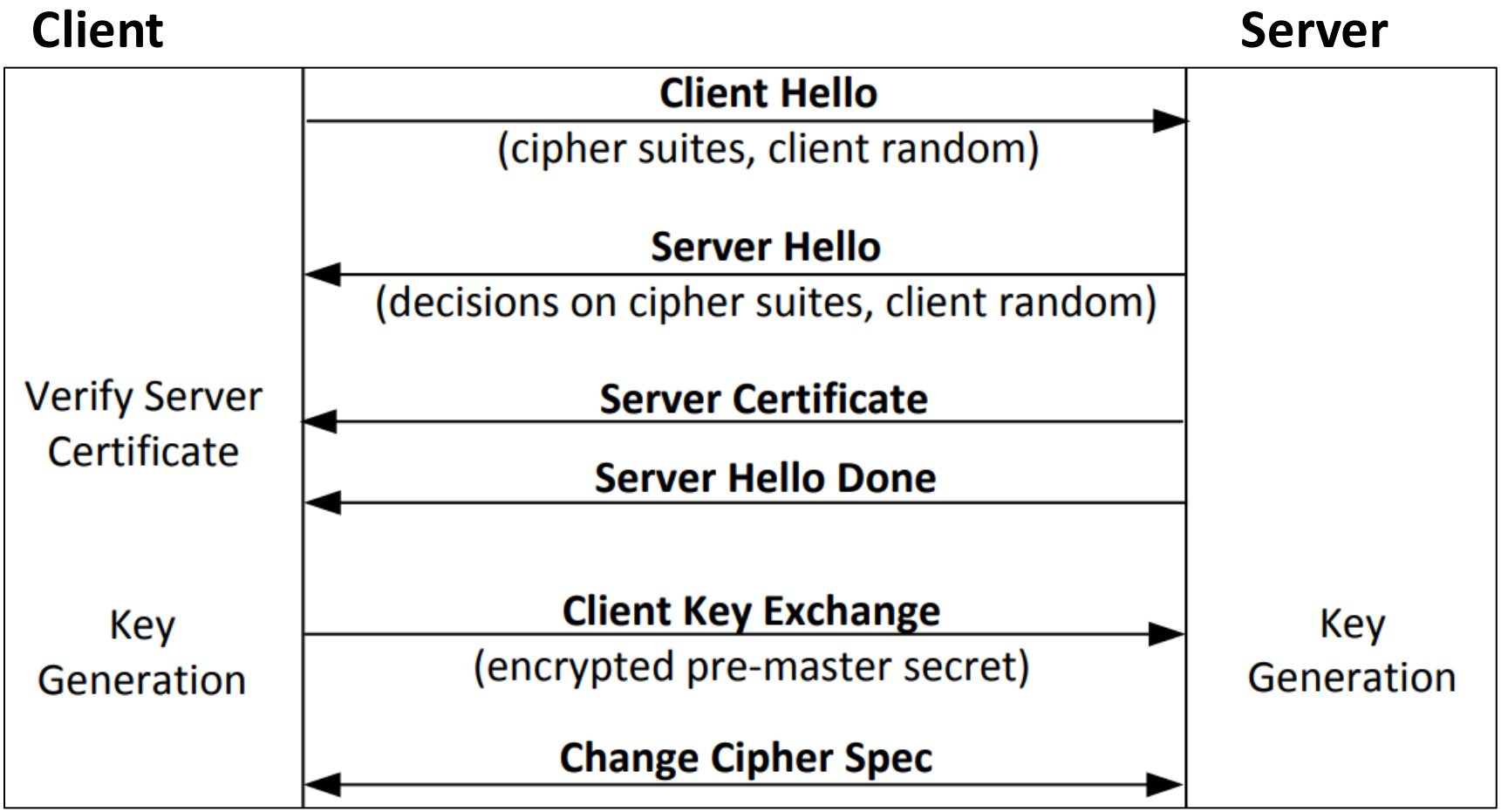


Client creates pre-master secret

- **Produces a random number to serve as the pre-master secret**
- **Encrypts random number with server's public key and sends to server**

Adapted from https://www.youtube.com/watch?v=86cQJ0MMses

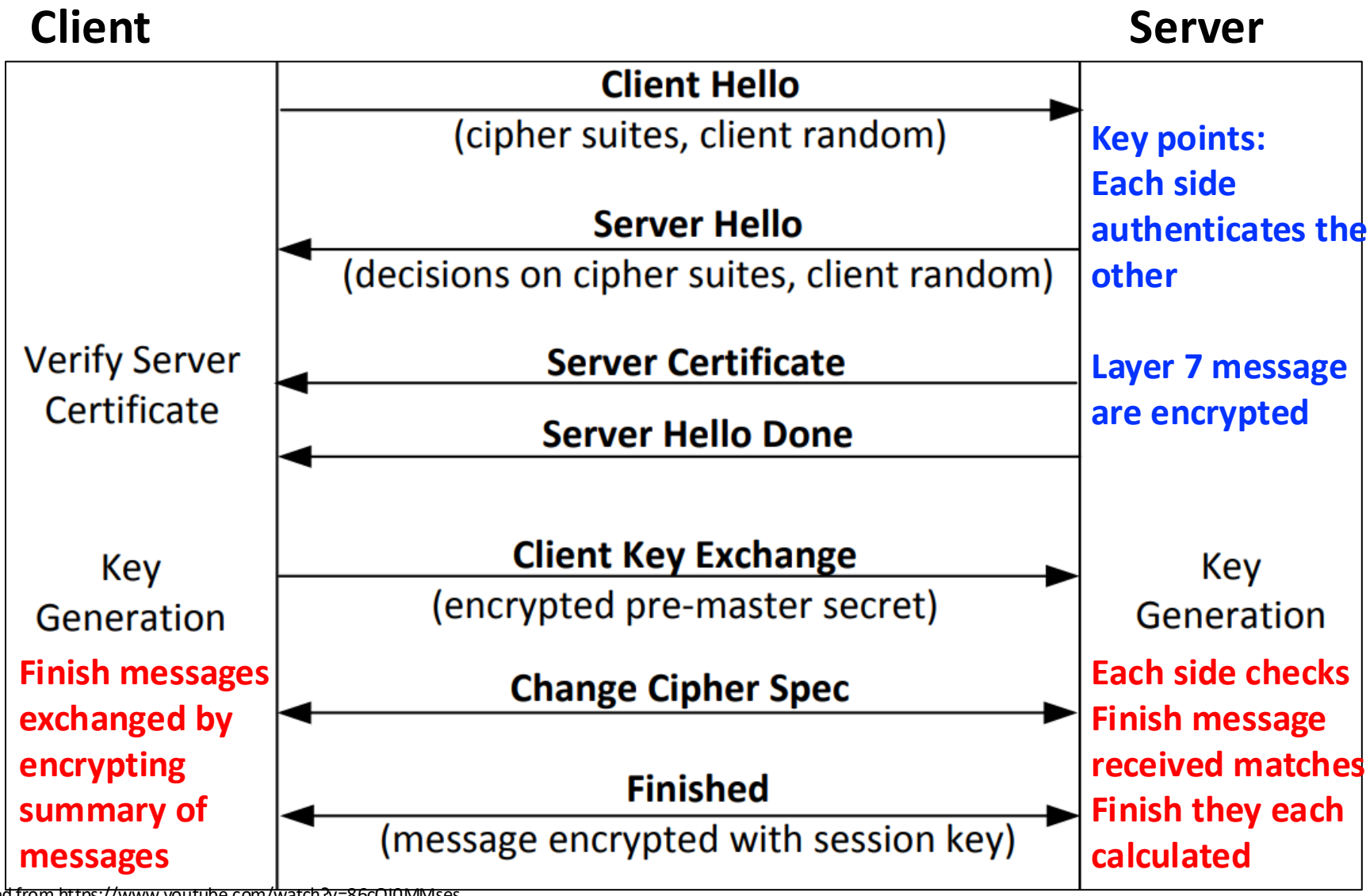
TLS involves a handshake between client and server to agree upon parameters



Server and client create 48-byte master secret by combining pre-master secret with nonces

Adapted from https://www.youtube.com/watch?v=86cQJ0MMses

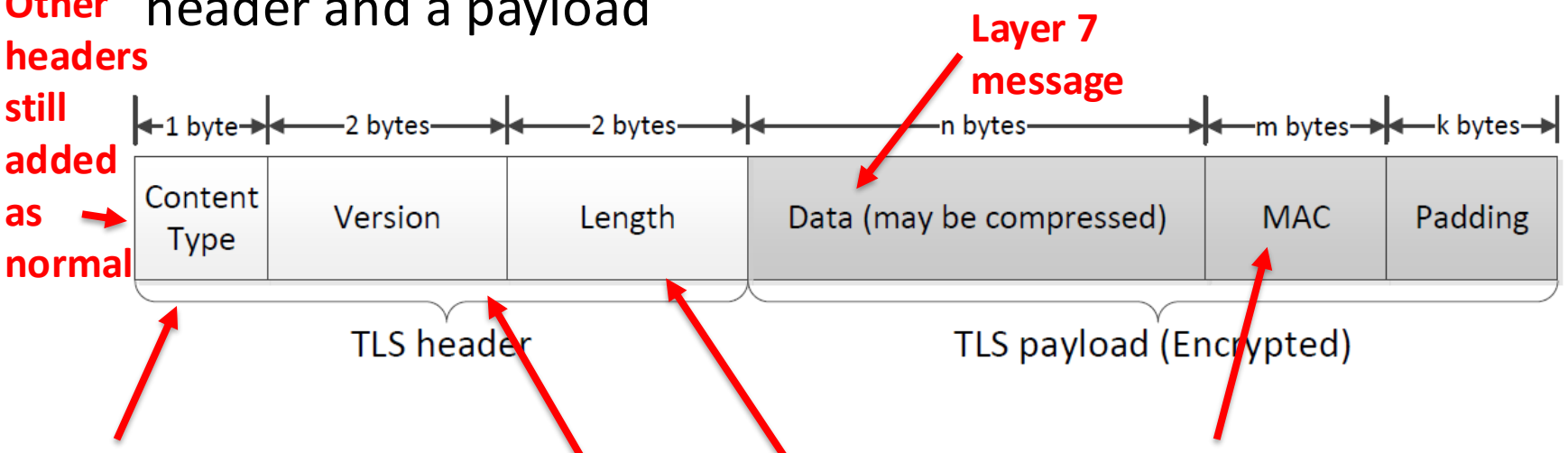
TLS involves a handshake between client and server to agree upon parameters



Adapted from https://www.youtube.com/watch?v=86cQJ0MMses

After handshake, client and server exchange encrypted data using records

Data is transferred using records, each record contains a header and a payload



Indicates the type of data carried

- Alert
- Application data
- Heartbeat
- Change Cipher Spec
- 0x17 = application data

Which TLS version to use

- 1.2
- 1.3

How many bytes are in the payload

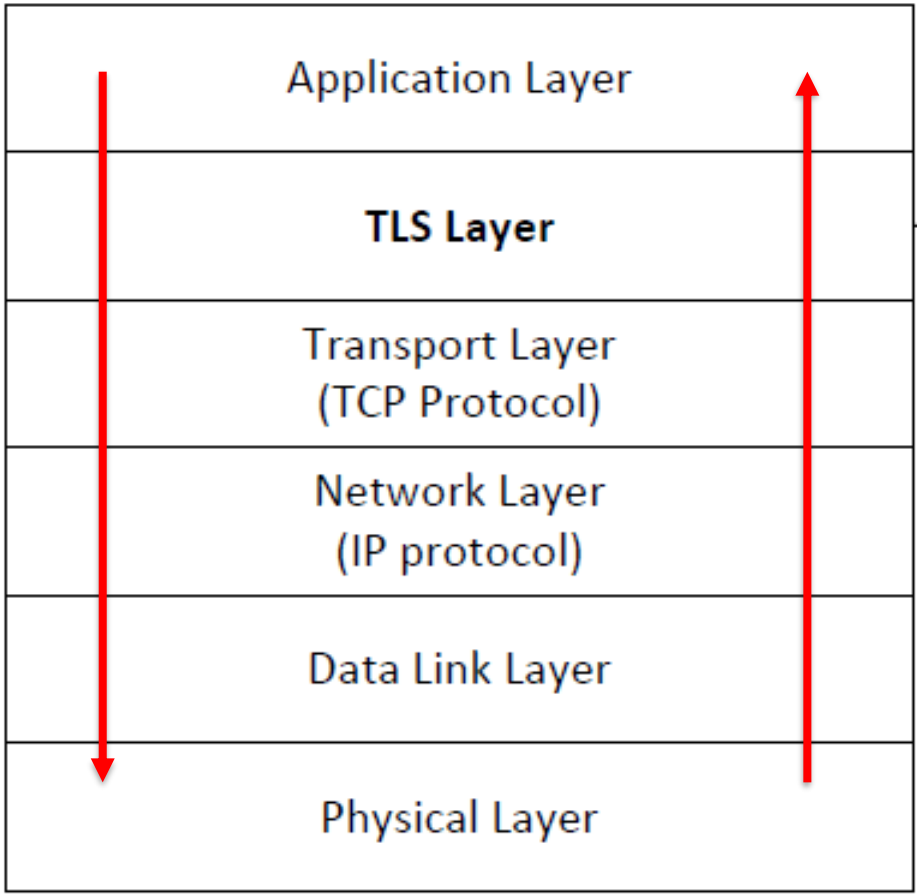
Message Authentication Code (MAC)

- Sender hashes data and secret key
- Sender includes MAC with data
- Receiver computes MAC on data and secret key
- Compares with received MAC
- Detects data was modified if MACs don't match (Integrity)

Remember TLS sits between the Application and Transport Layers

Sender

- Takes data from Application Layer
- Adds header and encrypts payload
- Sends record to Transport Layer and down stack



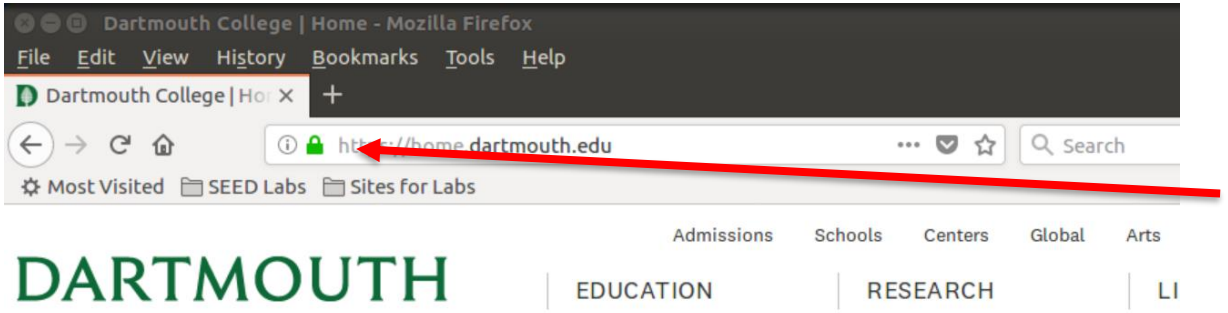
Receiver

- Takes data from TCP Layer
- Strips header and decrypts
- Sends to Application Layer

Transport, Network, and Link Layer headers are visible to adversary, but application data is encrypted

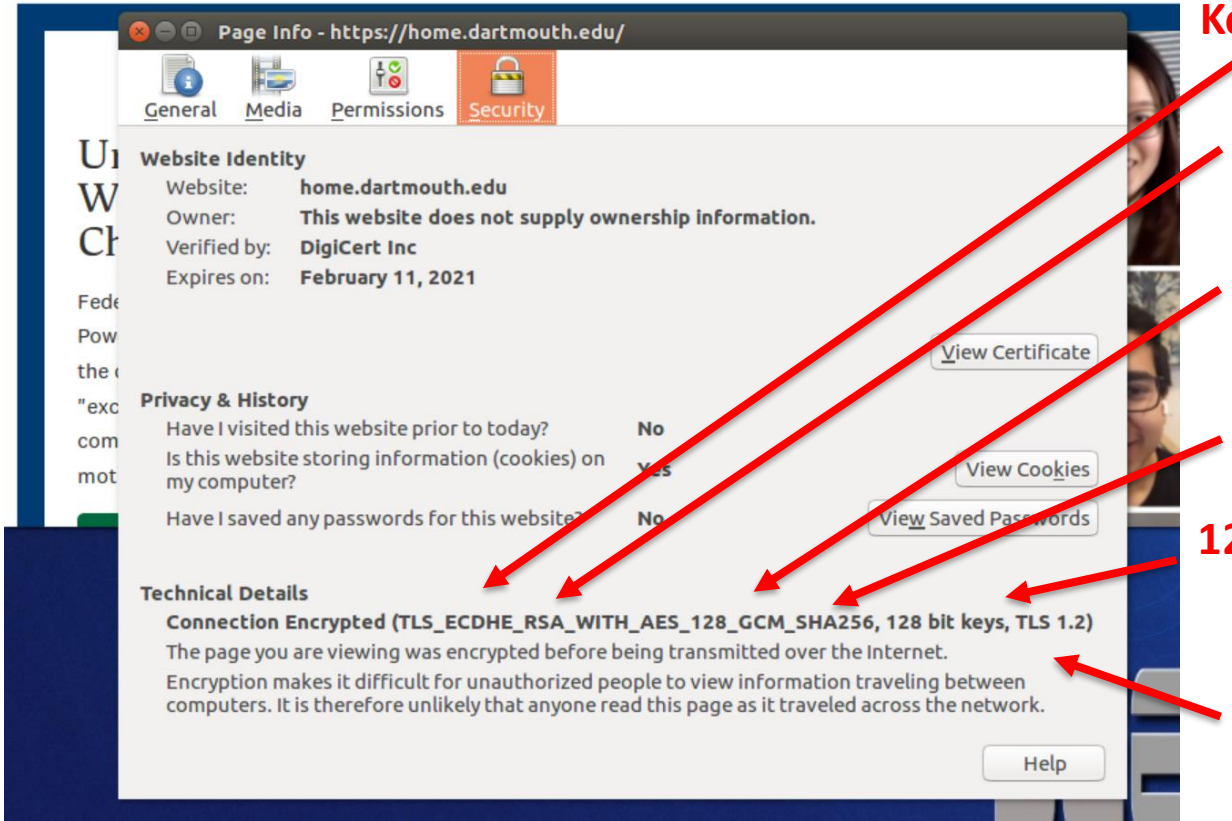
MAC needed for switching
IP needed for routing

You can see the details in your browser



In Firefox

- Click on lock in URL bar
- Select More Information



Key exchange uses ECDHE

RSA for public key authentication of certificates

128-bit AES encryption using GCM

SHA256 for hashing

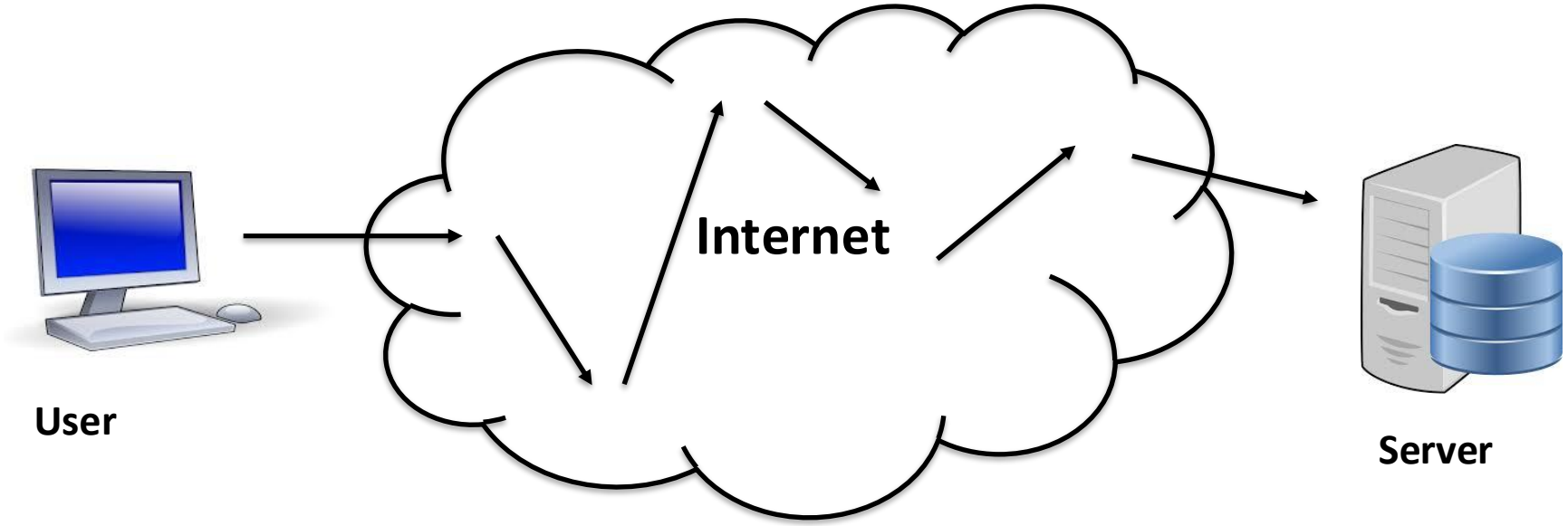
128-bit keys

TLS version 1.2

Agenda

1. Encryption
   - Symmetric (AES)
   - Hashing (SHA-2)
   - Asymmetric (RSA)
2. TLS
3. TOR
4. VPN
5. Wi-Fi security
6. Supplemental crypto material

When you logon to a server, anyone sniffing along the way can see the traffic



What types of information can an adversary learn?
Can source and destination be encrypted?

The Onion Router (TOR) obscures a message *and* the source and destination



User

User wants to access server, but doesn't want anyone to know they are accessing that server

TOR Node | TOR Node | TOR Node | TOR Node

TOR Node | TOR Node | TOR Node | TOR Node

TOR Node | TOR Node | TOR Node | TOR Node

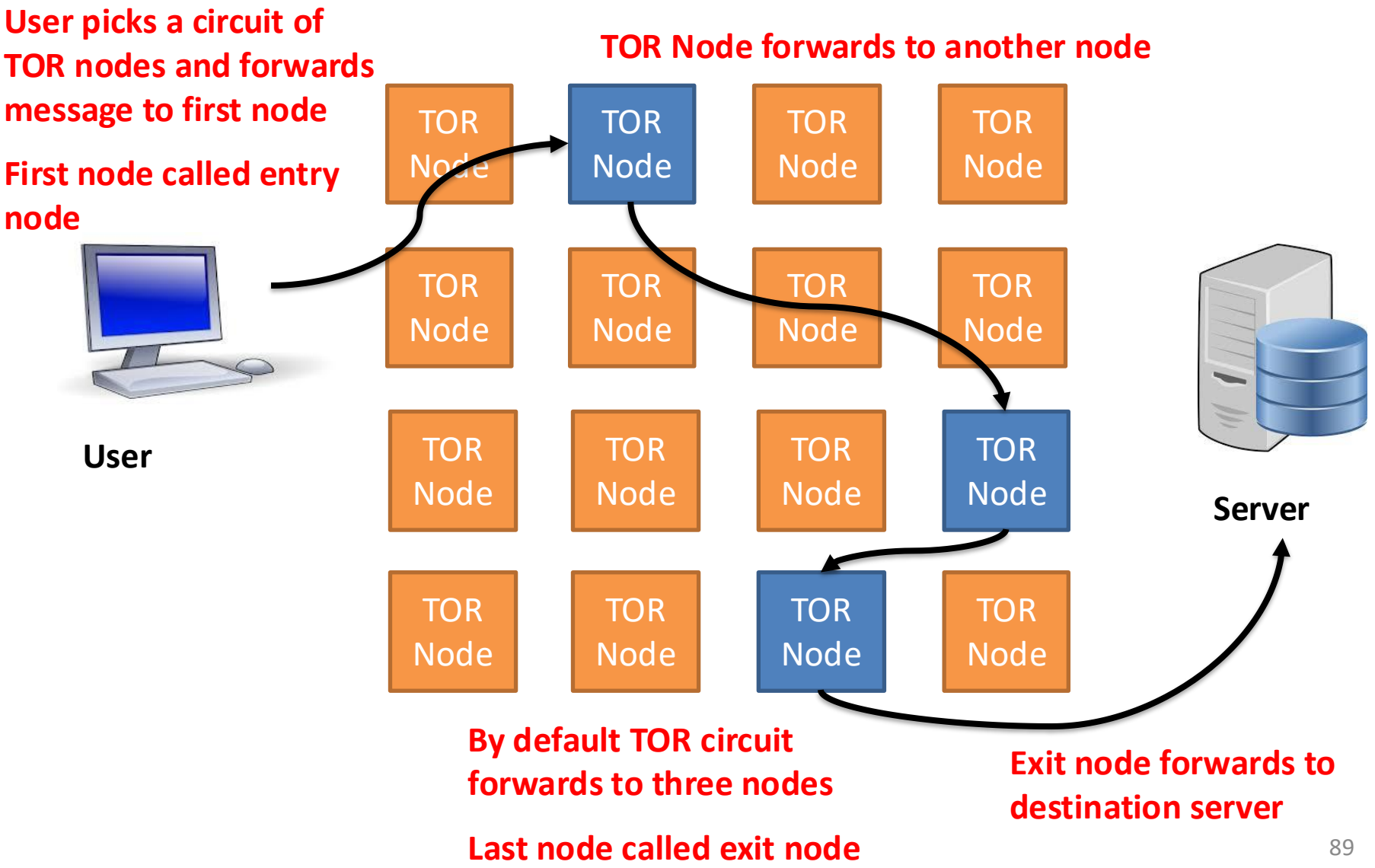
TOR Node | TOR Node | TOR Node | TOR Node



Server

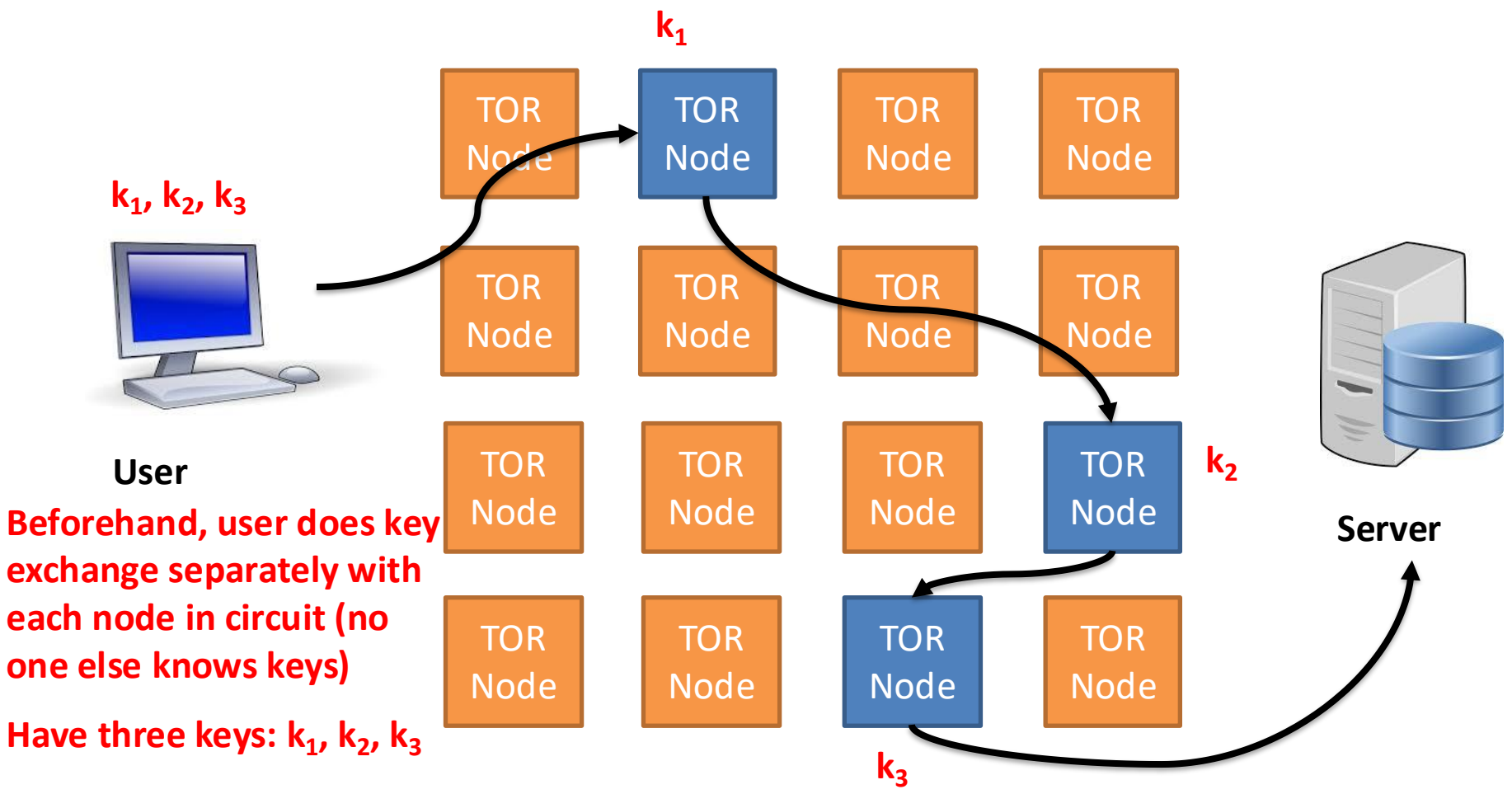
Adapted from https://www.youtube.com/watch?v=QRYzre4bf7I

The Onion Router (TOR) obscures a message *and* the source and destination



Adapted from https://www.youtube.com/watch?v=QRYzre4bf7I

The Onion Router (TOR) obscures a message *and* the source and destination

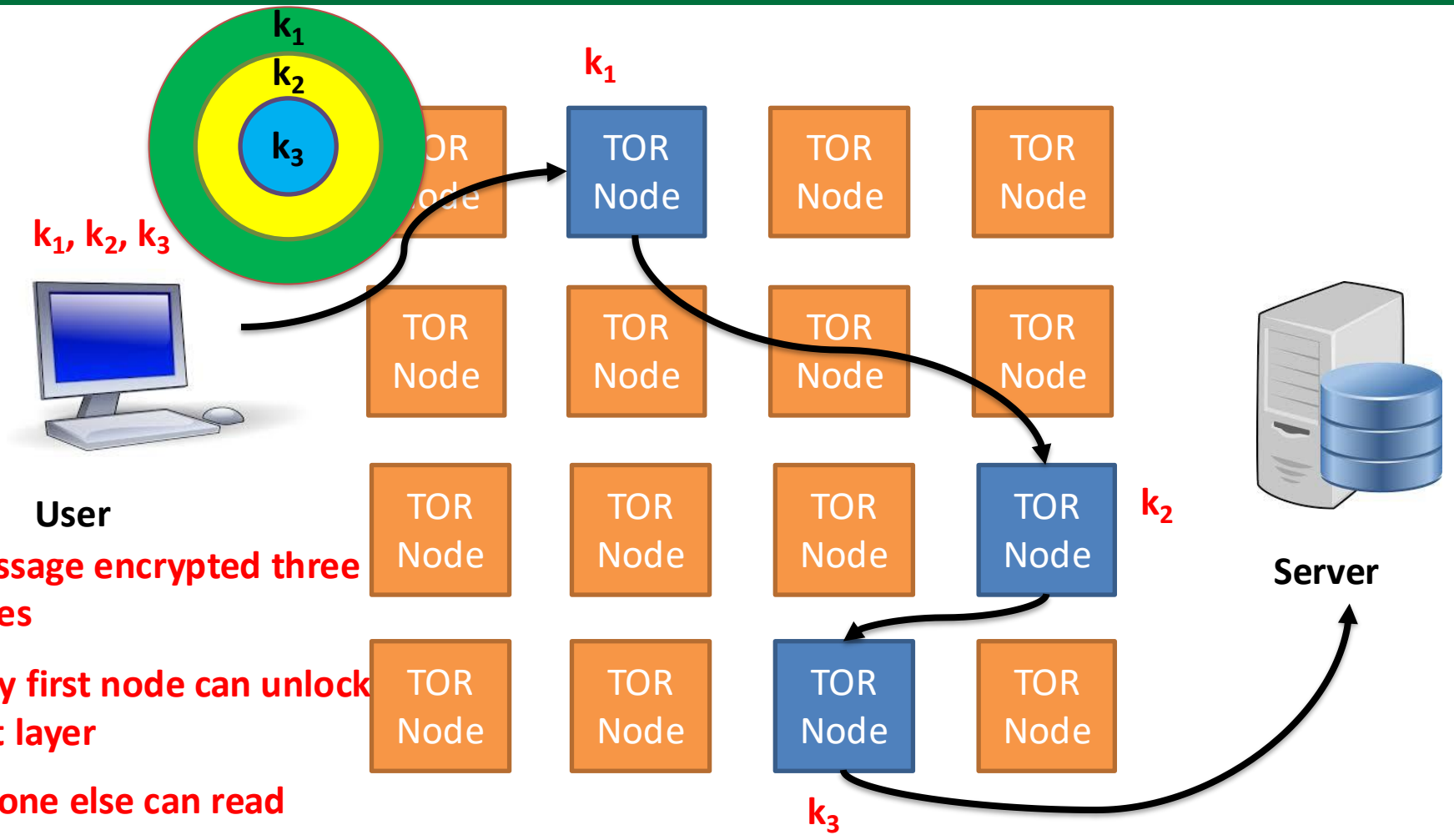


Beforehand, user does key exchange separately with each node in circuit (no one else knows keys)

Have three keys: k_1, k_2, k_3

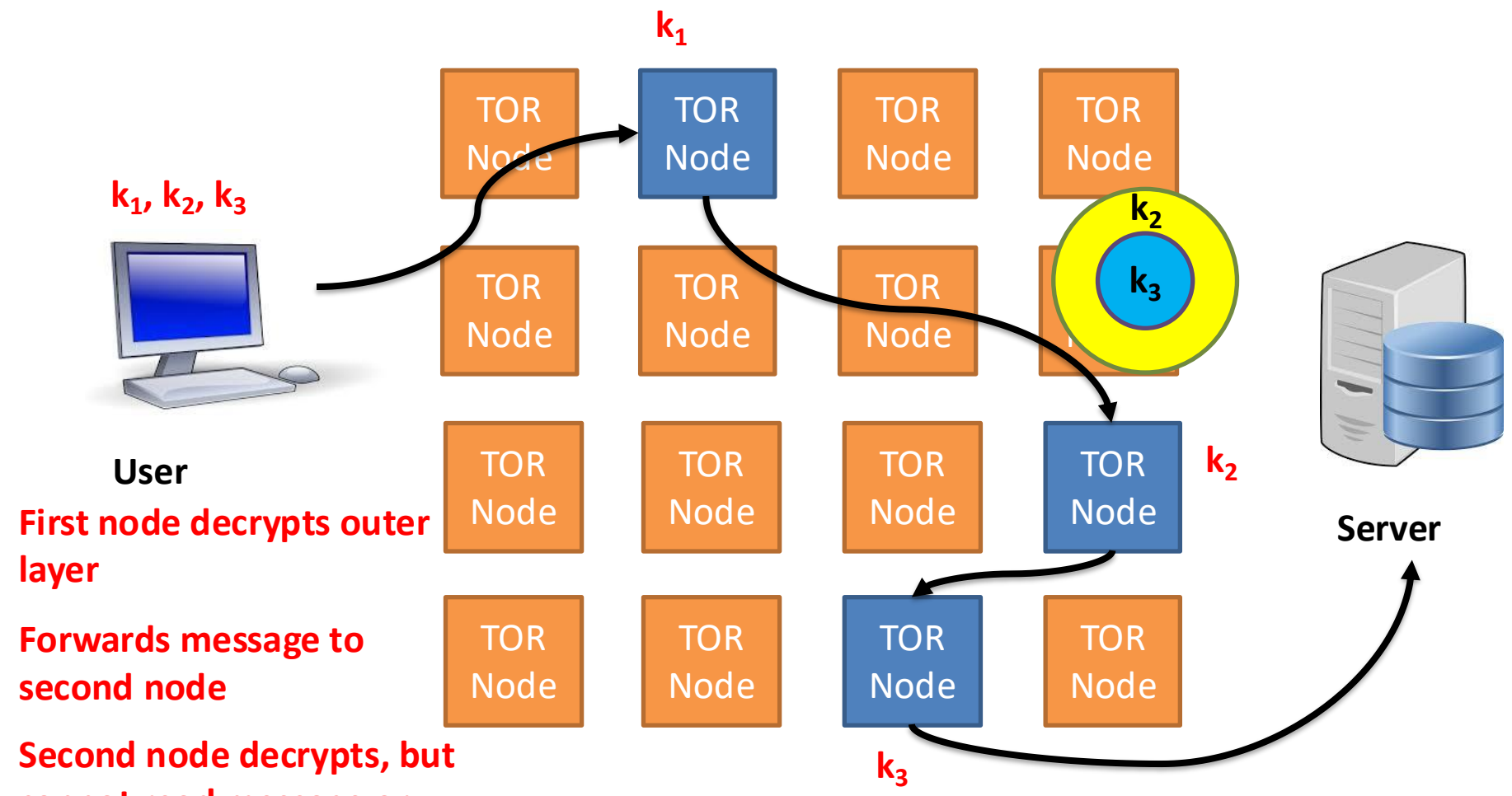
Adapted from https://www.youtube.com/watch?v=QRYzre4bf7I

The Onion Router (TOR) obscures a message *and* the source and destination

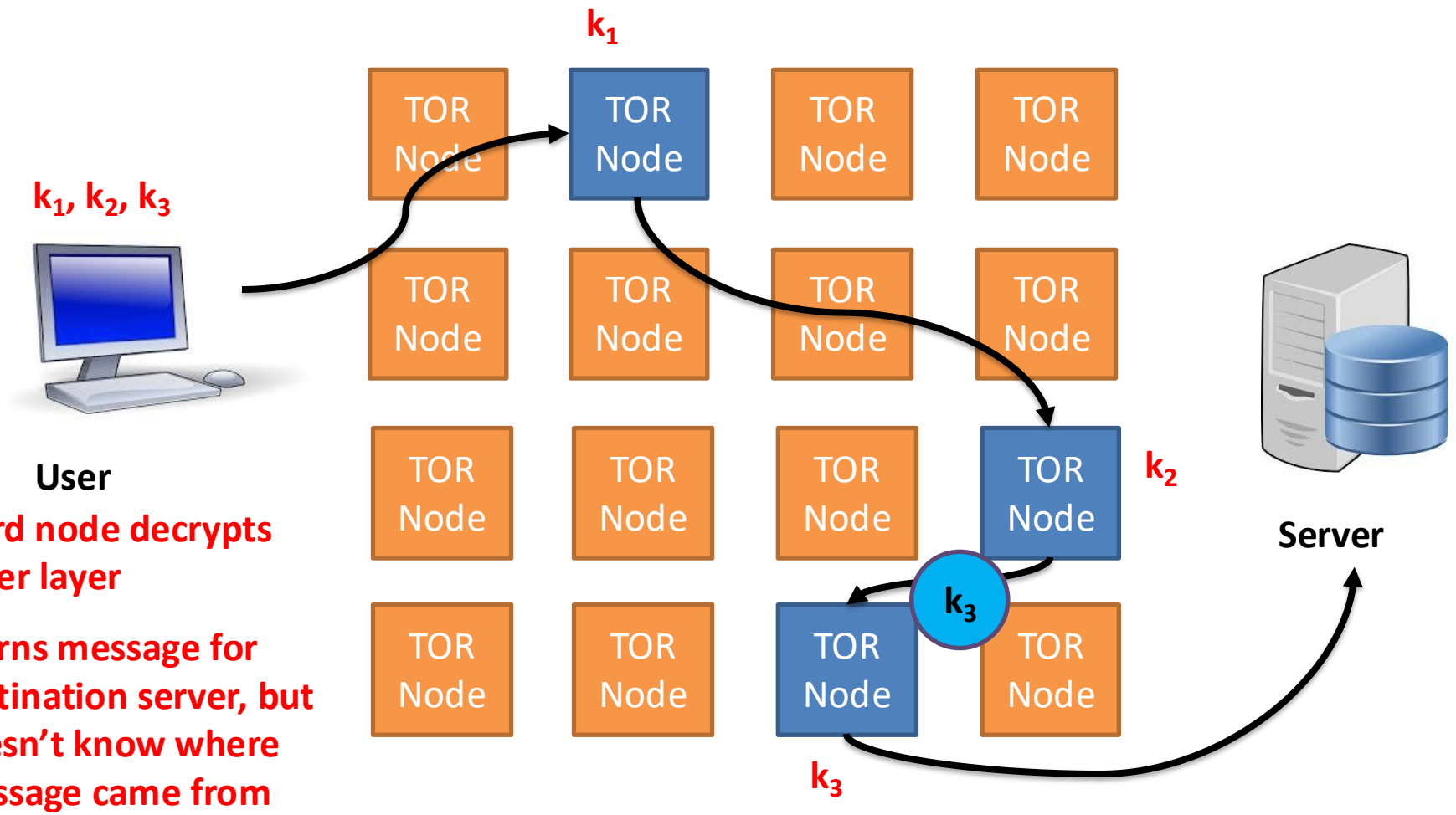


Adapted from https://www.youtube.com/watch?v=QRYzre4bf7I

The Onion Router (TOR) obscures a message *and* the source and destination



The Onion Router (TOR) obscures a message *and* the source and destination

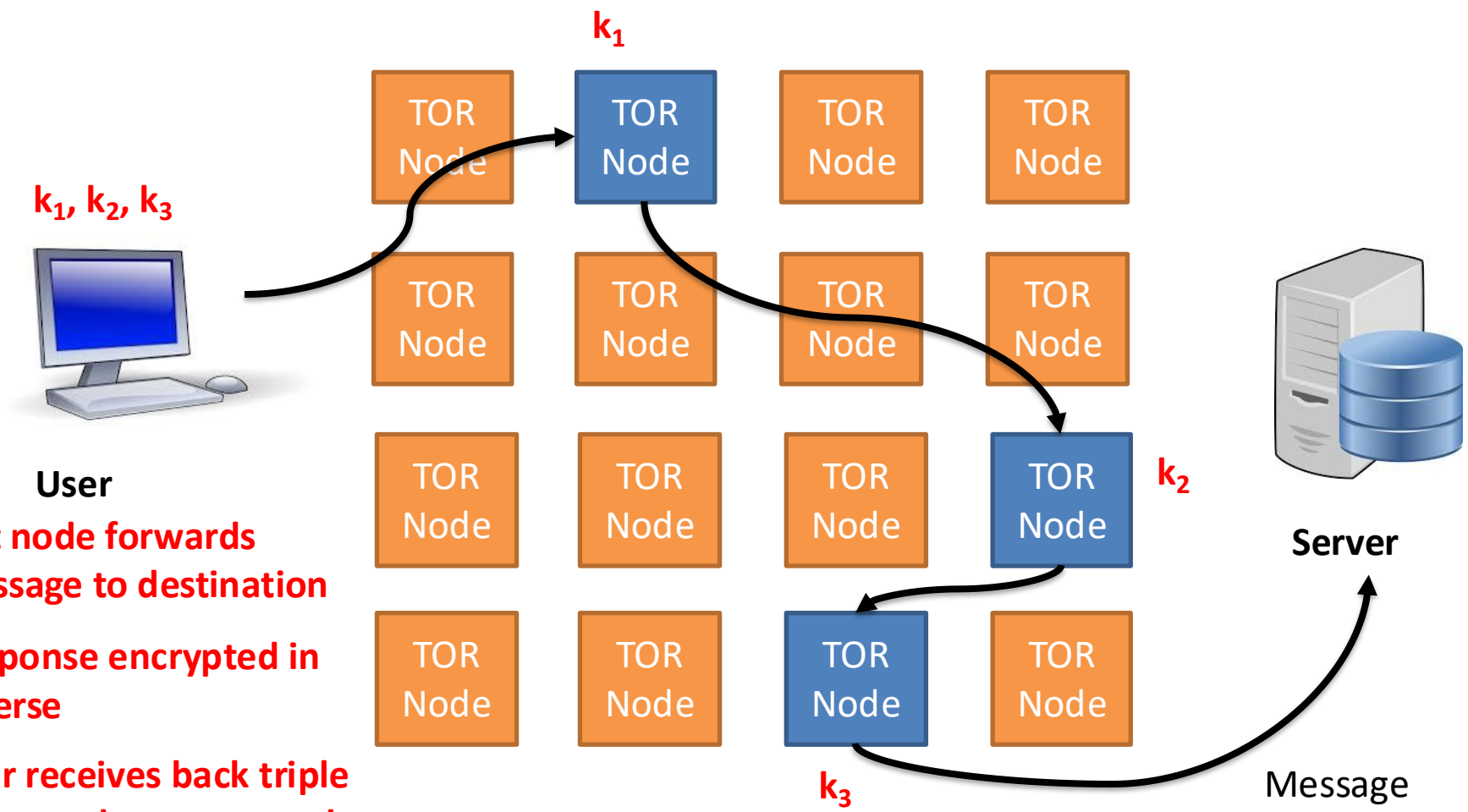


Third node decrypts outer layer

Learns message for destination server, but doesn't know where message came from originally (does know previous node)

Adapted from https://www.youtube.com/watch?v=QRYzre4bf7I

The Onion Router (TOR) obscures a message *and* the source and destination



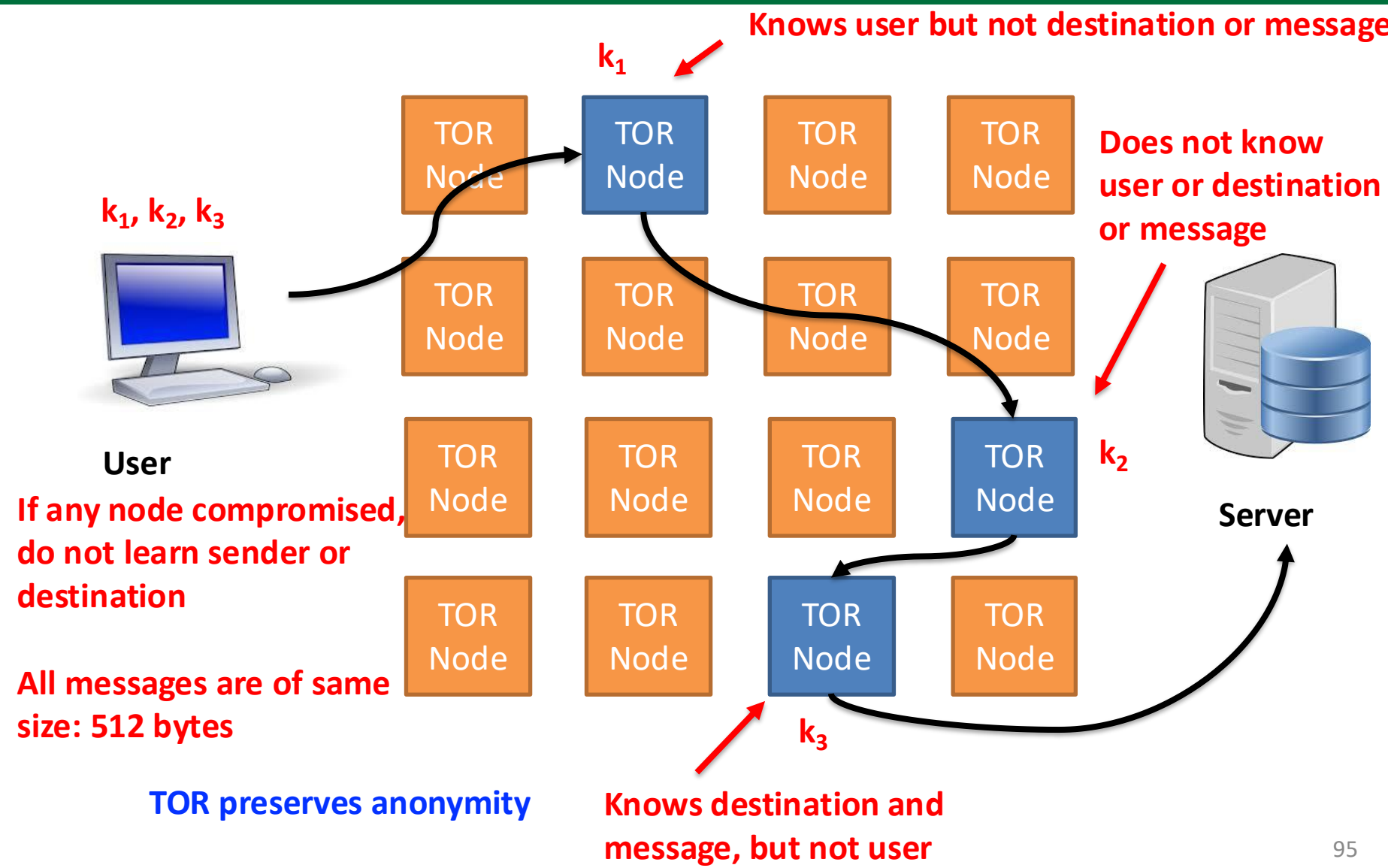
Exit node forwards message to destination

Response encrypted in reverse

User receives back triple encrypted message, only one who can decrypt

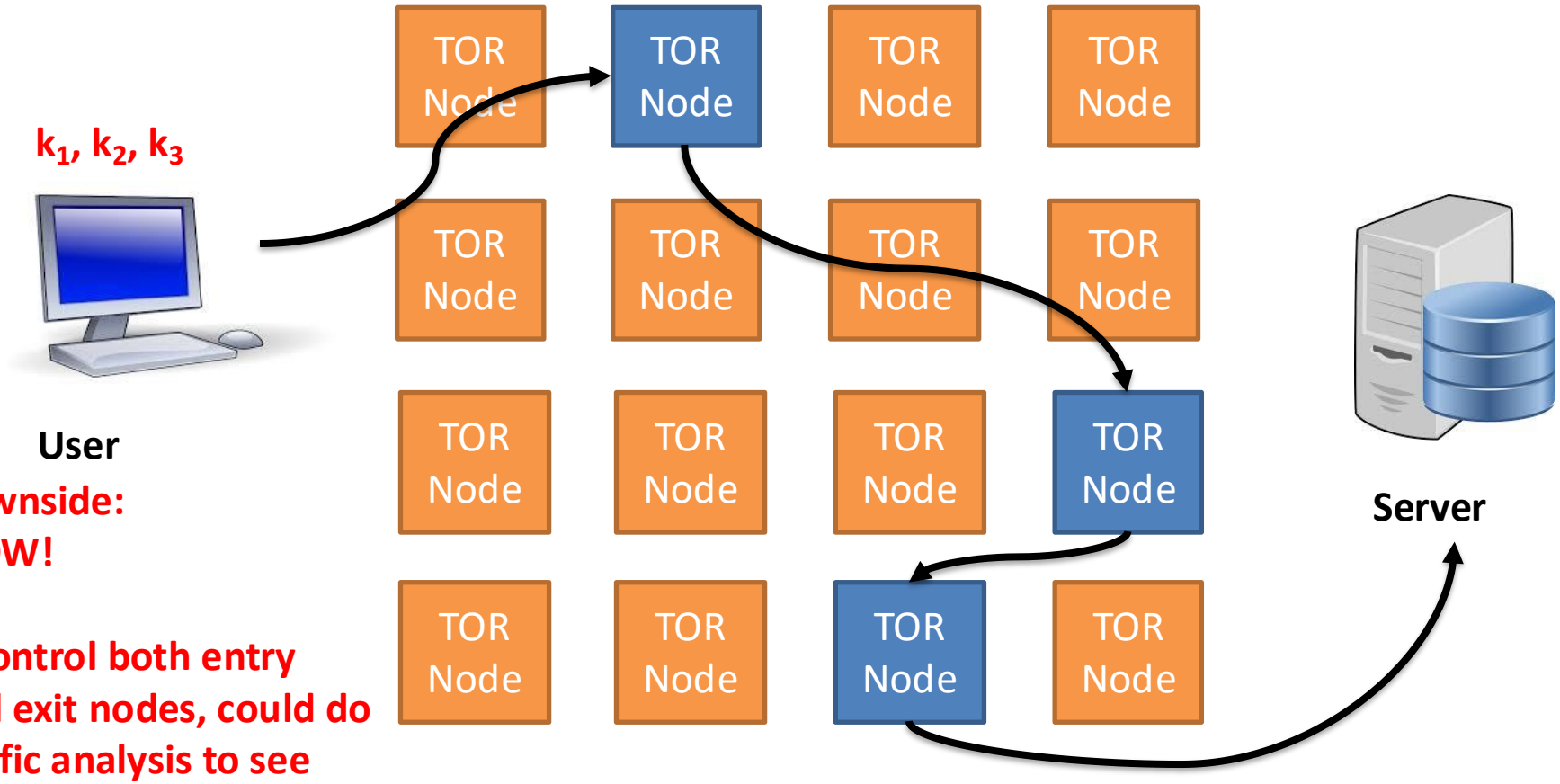
Adapted from https://www.youtube.com/watch?v=QRYzre4bf7I

The Onion Router (TOR) obscures a message *and* the source and destination



Adapted from https://www.youtube.com/watch?v=QRYzre4bf7I

The Onion Router (TOR) obscures a message *and* the source and destination



Downside: SLOW!

If control both entry and exit nodes, could do traffic analysis to see who is talking to whom

Adapted from https://www.youtube.com/watch?v=QRYzre4bf7I

DEMO

Start TOR browser

Go to www.amazon.com (will be slow to load, but Amazon doesn't know it's me, but shows me in Poland or else where)

See location by visiting https://ipapi.co/ in TOR browser
<wait for a bit…>
To change location in TOR browser:
File->New Circuit for this Site

Recently I got an exit node in Antarctica! Cool!

Agenda

1. Encryption
   - Symmetric (AES)
   - Hashing (SHA-2)
   - Asymmetric (RSA)
2. TLS
3. TOR
4. VPN
5. Wi-Fi security
6. Supplemental crypto material

Local Area Networks (LANs) are networks set up for a physical location

Often people outside want access to LAN

- Work from home
- Travelling
- Customer/partners

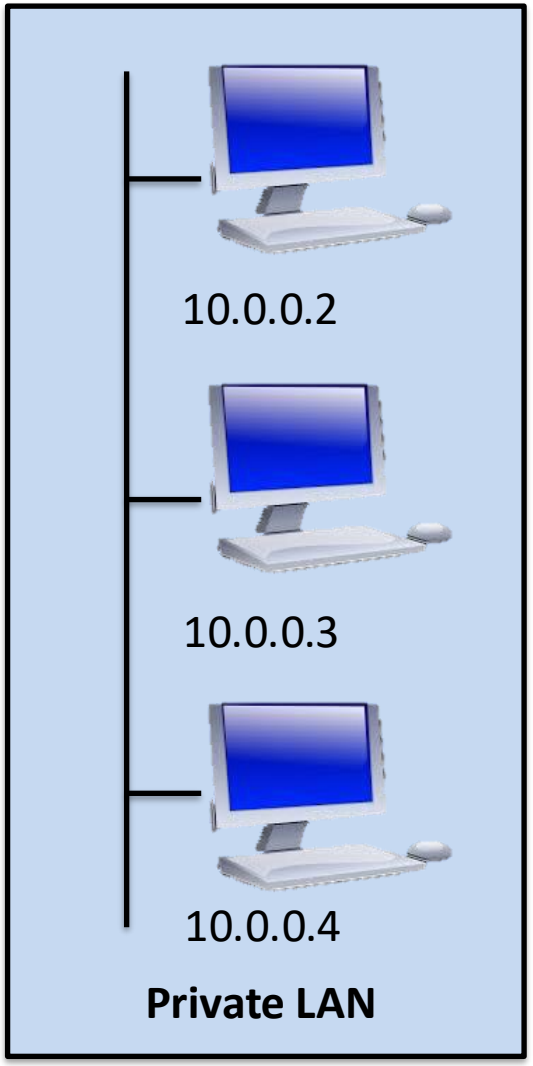
Could give devices routable IP address (or do port forwarding) and open them to the Internet

Problem?

- Increases attack surface!
- Netscout found devices attacked within 5 minutes[1]
- Hard and crunchy on the outside, software and chewy on the inside!

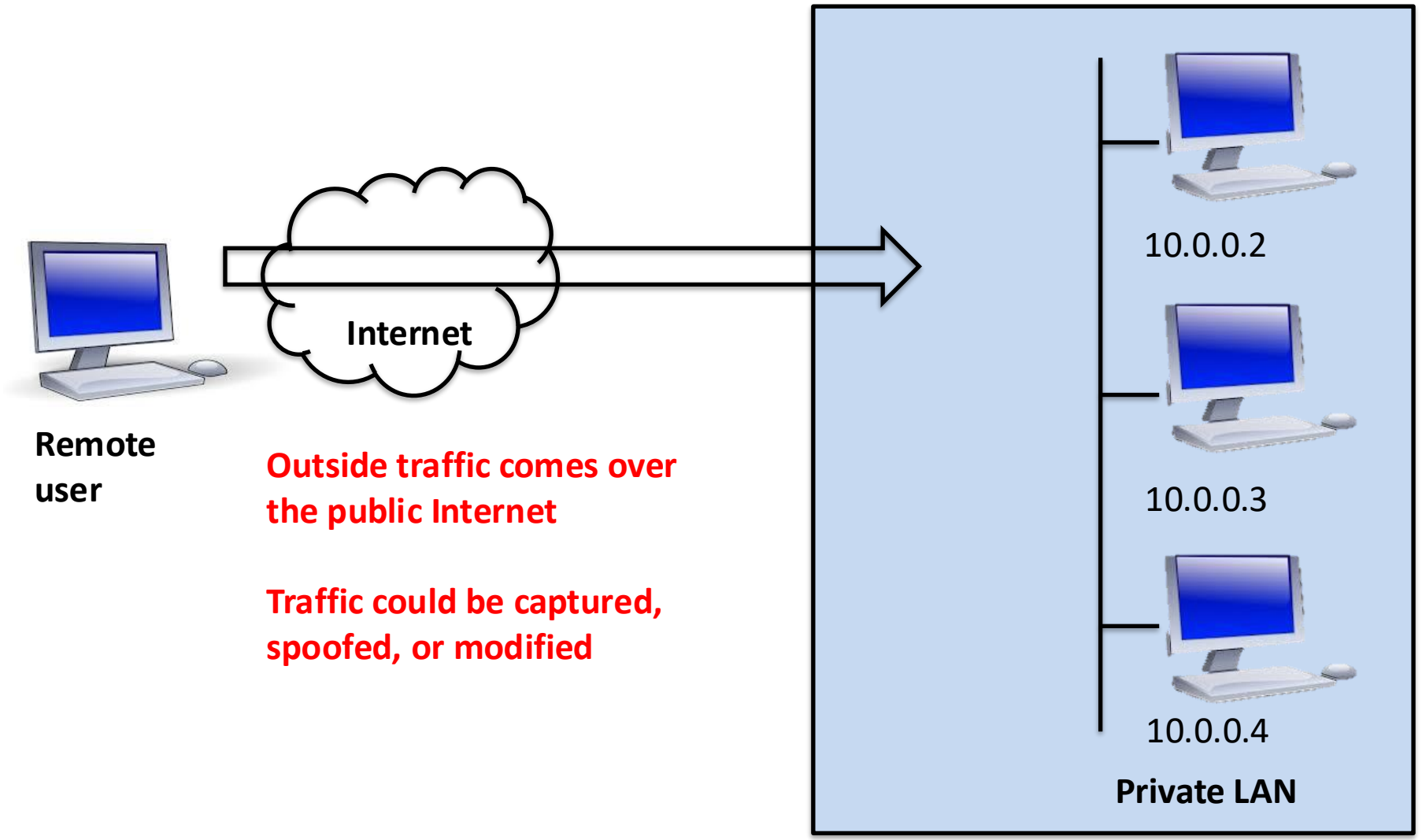
Devices set up with addresses in non-routable range(e.g., 10.0.0.0/8 or 192.168.0.0/16)

Device in local area able to talk to each other (router not shown)



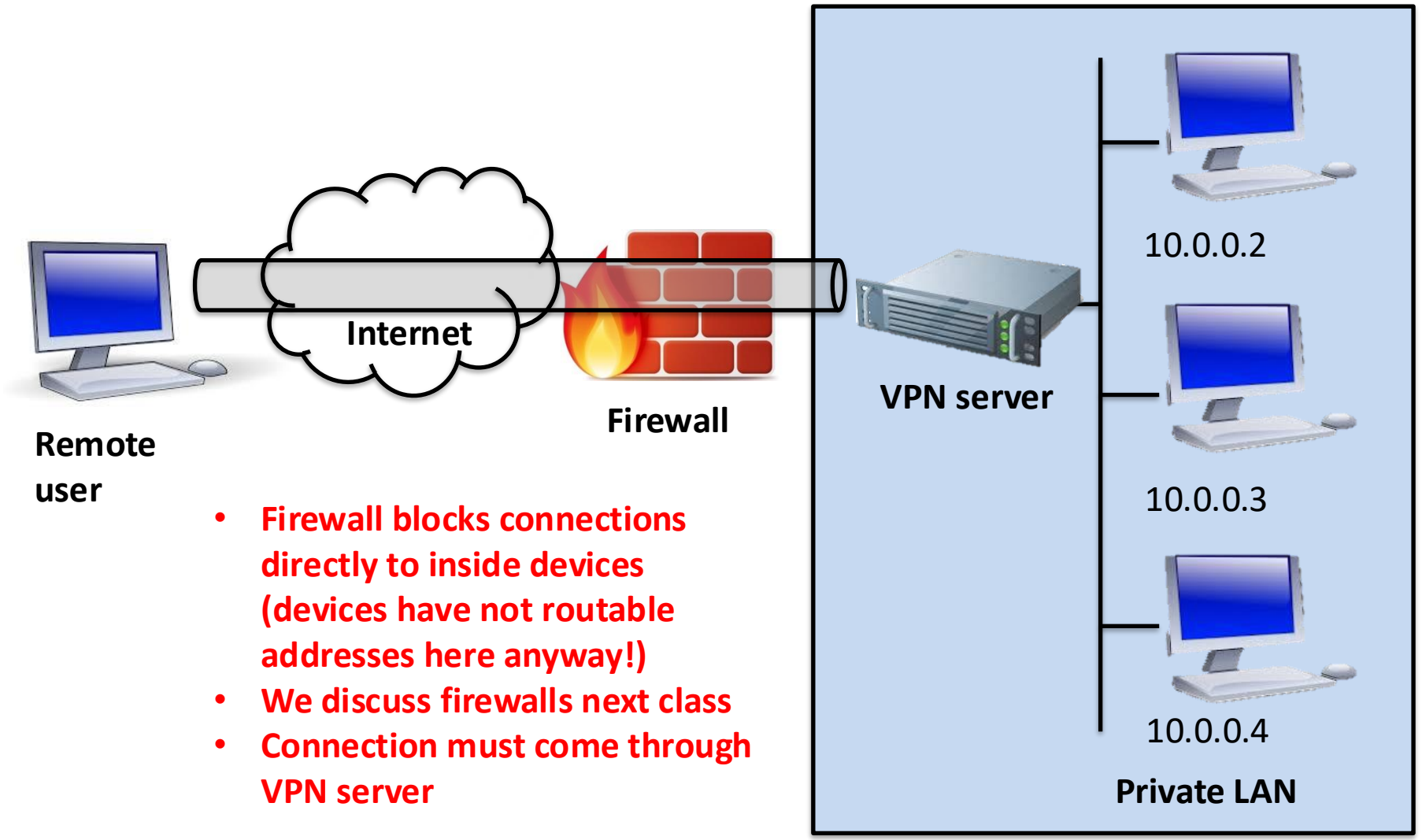
[1] https://www.netscout.com/sites/default/files/2019-07/SECR_010_EN-1901%20%E2%80%93%20NETSCOUT%20Threat%20Report%201H%202019%20%E2%80%93%20Web.pdf

Outside traffic that comes over the public Internet cannot be trusted



Adapted from *Computer & Internet Security* by Du

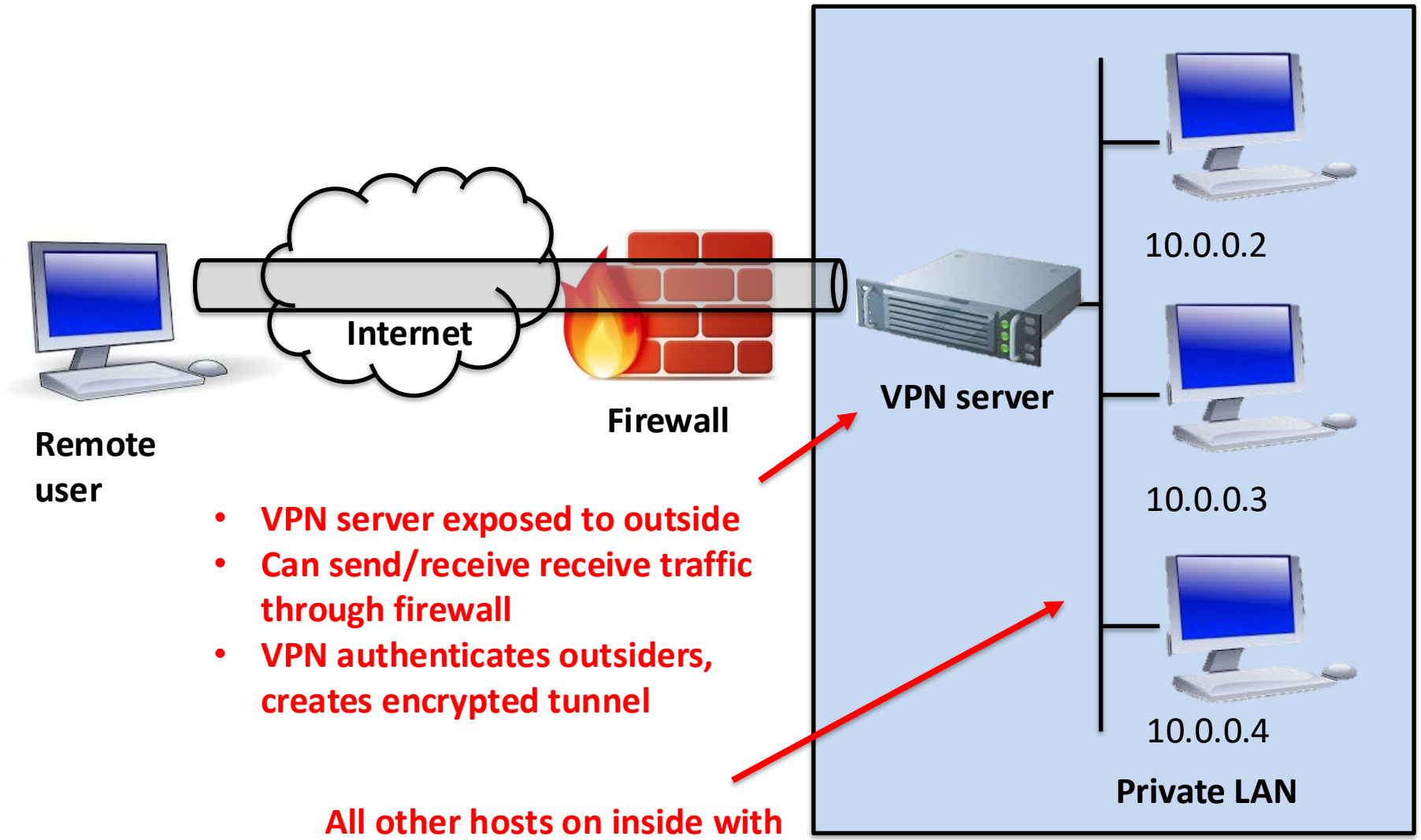
All traffic from outside to inside devices must come through VPN server



- Firewall blocks connections directly to inside devices (devices have not routable addresses here anyway!)
- We discuss firewalls next class
- Connection must come through VPN server

Adapted from *Computer & Internet Security* by Du

VPNs allow secure access to private LAN from outside as if device is inside

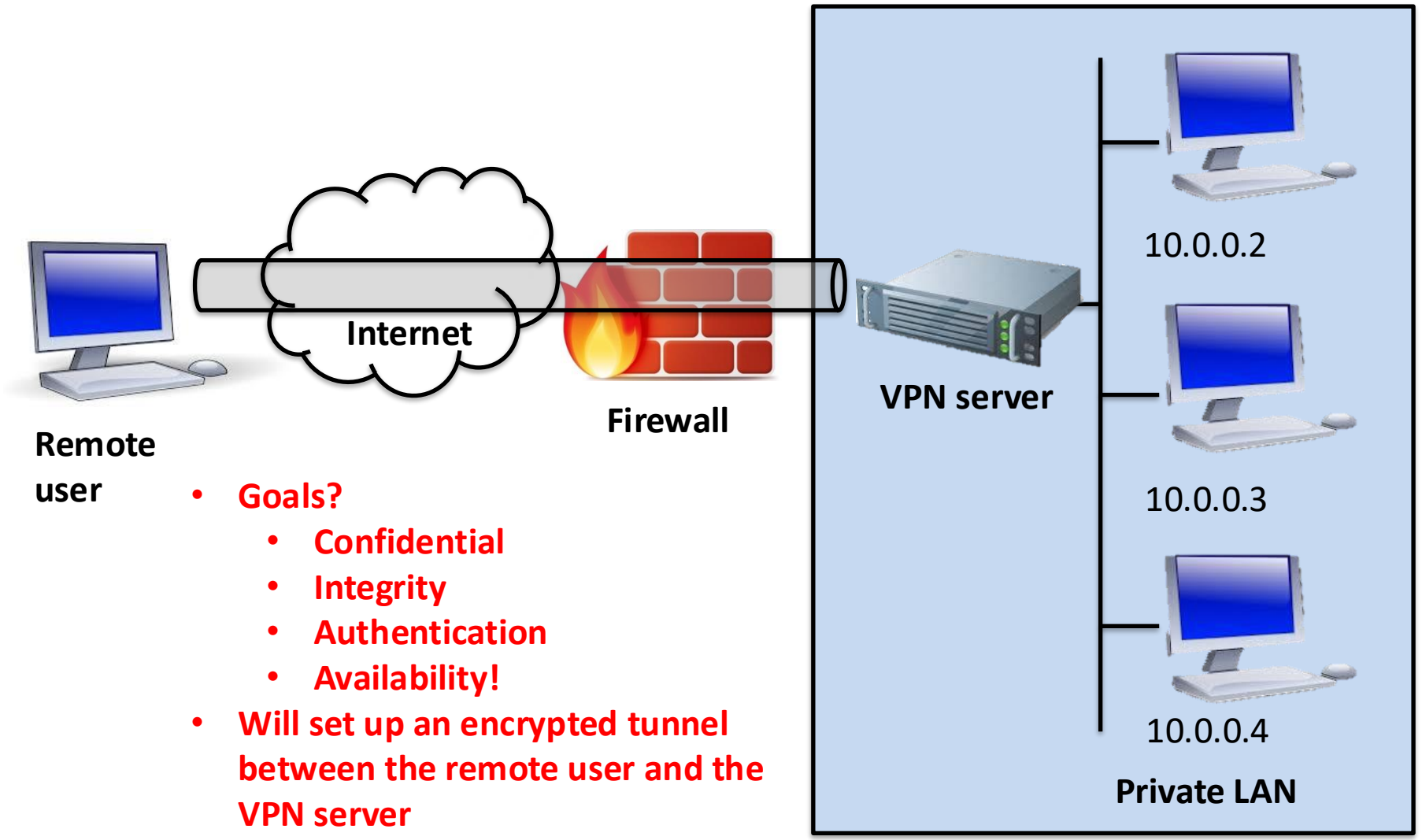


- VPN server exposed to outside
- Can send/receive receive traffic through firewall
- VPN authenticates outsiders, creates encrypted tunnel

All other hosts on inside with non-routable addresses

Adapted from *Computer & Internet Security* by Du

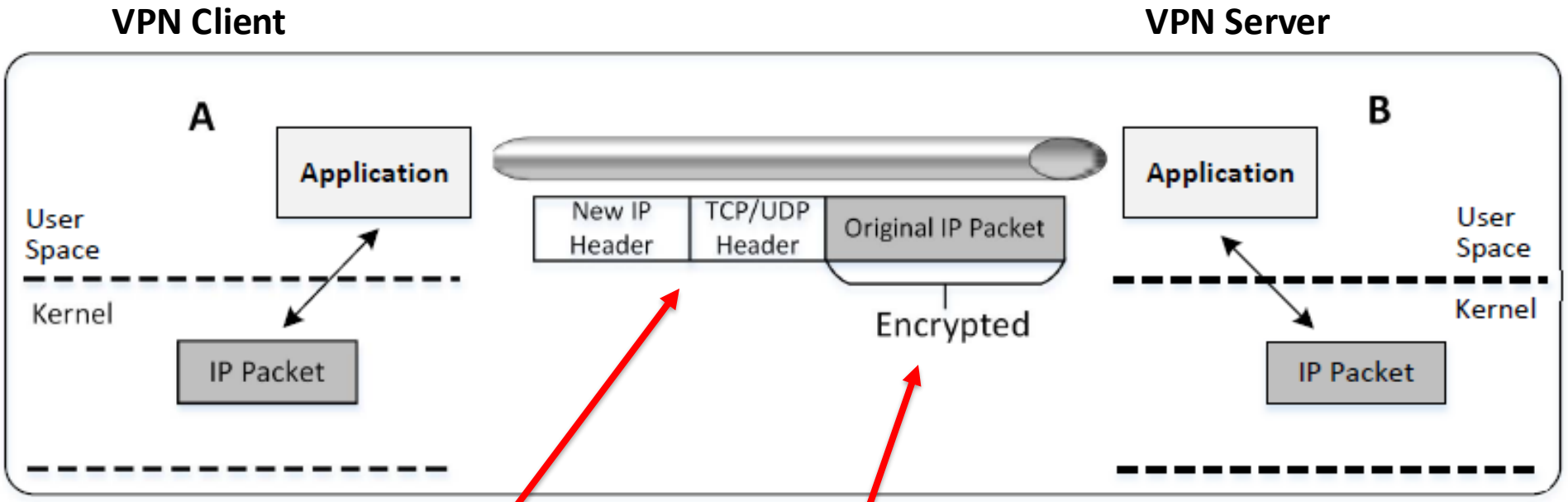
All traffic from outside to inside devices must come through VPN server



- Goals?
  - Confidential
  - Integrity
  - Authentication
  - Availability!
- Will set up an encrypted tunnel between the remote user and the VPN server
- VPN server will give access to LAN

Adapted from *Computer & Internet Security* by Du

Transport Layer tunneling securely sends packets to another network in three steps



VPN client application encrypts IP packet

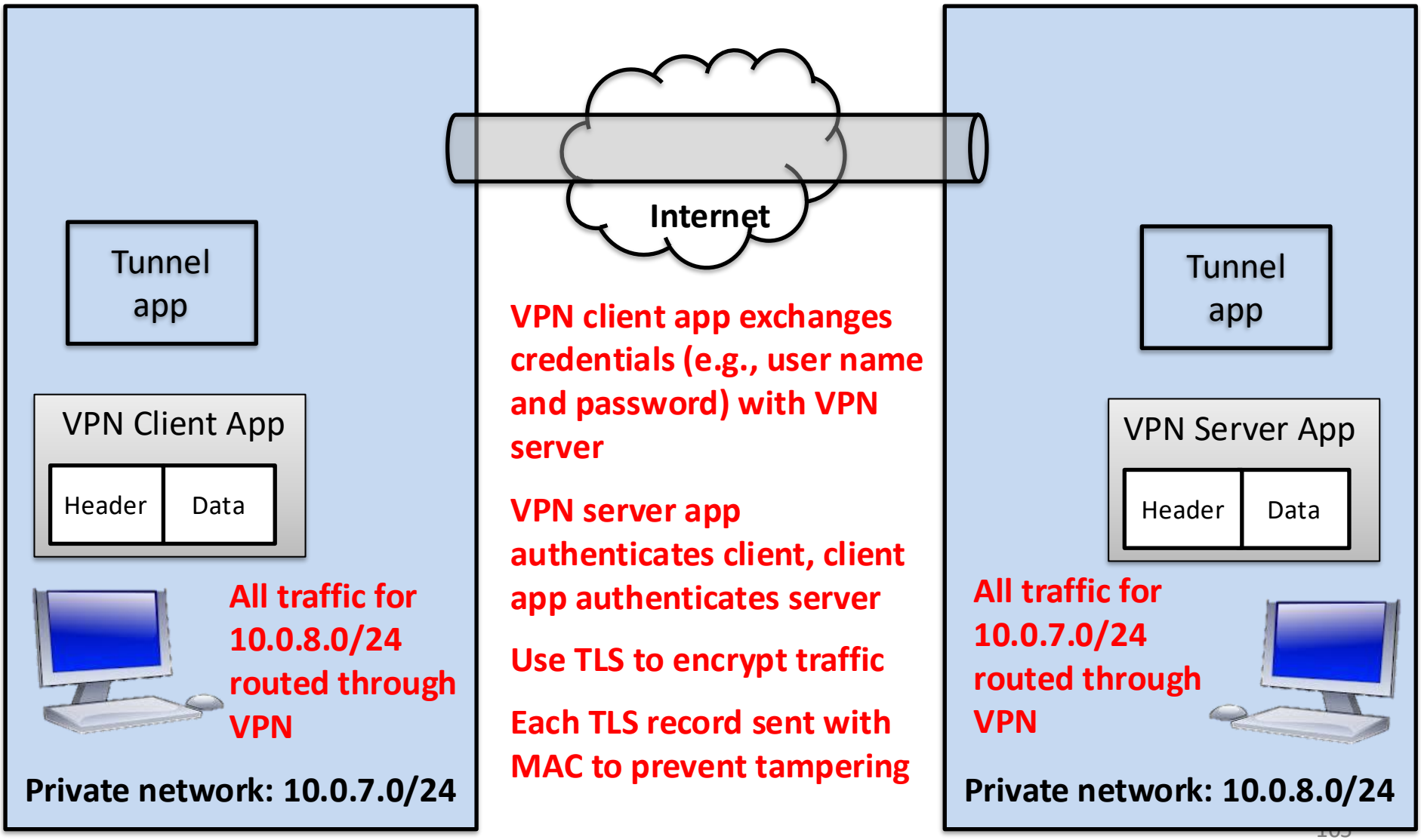
VPN client app adds new IP headers addressed to *VPN server*

Original source and destination IP addresses as well as payload are encrypted

Send packet over TLS connection to server
Someone sniffing on Internet sees packet, but does not know ultimate destination or data
TLS without VPN would reveal destination (here just see the VPN server as destination)
This technique is called TLS tunneling

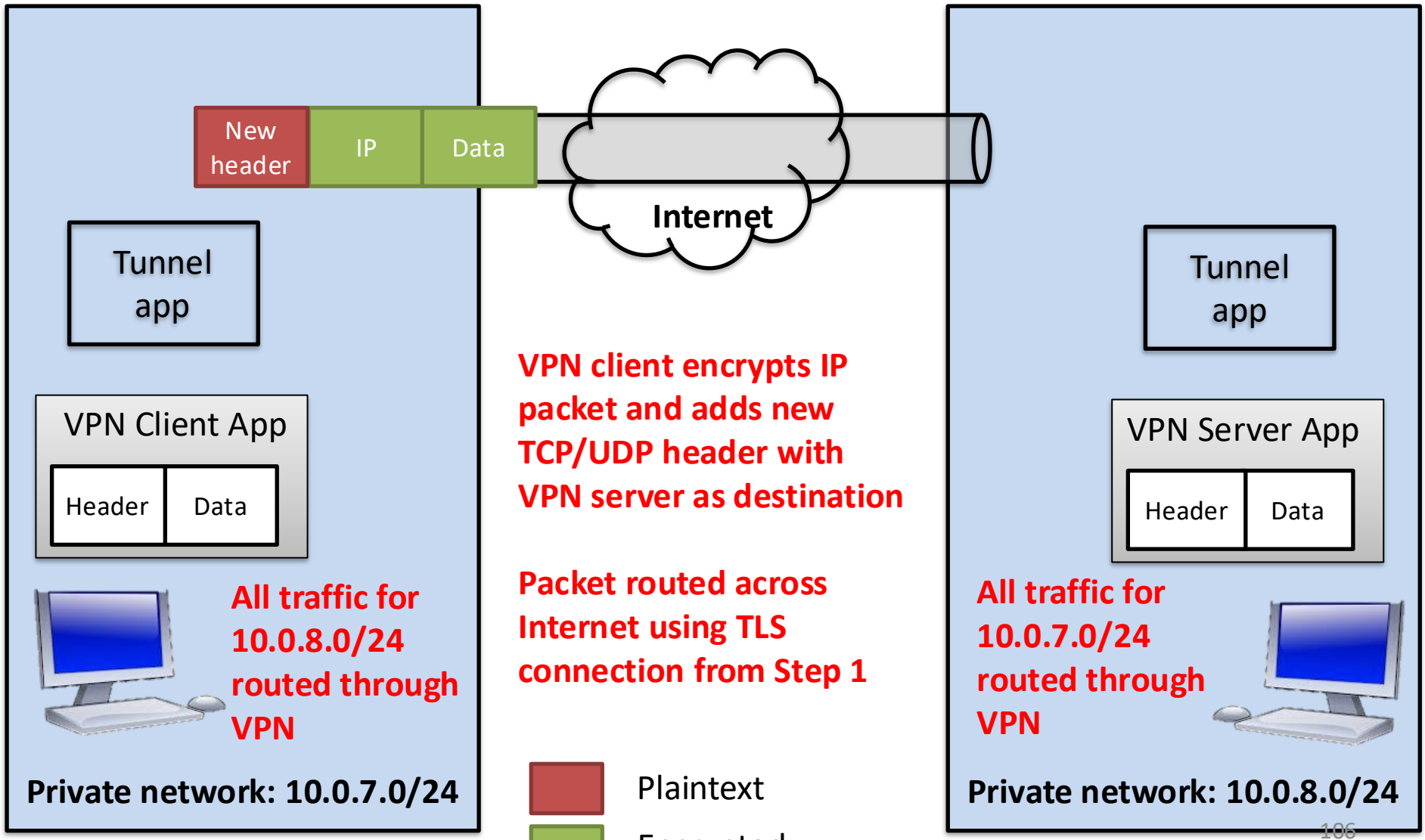
Adapted from *Computer & Internet Security* by Du

Step 1: Establish a TLS tunnel



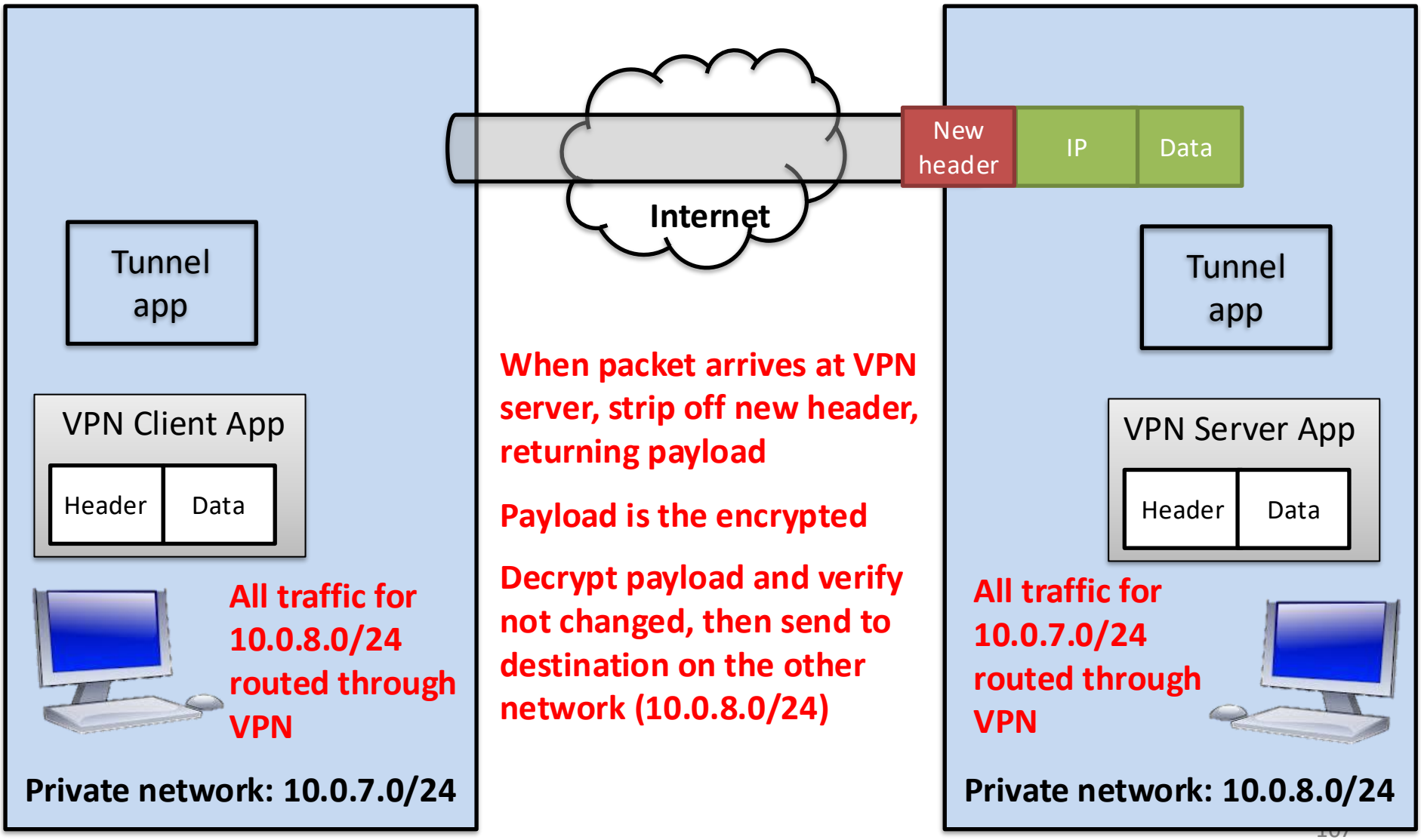
Adapted from *Computer & Internet Security* by Du

Step 2: Forward IP packets



Adapted from *Computer & Internet Security* by Du

Step 3: Release IP packets on the other network



Adapted from *Computer & Internet Security* by Du

Apart from tunneling, VPNs can be used to bypass firewalls

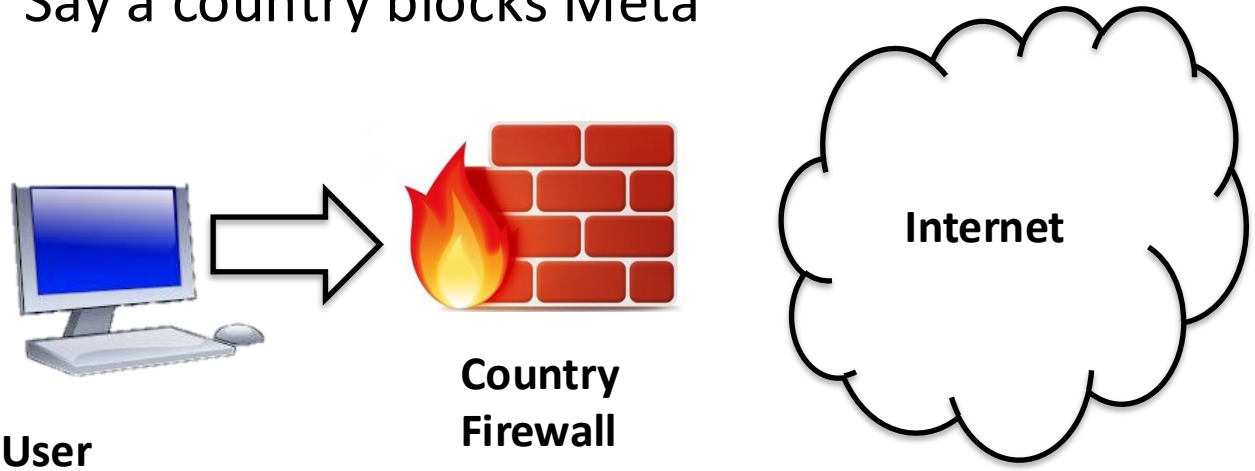
Image a country has implemented a firewall to prevent access to some international Internet sites

- Firewall looks at destination IP address
- Drops packet if address is on blacklist

Say a country blocks Meta



Meta



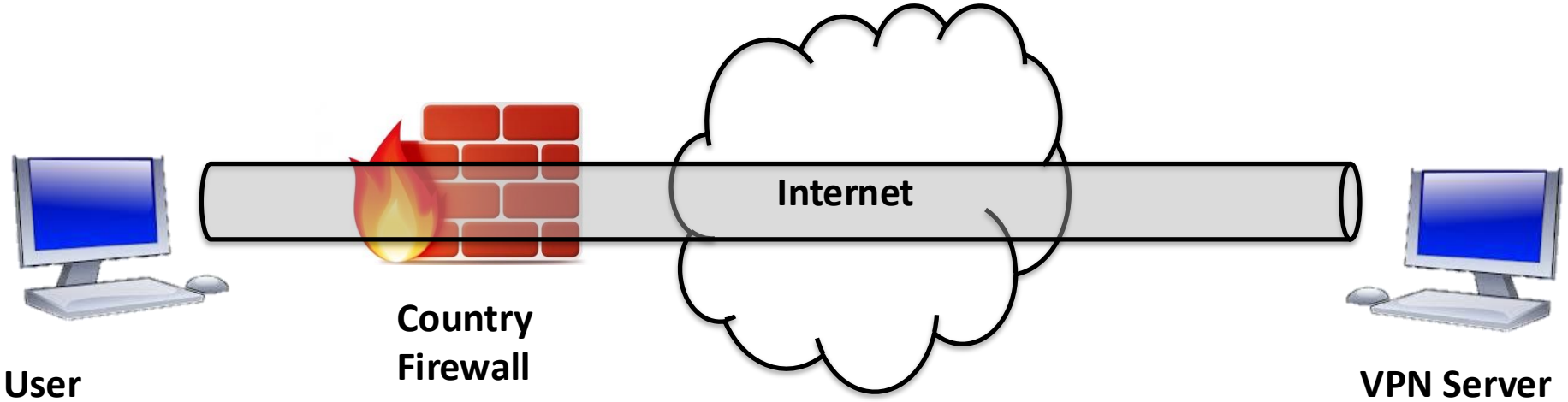
Apart from tunneling, VPNs can be used to bypass firewalls

To bypass firewall

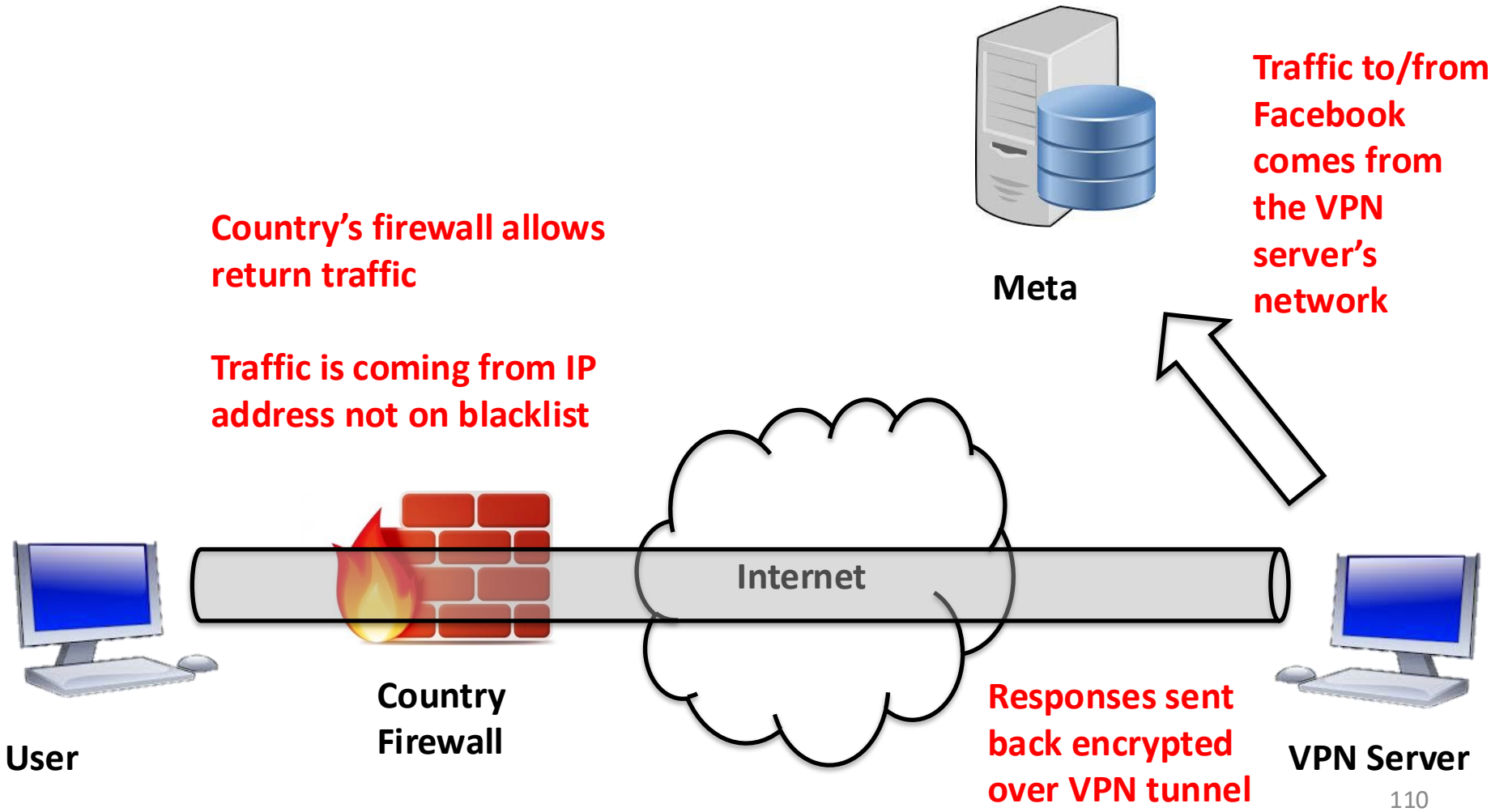
- Establish VPN to server not on blocked list
- Say VPN server is in another country and VPN server IP address not block by country firewall
- Firewall cannot see real destination (e.g., Meta) because IP headers encrypted inside VPN tunnel
- Firewall only knows packet bound for VPN server
- Traffic allowed by firewall



Meta



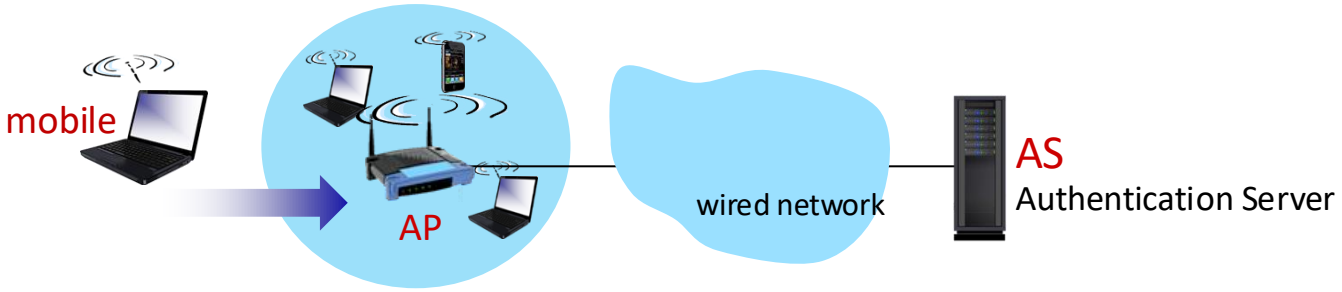
Apart from tunneling, VPNs can be used to bypass firewalls



Agenda

1. Encryption
   - Symmetric (AES)
   - Hashing (SHA-2)
   - Asymmetric (RSA)
2. TLS
3. TOR
4. VPN
5. Wi-Fi security
6. Supplemental crypto material

802.11: authentication, encryption

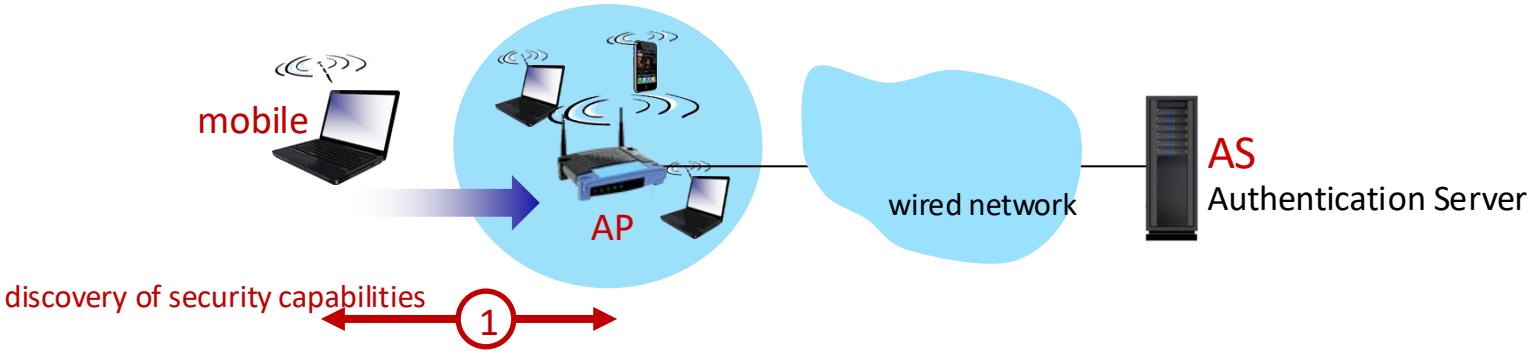


Arriving mobile (client) must:

- associate with access point: (establish) communication over wireless link
- perform mutual authentication with network
- encrypt over-the-air frames, during authentication and after (i.e., user data frames)

Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

802.11: authentication, encryption



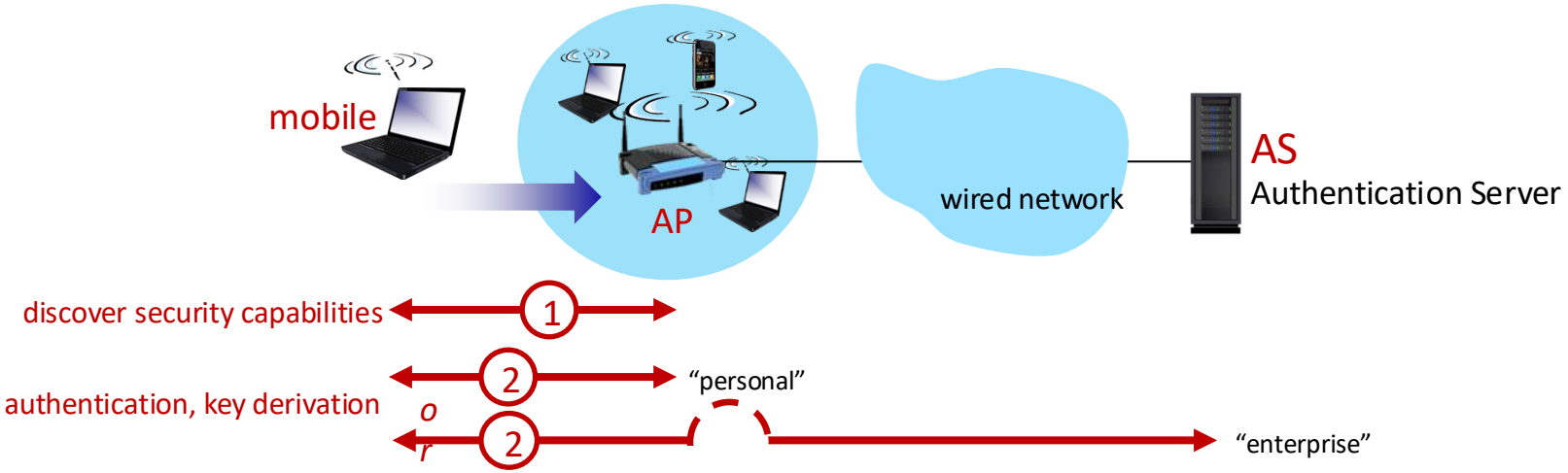
1 discovery of security capabilities:

- AP advertises its presence via `beacon` frame or device sends `probe request` frame
- AP sends it forms of authentication and encryption provided
- device requests specific forms authentication, encryption desired

initially message exchange: device not authenticated, encryption keys not known

Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

802.11: authentication, encryption

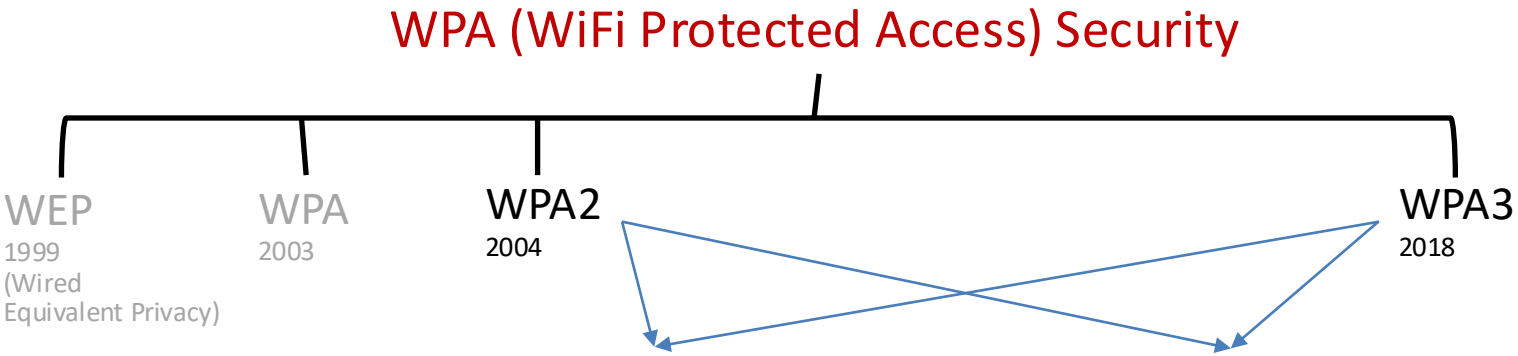


(2) mutual authentication and shared symmetric key derivation:

- AP, mobile may already have shared common secret (e.g., password)
- Use shared secret, nonces (to prevent relay attacks), cryptographic hashing (to ensure message integrity) to
  - authenticate each other
  - derive symmetric key (e.g., AES key) for user session frames

Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

WiFi security standards: personal, enterprise



Personal:

- common in home, small offices
- *same password* for all devices
- *protocol between user device, AP*
- WPA2/3-PSK (pre-shared key)

Enterprise:

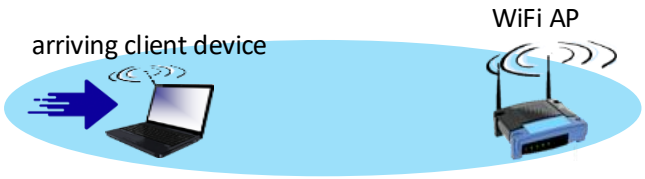
- common in enterprises
- password- and/or certificate-based
- *different per-user passwords*
- *protocols among user device, AP, authentication server* (e.g., RADIUS)
- EAP(Extensible Authentication Protocol: EAP-TLS, EAP-TTLS plus many more

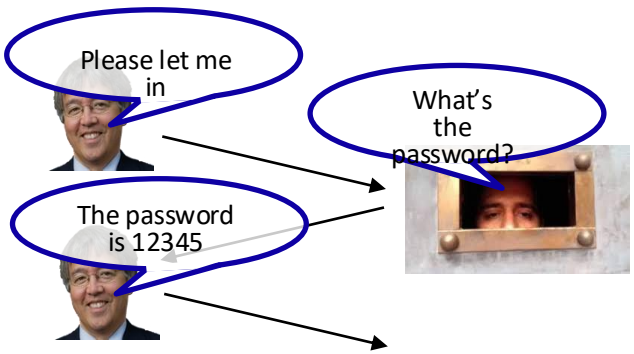
Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

WiFi security: infrastructure

Personal

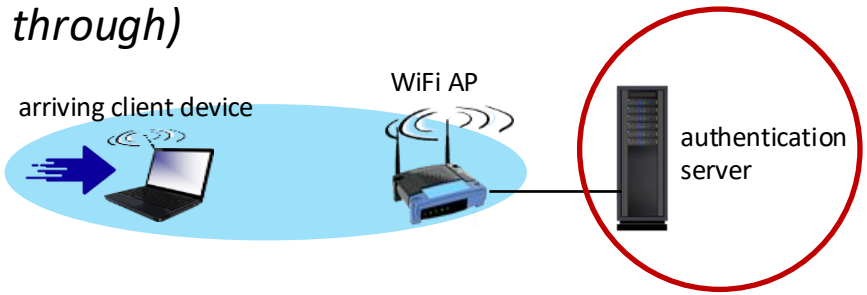
- Between user device and AP
- WPA2/3-PSK: password known to device, AP

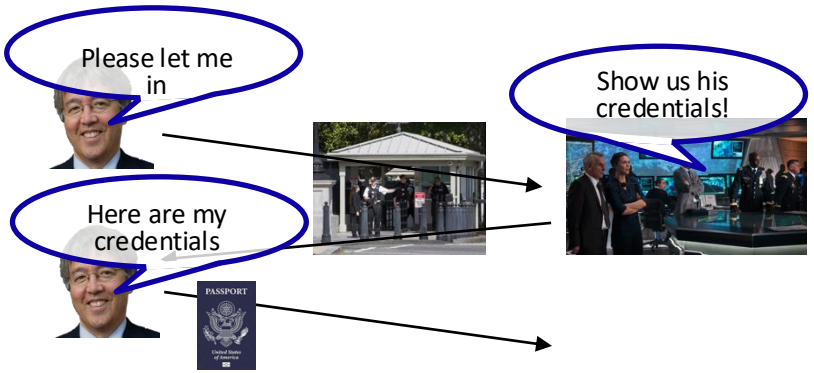




Enterprise

- Mostly between user device authentication server *(AP acts as pass through)*





Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

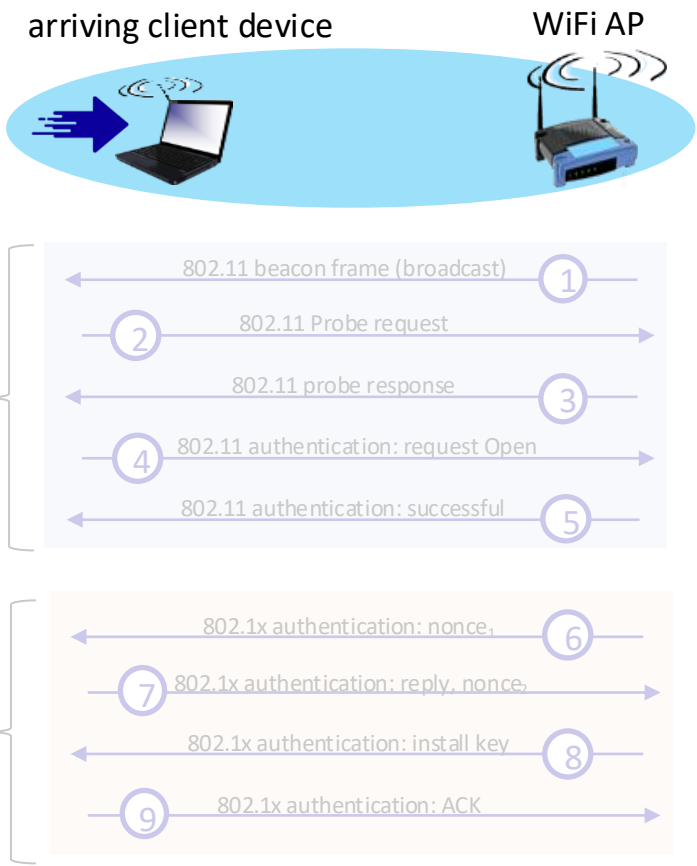
WPA2-PSK (pre-shard key)

- same password configured on AP for all devices
- password known ahead of time ("pre-shared") by devices

Two "phases" of WPA2-PSK authentication

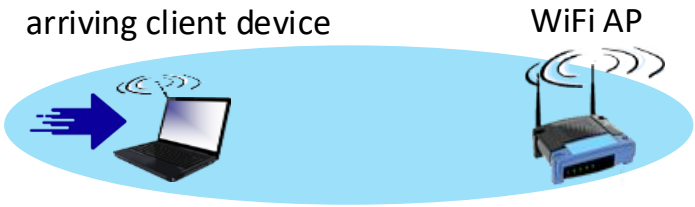
802.11 open "authentication": client associates with AP (no actual "authentication" yet)

802.1x WPA2/3-PSK: authentication, generate new AES session encryption key



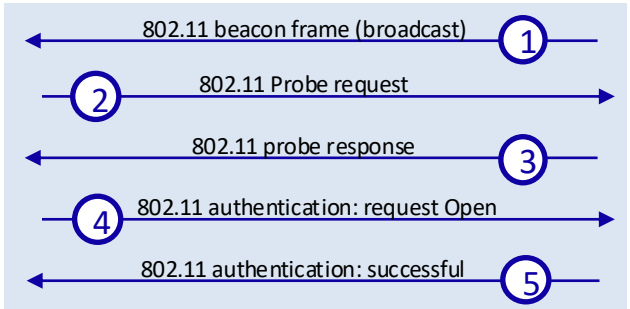
Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

WPA2-PSK: phase 1



1. AP broadcasts 802.11 beacon frames (packet 1 in Wireshark trace*)
2. Client sends 802.11 Probe Request (packet 2)
3. AP responds with 802.11 Probe Response (packet 3)
4. Client sends 802.11 Authentication frame, requesting Open System," i.e., *no password* authentication (packet 4)
5. AP responds with 802.11 Authentication frame (packet 5), with "Status Code: Successful"

... at this point, "real" authentication has not yet happened!



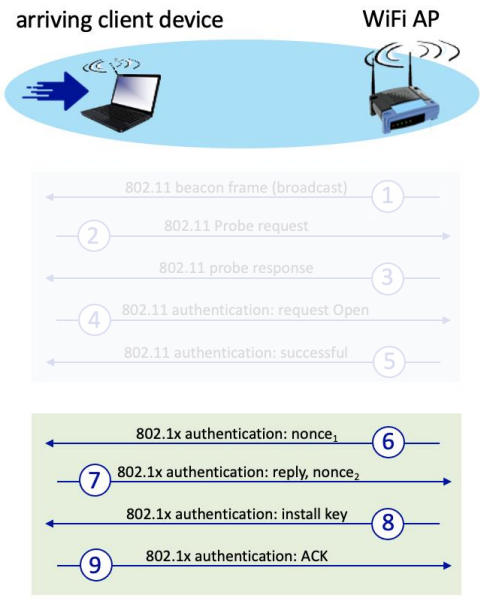
* http://gaia.cs.umass.edu/kurose/carioca_Iphone_association_selected_packets_onlyV2.pcapng

Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

WPA2-PSK: phase 2 - 4-way EAPOL handshake

EAPOL: Extensible Authentication Protocol over LAN

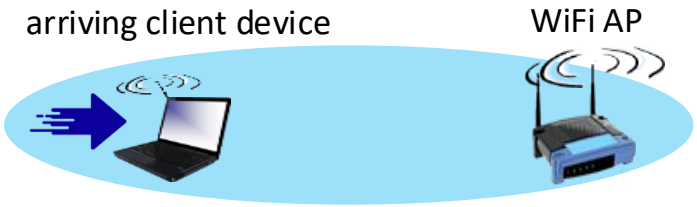
- client, AP have established (unsecured) 802.11 session, know shared secret (password)
- 4-way EAPOL handshake: use shared secret, nonces to generate *new one-time session key* for encrypting this session:
  - *shared secret:* used to encrypt, and provide message integrity (MIC) for EAPOL messages
  - *MIC nonces:* guarantee liveness, protect against replay attacks
  - *new, one-time AES session key:* protects against brute force decryption attacks on shared secret



4-way EAPOL handshake for WPA2-PSK

Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

WPA2-PSK: phase 2



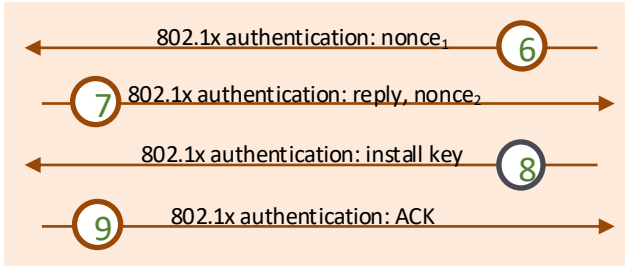
6. AP send 802.1x EAPOL message: WPA2-PSK (aka RSN), $nonce_1$ (packet 6 in Wireshark trace*)

7. Client uses shared secret and $nonce_1$ to generate reply with MIC, also generates, sends $nonce_2$ (packet 7)

8. AP checks MIC (verifying client liveness), generates new AES session key using shared secret, MIC, sends to client (packet 8)

9. Client verifies MIC, sends ACKs(packet 9), installs AES session key … *both sides ready to go!*

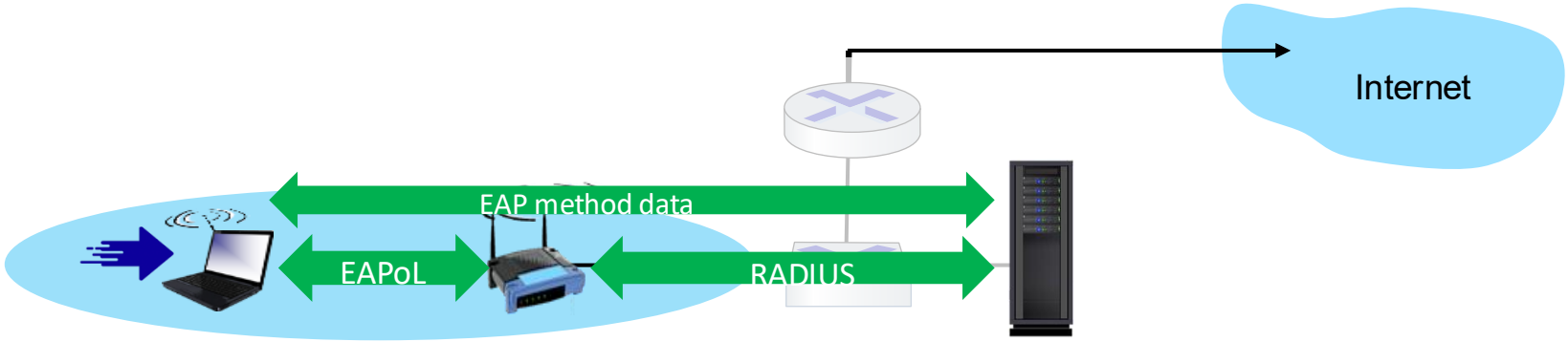
… at this point, AES encryption key known on both sides (note: shared secret never actually exchanged!)



* http://gaia.cs.umass.edu/kurose/carioca_Iphone_association_selected_packets_onlyV2.pcapng

Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

802.1x Enterprise Authentication Framework



Supplicant

Client device needing authentication before being allowed network access

Authenticator

relays messages between client and authentication server to authenticate client, authorize client use of WLAN

Authentication server

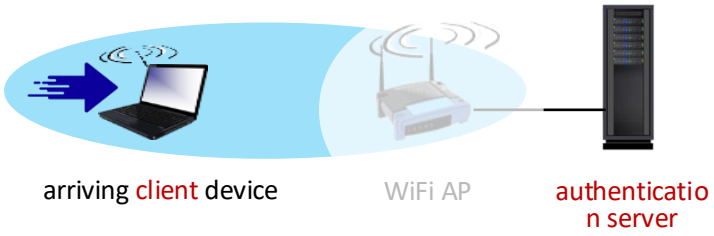
server with clients' authentication and access rights information

802.1x enterprise terminology

Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

802.1x Enterprise: Extensible Authentication Protocols: EAPs

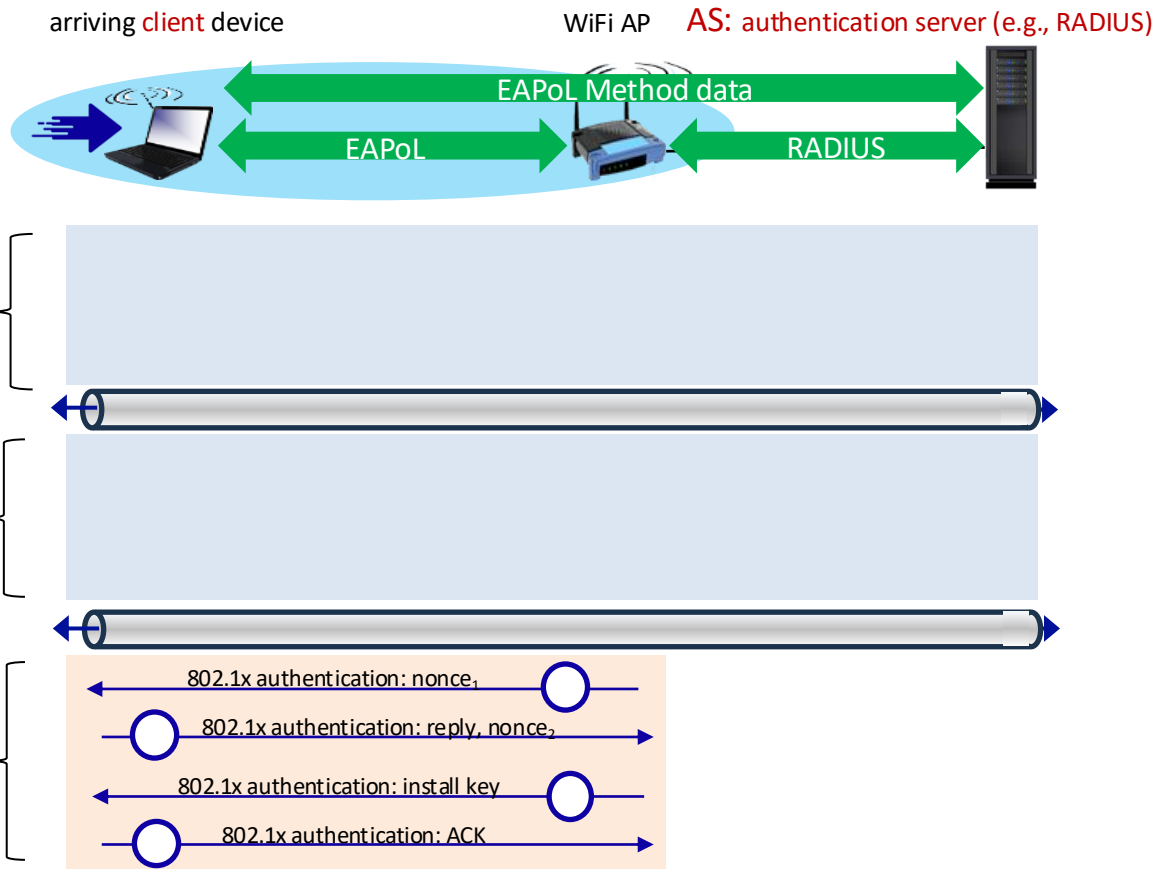
- EAP primarily between client, authentication server (e.g., RADIUS server)
  - AP acts mostly as pass through
- credentials stored at clients, authentication server
- ~40 different EAP protocols (!)
  - Differing in credentials: passwords, certificates, SIM cards, tokens ....
  - commonly used : EAP-MD-5, EAP-TLS, EAP-PEAP, EAP-TTLS, EAP-Fast, and Cisco LEAP. EAP-MD-5





EAP-TTLS: Tunneled Transport Layer Security [RFC 5281]

Two "phases" of EAP-TTLS

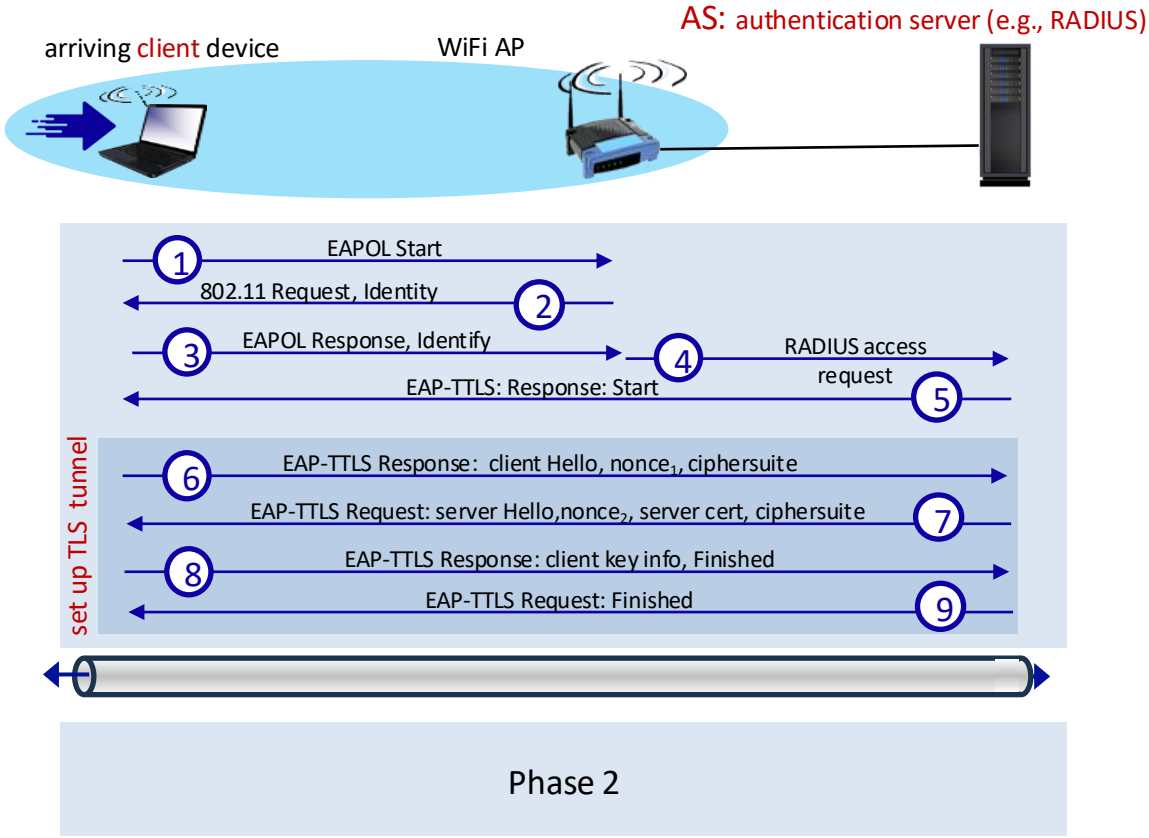


Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

EAP-TTLS: message exchange phase 1 (simplified details)

3. After some handshaking, client provides name to A
4. AP: a client is seeking authentication
5. RADIUS server: initiates EAP-TTLS with client
6. Client sends $nonce_1$, acceptable encryption schemes to AS
7. AS response with its certificate (for authentication), $nonce_2$, chosen encryption method.
8. Client authenticates server, sends client info (encrypted using server public key)

… at this point, AS is authenticated by client, encrypted session between client, AP



Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

EAP-TTLS: message exchange phase 2 (simplified details)

10 AS send identity request to client

11 Client responds with identity

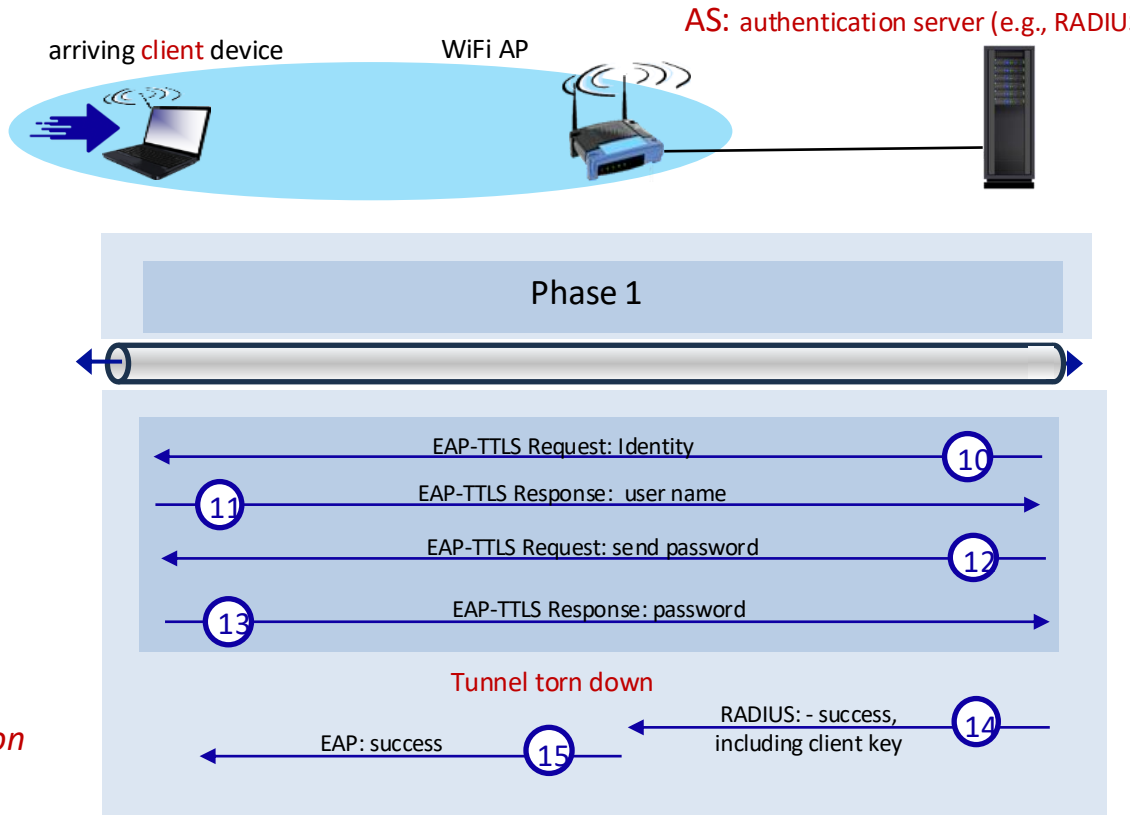
12 AS challenge - send password, or other credential, depending on EAP-TTLS authentication method

13 Client responds with credential

14 AS informs AP that client is authenticated, sends info needed for AP, client channel encryption

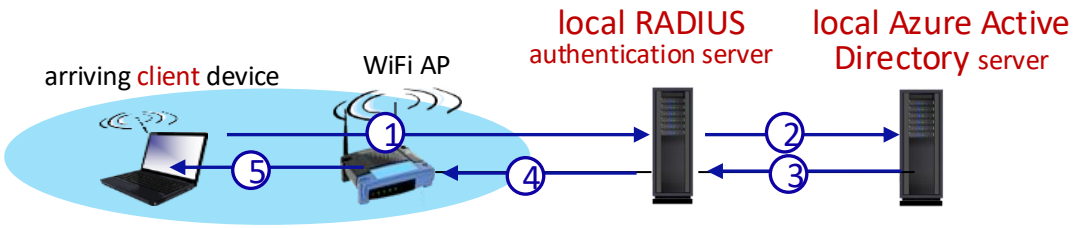
15 AP pass EAP success message to client

Next: client/AP agree on channel encryption using EAPoL 4-way handshake, then client can run DHCP

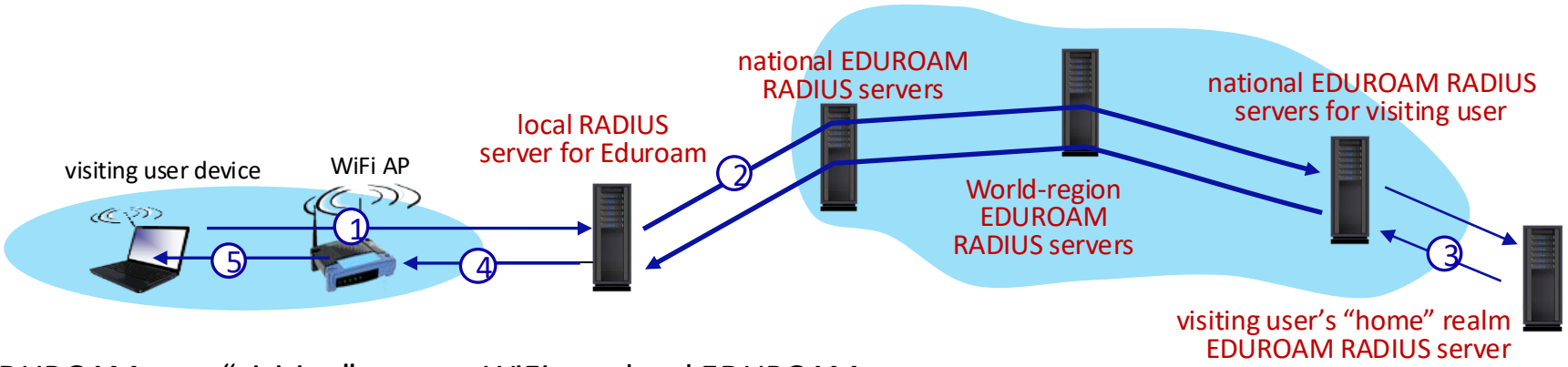


Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

Local Authentication Server may use proxies



Authentication proxied from local RADIUS server to local AD server



EDUROAM user "visiting" remote WiFi net: local EDUROAM RADIUS server proxies to user's home "realm" RADIUS server and accepts remote authentication decision

Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

Agenda

1. Encryption
   - Symmetric (AES)
   - Hashing (SHA-2)
   - Asymmetric (RSA)
2. TLS
3. TOR
4. VPN
5. Wi-Fi security
6. Supplemental crypto material

Supplemental crypto slides

Details on how to compute asymmetric keys

Prerequisite: modular arithmetic

- x mod n = remainder of x when divide by n
- Facts:

  $[(a \bmod n) + (b \bmod n)] \bmod n = (a+b) \bmod n$

  $[(a \bmod n) - (b \bmod n)] \bmod n = (a-b) \bmod n$

  $[(a \bmod n) * (b \bmod n)] \bmod n = (a*b) \bmod n$
- Thus

  $(a \bmod n)^d \bmod n = a^d \bmod n$
- Example: x=14, n=10, d=2:

  $(x \bmod n)^d \bmod n = 4^2 \bmod 10 = 6$

  $x^d = 14^2 = 196 \quad x^d \bmod 10 = 6$

Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

RSA: getting ready

- Message: just a bit pattern
- Bit pattern can be uniquely represented by an integer number
- Thus, encrypting a message is equivalent to encrypting a number

Example:

- m= 10010001. This message is uniquely represented by the decimal number 145.
- To encrypt m, we encrypt the corresponding number, which gives a new number (the ciphertext).

Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

RSA: Creating public/private key pair

1. Choose two large prime numbers *p, q.* (e.g., 1024 bits each)

2. Compute *n = pq, z = (p-1)(q-1)*

3. Choose *e (*with *e<n)* that has no common factors with z (*e, z* are said to be "relatively prime")

4. Choose *d* such that *ed-1* is exactly divisible by *z* (in other words: *ed* mod *z = 1*)

5. *Public* key is *(n,e)*. *Private* key is *(n,d)*.

K_B^+ K_B^-

Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

RSA: encryption, decryption

0. Given (n,e) and (n,d) as computed above

1. To encrypt message m ($<n$), compute
$c = m^e \bmod n$

2. To decrypt received bit pattern, c, compute
$m = c^d \bmod n$

magic happens! $m = (\underbrace{m^e \bmod n}_{c})^d \bmod n$

$c^d \bmod n$

Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

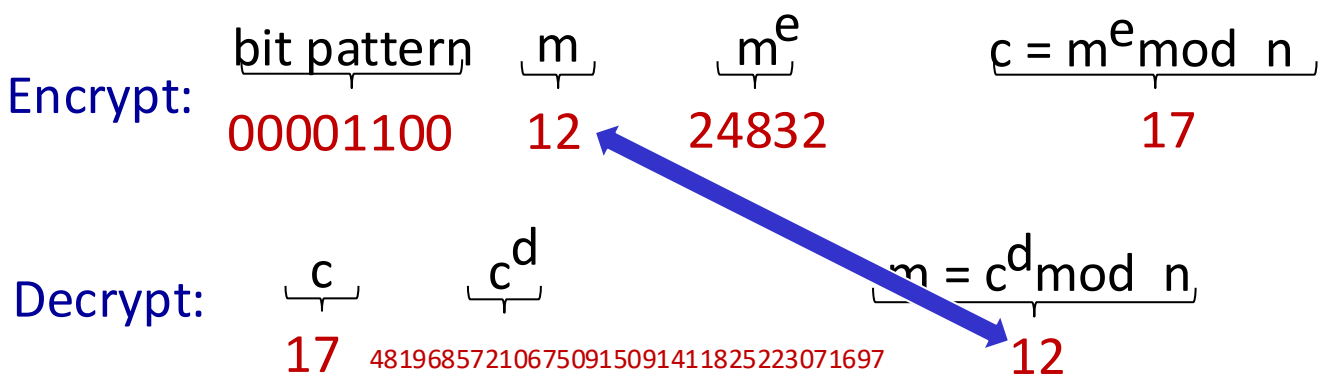
RSA example:

Bob chooses *p=5, q=7*. Then *n=35, z=24*.

e=5 (so *e, z* relatively prime).

d=29 (so *ed-1* exactly divisible by z).

Encrypting 8-bit messages



Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

Why does RSA work?

- Must show that $c^d \bmod n = m$, where $c = m^e \bmod n$
- Fact: for any x and y: $x^y \bmod n = x^{(y \bmod z)} \bmod n$
  - Where n= pq and z = (p-1)(q-1)
- Thus,

$$c^d \bmod n = (m^e \bmod n)^d \bmod n$$
$$= m^{ed} \bmod n$$
$$= m^{(ed \bmod z)} \bmod n$$
$$= m^1 \bmod n$$
$$= m$$

Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

RSA: another important property

The following property will be *very* useful later:

$$\underbrace{K_B^-(K_B^+(m))}_{\text{use public key first, followed by private key}} = m = \underbrace{K_B^+(K_B^-(m))}_{\text{use private key first, followed by public key}}$$

result is the same!

Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

RSA: another important property

Why $K_B^-(K_B^+(m)) = m = K_B^+(K_B^-(m))$?

follows directly from modular arithmetic:

$$(m^e \bmod n)^d \bmod n = m^{ed} \bmod n$$
$$= m^{de} \bmod n$$
$$= (m^d \bmod n)^e \bmod n$$

Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

Why is RSA secure?

- Suppose you know Bob's public key (n,e). How hard is it to determine d?
- Essentially need to find factors of n without knowing the two factors p and q
  - Fact: factoring a big number is hard

Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

RSA in practice: session keys

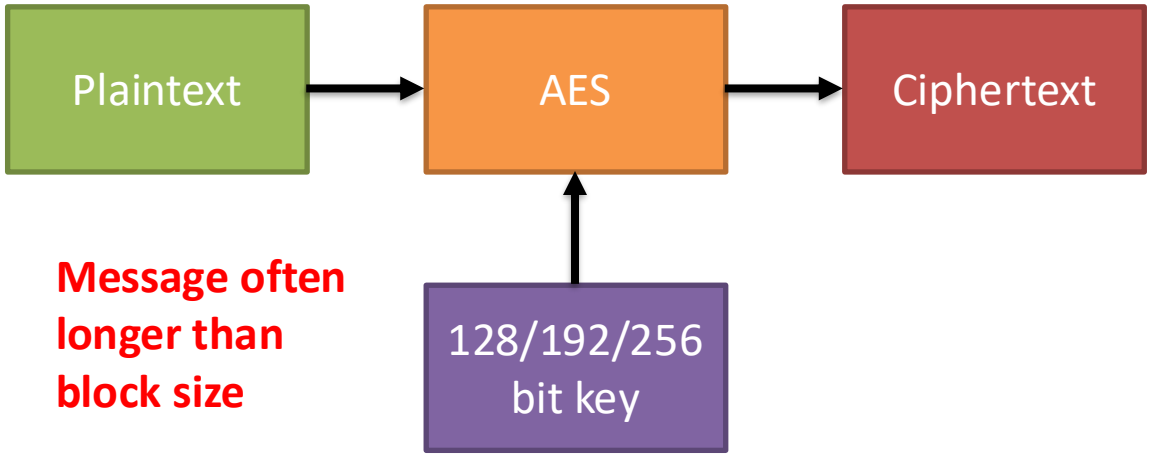
- Exponentiation in RSA is computationally intensive
- DES is at least 100 times faster than RSA
- Use public key crypto to establish secure connection, then establish second key – symmetric session key – for encrypting data

Session key, K_S

- Bob and Alice use RSA to exchange a symmetric session key K_S
- Once both have K_S, they use symmetric key cryptography

Adapted from Kurose and Ross: Computer Networking: A Top-Down Approach

Advanced Encryption Standard (AES) builds on the concept of an S-P network



Message often longer than block size

Break message into multiple 128-bit blocks

Key is padded with '#' (0x23) characters until multiple of 128 bits (16 bytes)

Advanced Encryption Standard (AES)

- Sometimes called Rijndael (pronounced like "rhine doll") after inventor names
- Established by NIST in 2001 (but still secure)
- Uses 128-bit (16-byte) input blocks, outputs 128-bit ciphertext blocks
- Key can be 128, 192, or 256 bits (equals 10,12 or 14 rounds)

Adapted from: https://www.educative.io/edpresso/what-is-the-aes-algorithm

Advanced Encryption Standard (AES) uses the concept of an S-P network

Simplified

128/192/256 bit key

Key expanded into 10, 12, or 14 round keys using key schedule algorithm

Adapted from: Mao, Wenbo. "Modern cryptography: theory and practice." (2004).

Advanced Encryption Standard (AES) uses the concept of an S-P network

128/192/256 bit key

Key expanded into 10, 12, or 14 round keys using key schedule algorithm

Plaintext

Key_0

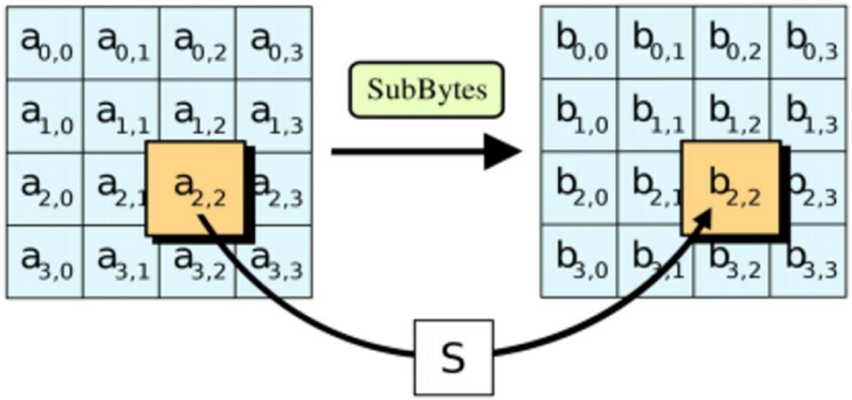
Round

- SubBytes(State)
- ShiftRows(State)
- MixColumns(State)
- AddRoundKey(State,Key)

Simplified

SubBytes(State)

- Substitute bytes from input based on pre-defined rules
- S-box

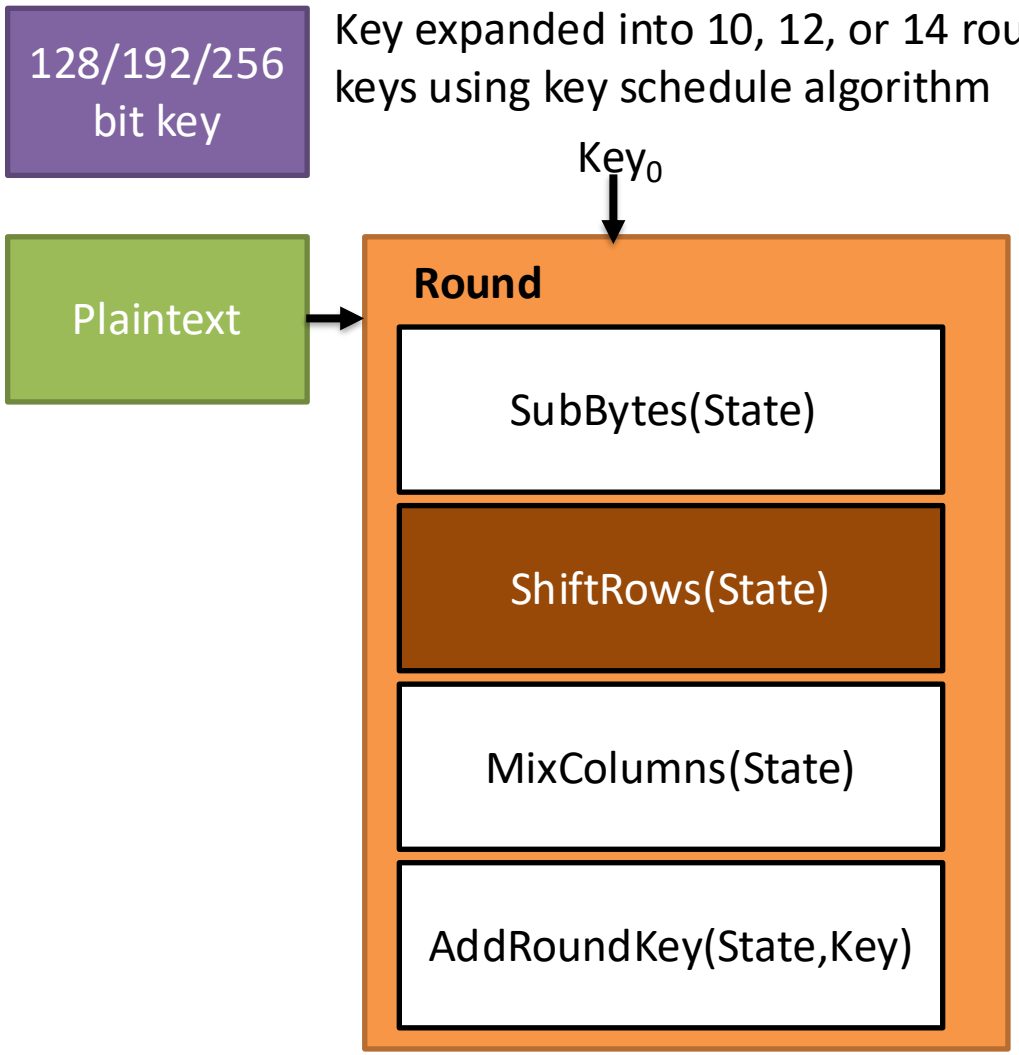


Adapted from: Mao, Wenbo. "Modern cryptography: theory and practice." (2004).

Advanced Encryption Standard (AES) uses the concept of an S-P network

128/192/256 bit key

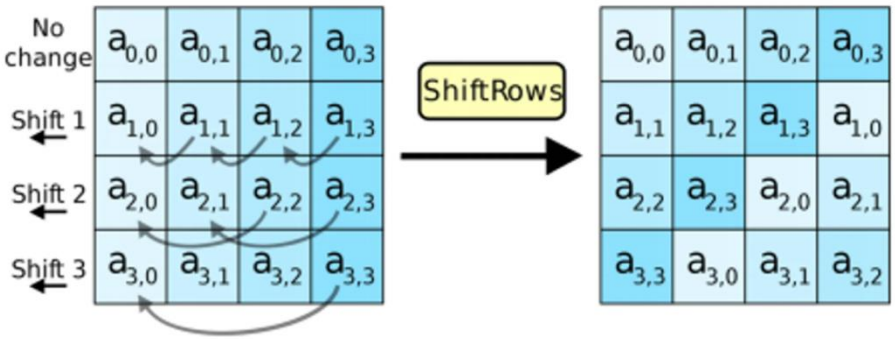
Key expanded into 10, 12, or 14 round keys using key schedule algorithm



Simplified

ShiftRows(State)

- Permute message by shifting row
- Row 0 not changed; other rows shift by increasing amount
- P-box



Adapted from: Mao, Wenbo. "Modern cryptography: theory and practice." (2004). Images from: https://www.educative.io/edpresso/what-is-the-aes-algorithm

Advanced Encryption Standard (AES) uses the concept of an S-P network

Simplified

128/192/256 bit key

Key expanded into 10, 12, or 14 round keys using key schedule algorithm

Key_0

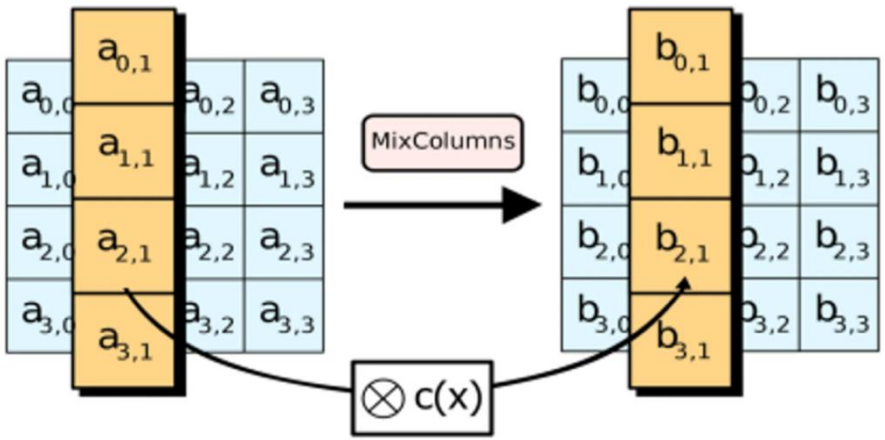
Plaintext

Round

- SubBytes(State)
- ShiftRows(State)
- MixColumns(State)
- AddRoundKey(State,Key)

MixColumns(State)

- Multiply columns by constant matrix



Adapted from: Mao, Wenbo. "Modern cryptography: theory and practice." (2004). Images from: https://www.educative.io/edpresso/what-is-the-aes-algorithm

Advanced Encryption Standard (AES) uses the concept of an S-P network

Simplified

128/192/256 bit key

Key expanded into 10, 12, or 14 round keys using key schedule algorithm

Key_0

Plaintext

Round

SubBytes(State)

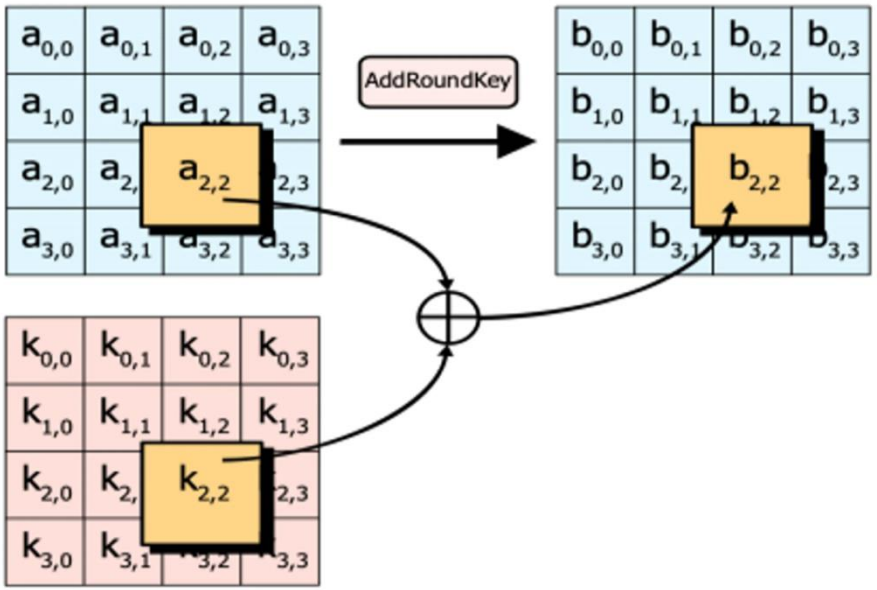
ShiftRows(State)

MixColumns(State)

AddRoundKey(State,Key)

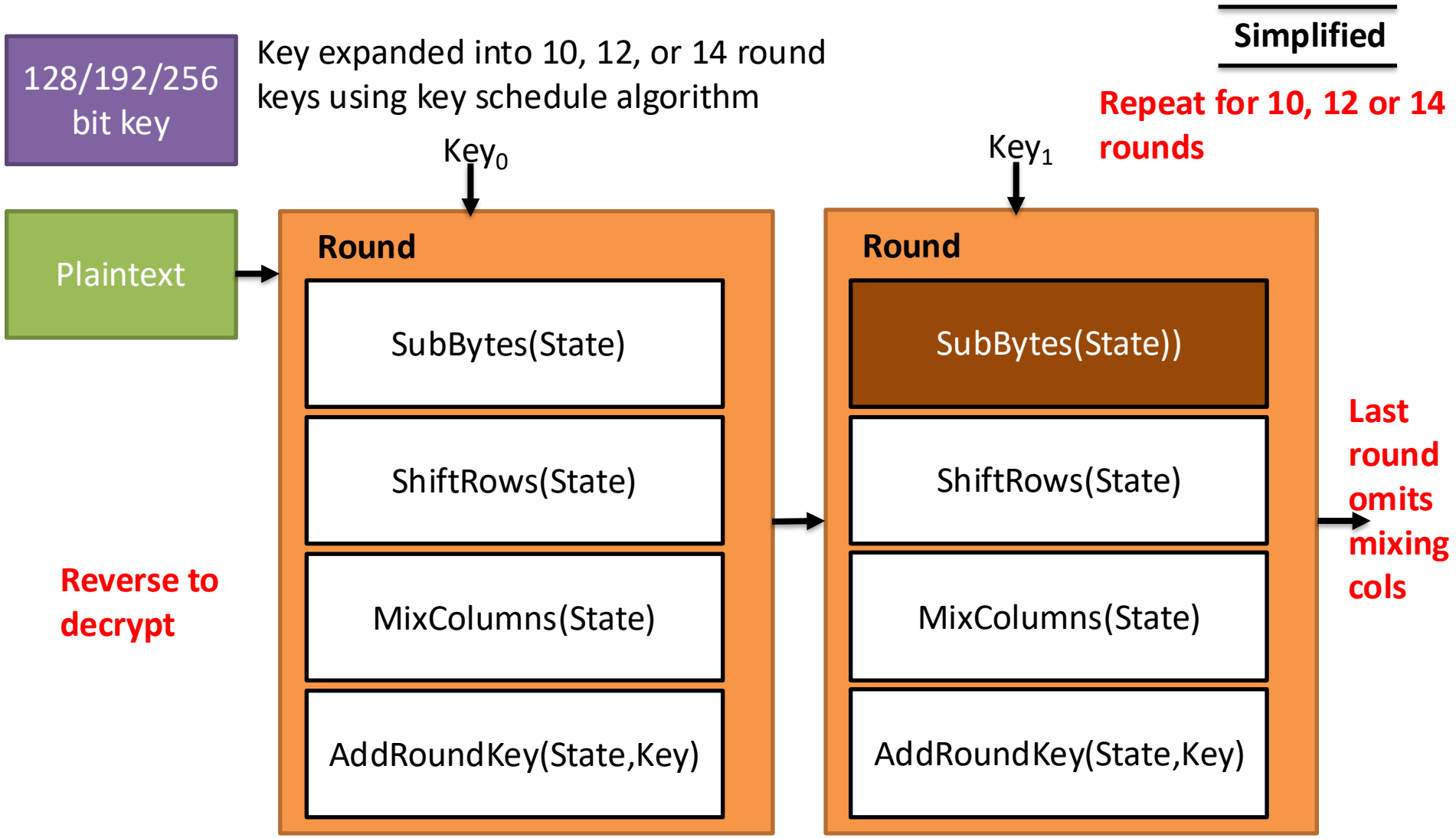
AddRoundKey(State, Key)

- XOR message with round key



Adapted from: Mao, Wenbo. "Modern cryptography: theory and practice." (2004). Images from: https://www.educative.io/edpresso/what-is-the-aes-algorithm

Advanced Encryption Standard (AES) uses the concept of an S-P network



Adapted from: Mao, Wenbo. "Modern cryptography: theory and practice." (2004).

A brief history of modern encryption: DES and AES

Data Encryption Standard (DES)

- Developed by US government in 1970s from original IBM proposal
- Uses 56-bit keys (NSA reduced keys size by half….)
- Block cipher – works on blocks of 64 bits vs. stream ciphers which work bit-by-bit
- RSA Security sponsored series of DES challenges
  - Broken in 1997 by using idle cycles of machines on the Internet
  - In 1998 Electronic Freedom Foundation (EFF) built special-purpose Deep Crack which could crack DES messages in 56 hours



Solution: Triple DES

- Apply DES three times with different keys each time
- Key size now 3 * 56 = 168 bits, but slow to compute

Cryptography – creating codes

Cryptanalysis – breaking codes

Better solution: Advanced Encryption Standard (AES) in 2000