

# CT437 COMPUTER SECURITY AND FORENSIC COMPUTING

## DEFINITIONS, TERMINOLOGY, AND CASE STUDIES

Dr. Michael Schukat



# What is Cybersecurity?

2

- Computer security, cybersecurity or information technology security (IT security) is the protection of computer systems and networks from the theft of or damage to their hardware, software, or electronic **data**, as well as from the **disruption or misdirection of the services they provide** (Wikipedia), i.e.:
- Protection from cybercrime of
  - ▣ data (from theft or manipulation)
  - ▣ services (from disruption or misuse)
- This protection can be on a personal, organisational or government level

# States of Data

- Data at rest
  - ▣ Rest refers to data stored in memory or on a permanent storage device such as a hard drive, solid-state drive or USB drive
- Data in process
  - ▣ Processing refers to data that is being used to perform an operation such as updating a database record
- Data in transit
  - ▣ Transmission refers to data traveling between information systems, e.g. data transfer over a network via TCP/IP

# How to provide Protection?

4

- **Awareness, training and education** are the measures put in place by an organisation to ensure that users are knowledgeable about potential security threats and the actions they can take to protect information systems
- **Technology** refers to the software and hardware-based solutions designed to protect information systems such as firewalls, which continuously monitor your network in search of possible malicious incidents
- **Policy and procedure** refers to the administrative controls that provide a foundation for how an organization implements information assurance, such as incident response plans and best practice guidelines

# Defense in Depth

5

- Defense in Depth (DiD) is an approach to cybersecurity in which a series of defensive mechanisms are layered in order to protect assets
- If one mechanism fails, another one steps up immediately to thwart an attack



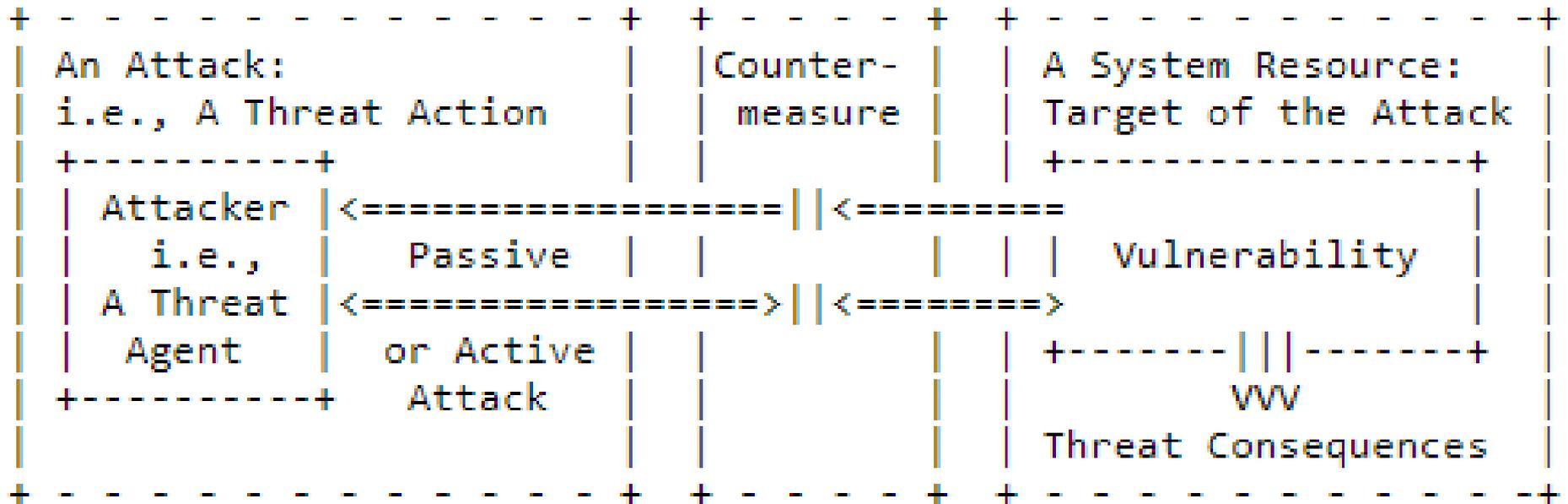
# European Union Agency for Cybersecurity (ENISA)

6

- ENISA is the Union's agency dedicated to achieving a high common level of cybersecurity across Europe
- <https://www.enisa.europa.eu/>
- ENISA threat landscape report: <https://www.enisa.europa.eu/topics/cyber-threats/threat-landscape>
- ENISA has also issued a 2024 report *providing policy makers at EU level with an evidence-based overview of the state of play of the cybersecurity landscape and capabilities at the EU, national and societal levels, as well as with policy recommendations to address identified shortcomings and increase the level of cybersecurity across the Union current threat landscape (see also Canvas)*

# The big Picture – RFC2828

- RFC2828, Internet Security Glossary
- <https://tools.ietf.org/html/rfc2828>



# What is a Threat Agent/Actor?

8

- The term *threat agent* or *threat actor* is used to indicate an individual, thing or a group that can manifest a threat
  - ▣ In computer security, a threat is a potential negative action or event facilitated by a vulnerability that results in an unwanted impact to a computer system or application
- Threat actors include:
  - ▣ Non-target specific, e.g. computer viruses, worms, trojans and logic bombs.
  - ▣ Employees, e.g. disgruntled staff or contractors
  - ▣ Organized crime and criminals
  - ▣ Corporations, e.g. partners or competitors
  - ▣ Human, unintentional (including accidents and carelessness)
  - ▣ Human, intentional
  - ▣ Natural, e.g. flood, fire, lightning, meteor, earthquakes

# Hackers

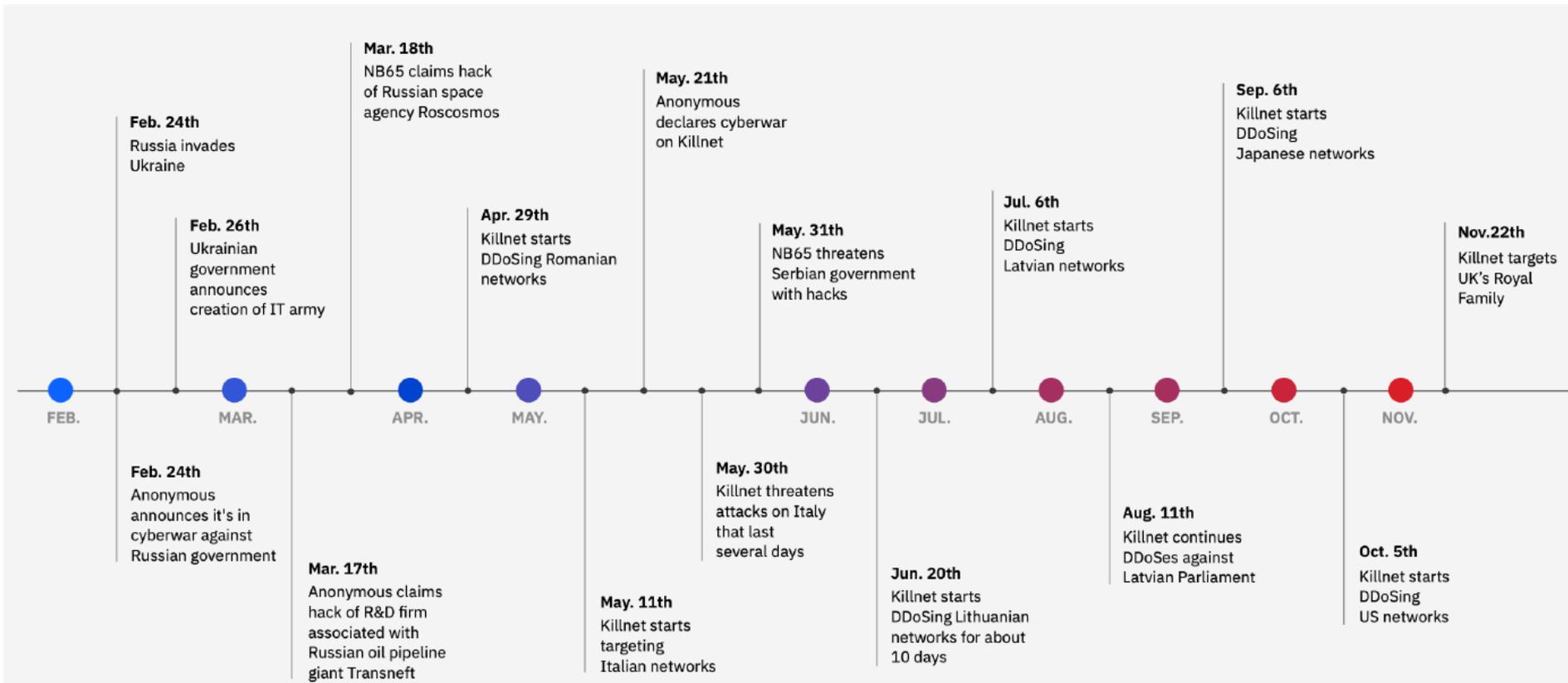
- Threat actors that break into computer systems or networks to gain access:
  - **White hat hackers** break into networks or computer systems to identify any weaknesses so that the security of a system or network can be improved. These break-ins are done with prior permission and any results are reported back to the owner
  - **Black hat hackers** take advantage of any vulnerability for illegal personal, financial or political gain
  - **Gray hat hackers** may set out to find vulnerabilities in a system without prior permission of the owner. When they uncover weaknesses, they do not exploit them, rather they report them, but they may demand payment in return
- Unskilled hackers are called **script kiddies**; they use scripts or programs developed by others, primarily for malicious purposes

# Hacktivists

10

- ❑ Hacktivists make political statements to create awareness about issues that are important to them
- ❑ In 2022, a significant increase in hacktivist activity has been observed, especially since the start of Russia-Ukraine conflict
- ❑ Target organisations through DDoS attacks, defacements and data leaks
- ❑ Some of the major Hacktivist groups include Anonymous, TeamOneFirst, GhostSec, Against the West, NB65, KILLNET, XakNet, and The Red Bandits

# Timeline of selected Hactivist Events 2022



Timeline of select hactivist events 2022

Source: IBM Security X-Force Threat Intelligence Index 2023

# Cyber Criminals

12

- Cyber criminals are usually highly sophisticated and organised
- Main interest in attacks that usually lead to ransomware deployment, coin mining, stealing cryptocurrency, or stealing credentials
- They may even provide cybercrime as a service to other criminals, aka hacker-for hire within the 'Access-as-a-Service' (AaaS) market

# State Sponsored Threat Actors

13

- ❑ State-sponsored attackers gather intelligence or commit sabotage on behalf of their government
- ❑ They are usually highly trained and well-funded
- ❑ Their attacks are focused on specific goals that are beneficial to their government
- ❑ Mostly involved in destructive or disruptive operations
- ❑ Example: Since the start of the Russia-Ukraine conflict, widespread use of wiper malware attacks to destroy and disrupt networks of governmental agencies and critical infrastructure entities have been observed

# Cyber Warfare

14

- Cyberwarfare is the use of technology to penetrate and attack another nation's computer systems and networks to cause damage or disrupt critical services
- The main reason for resorting to cyberwarfare is to gain advantage over adversaries
  - ▣ Industrial and military espionage e.g. steal defence secrets and gather information about technology
  - ▣ Impact infrastructure e.g. power grid
- Example Stuxnet

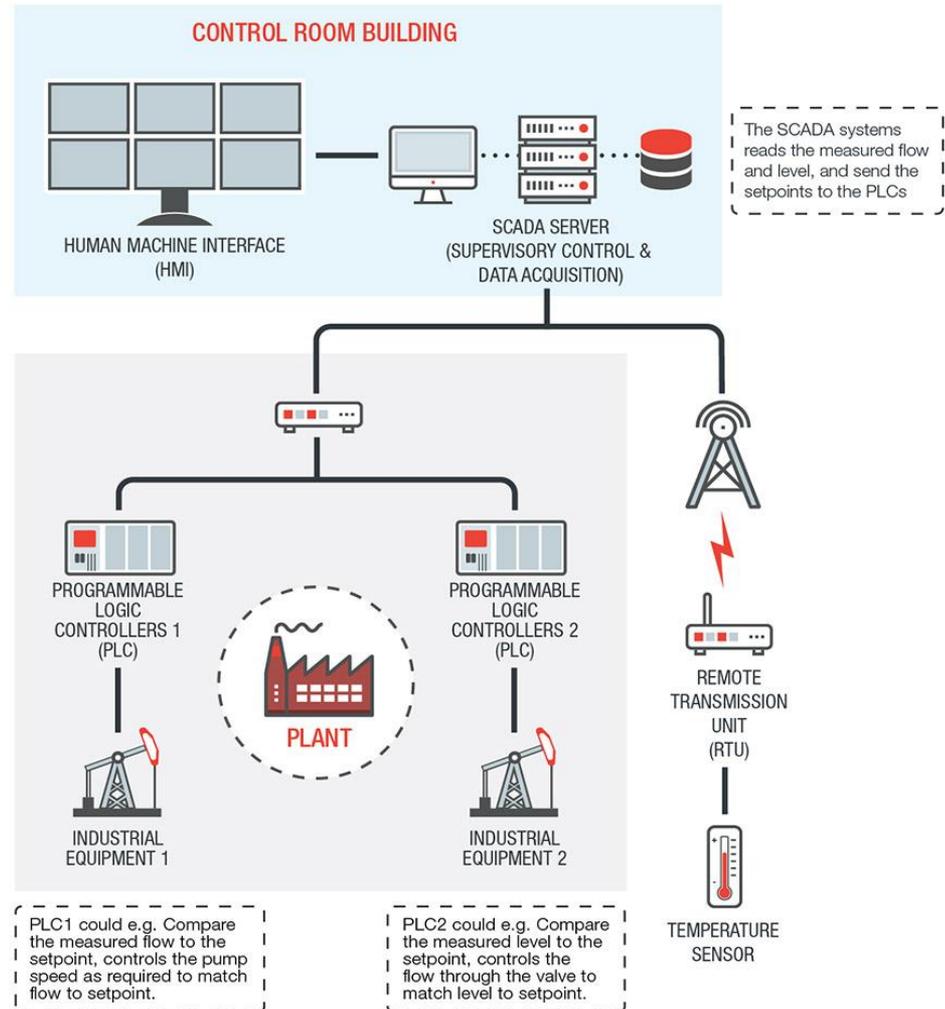
15

# Some Case Studies

# Background: Industrial Control Systems (ICS)

16

- An ICS is an electronic control system and associated instrumentation used for industrial process control
- Control systems can range in size from a few modular panel-mounted controllers to large interconnected and interactive distributed control systems (DCSs) with many thousands of field connections
- Control systems receive data from remote sensors measuring process variables (PVs), compare the collected data with desired setpoints (SPs), and derive command functions that are used to control a process through the final control elements (FCEs), such as control valves



# Cyberattacks on ICS

17

- Traditionally relatively “niche”, but potentially high-impact attacks on critical infrastructure
  - ▣ Power generation
  - ▣ Chemical plants
  - ▣ Steel mills
  - ▣ Transport systems
  - ▣ Oil pipelines
- A summary of some recent high-profile attacks can be found here:
  - ▣ <https://www.makeuseof.com/cyberattacks-on-industry-hackers/>

# Attacking Critical Infrastructure (Water Supply)

- [https://edition.cnn.com/2021/02/08/us/oldsmar-florida-hack-water-poison/index.html?utm\\_source=twCNN&utm\\_term=link&utm\\_medium=social&utm\\_content=2021-02-09T02%3A55%3A27](https://edition.cnn.com/2021/02/08/us/oldsmar-florida-hack-water-poison/index.html?utm_source=twCNN&utm_term=link&utm_medium=social&utm_content=2021-02-09T02%3A55%3A27)

## Someone tried to poison a Florida city by hacking into the water treatment system, sheriff says

By Amir Vera, Jamiel Lynch and Christina Carrega, CNN

🕒 Updated 0407 GMT (1207 HKT) February 9, 2021



Pinellas County Sheriff Bob Gualtieri speaks at a press conference on Monday, February 8, about the attempted hacking of the city of Oldsmar's water treatment system.

**(CNN)** — A hacker gained access into the water treatment system of Oldsmar, Florida, on Friday and tried to increase the levels of sodium hydroxide -- commonly referred to as lye -- in the city's water, officials said, putting thousands at risk of being poisoned.

# Cyberattacks on Industrial Infrastructure

19

## German Steel Plant Suffers Significant Damage from Targeted Attack

12 janvier 2015

An unknown number of attackers knowledgeable in IT security and industrial control systems (ICS) processes have caused massive damage to a German steel plant in 2014. The incident has been confirmed by the Federal Office for Information Security (BSI) of the German government in an [IT security report](#).



The attack, which appeared to specifically target operators of industrial plants, caused components of the plant controls to fail, resulting in an unregulated furnace, which then caused physical damage to the steel plant.

The individual or group responsible for the attack was able to infiltrate the system using spear phishing and social engineering techniques. These two methods are proven ways by which threat actors lure their victims using emails or social media links that appear to come from a legitimate source but can actually introduce threats for attackers to get inside the network.

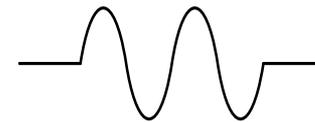
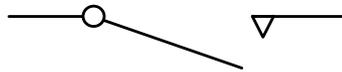
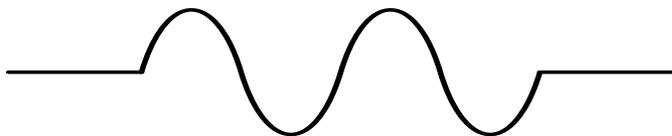
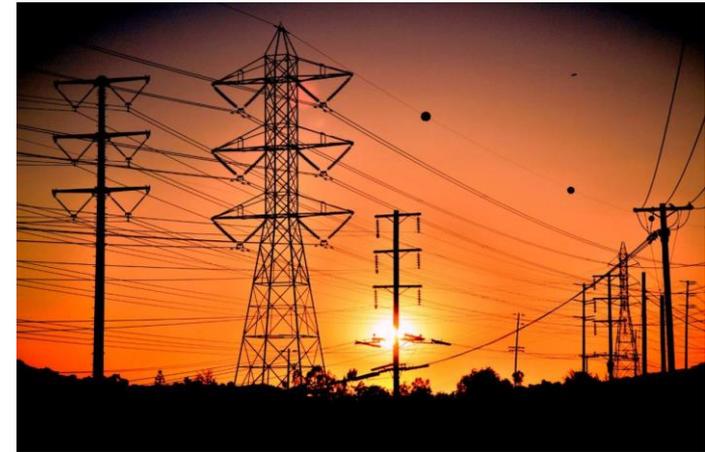
A number of news reports have dubbed this the second cyber attack to ever cause physical damage since the highly sophisticated [Stuxnet malware](#) wreaked havoc to the Natanz uranium enrichment plant in Iran. However, attacks affecting real-world operations of facilities have been

□ <https://www.trendmicro.com/vinfo/fr/security/news/cyber-attacks/german-steel-plant-suffers-significant-damage-from-targeted-attack>

# Attacking Critical Infrastructure (Energy Systems) - Synchroscope

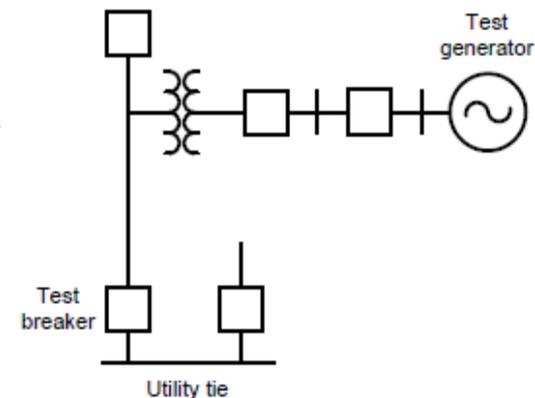
20

- In an electrical grid, power generators and distribution networks must be synced with regard to AC power frequency and phase
- Connecting two unsynchronized AC power systems together is likely to cause high currents to flow, which will severely damage any equipment



# The AURORA Cyberattack

- ❑ Experiment conducted by U.S. Department of Energy's Idaho laboratory
- ❑ A hacker gained remote access to a Diesel generator's control system, see [Video](#),
  - ▣ rapidly open and close a diesel generator's circuit breakers, causing it to become out of sync with the transmission network
  - ▣ thereby subjecting the engine to abnormal torques and ultimately causing it to explode
    - high electrical torque translates to stress on the mechanical shaft of the rotating equipment



# Example Stuxnet

22

- ❑ Stuxnet was a powerful computer worm designed by U.S. and Israeli intelligence around 2009 designed to disable a key part of the Iranian nuclear program
- ❑ It targeted the “air-gapped” nuclear facility at Natanz
- ❑ Stuxnet was designed to destroy the centrifuges Iran was using to enrich uranium as part of its nuclear program



# Stuxnet

## Software Sabotage

How Stuxnet disrupted Iran's uranium enrichment program

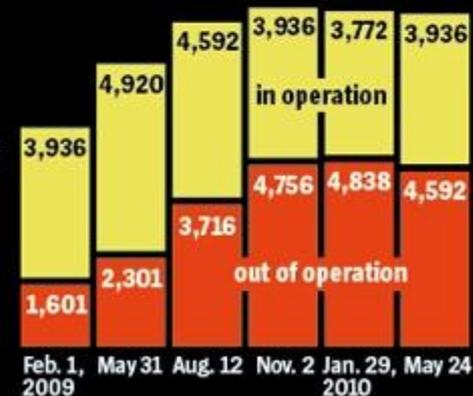
**1** The malicious computer worm probably entered the computer system - which is normally cut off from the outside world - at the uranium enrichment facility in Natanz via a removable USB memory stick.

**2** The virus is controlled from servers in Denmark and Malaysia with the help of two Internet addresses, both registered to false names. The virus infects some 100,000 computers around the world.

**3** Stuxnet spreads through the system until it finds computers running the Siemens control software. Step 7, which is responsible for regulating the rotational speed of the centrifuges.

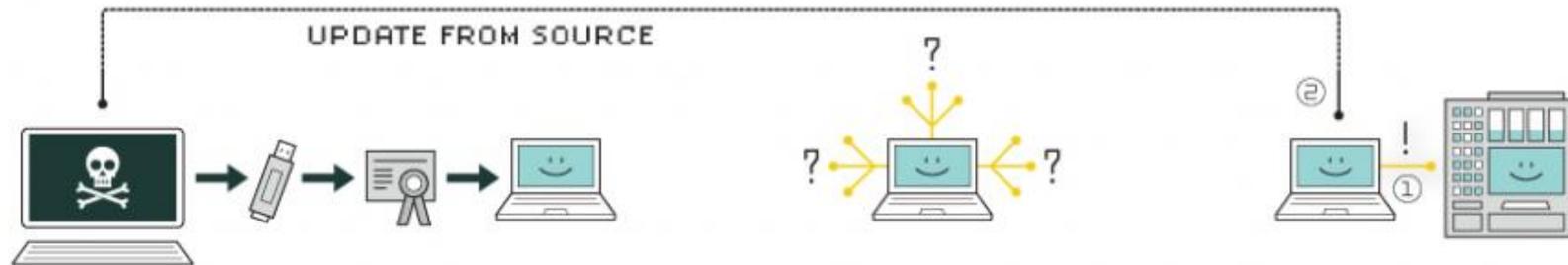
**4** The computer worm varies the rotational speed of the centrifuges. This can destroy the centrifuges and impair uranium enrichment.

**5** The Stuxnet attacks start in June 2009. From this point on, the number of inoperative centrifuges increases sharply.



# Stuxnet Anatomy

24



## 1. infection

Stuxnet enters a system via a USB stick and proceeds to infect all machines running Microsoft Windows. By brandishing a digital certificate that seems to show that it comes from a reliable company, the worm is able to evade automated-detection systems.

## 2. search

Stuxnet then checks whether a given machine is part of the targeted industrial control system made by Siemens. Such systems are deployed in Iran to run high-speed centrifuges that help to enrich nuclear fuel.

## 3. update

If the system isn't a target, Stuxnet does nothing; if it is, the worm attempts to access the Internet and download a more recent version of itself.



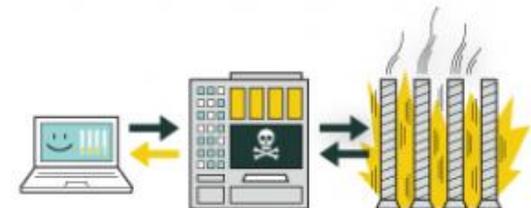
## 4. compromise

The worm then compromises the target system's logic controllers, exploiting "zero day" vulnerabilities—software weaknesses that haven't been identified by security experts.



## 5. control

In the beginning, Stuxnet spies on the operations of the targeted system. Then it uses the information it has gathered to take control of the centrifuges, making them spin themselves to failure.



## 6. deceive and destroy

Meanwhile, it provides false feedback to outside controllers, ensuring that they won't know what's going wrong until it's too late to do anything about it.

# Cyberattacks on Connected Cars

25

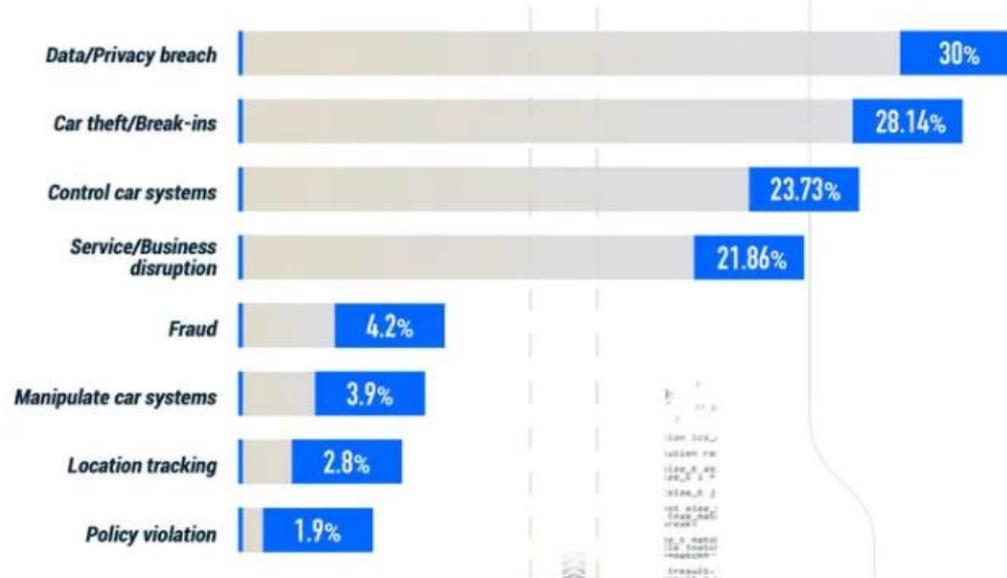
- <https://www.darkreading.com/attacks-breaches/cybercriminals-take-aim-at-connected-car-infrastructure>
- DEF CON 27: Car Hacking Deconstructed: <https://www.youtube.com/watch?v=gzav1K5KSl4>



Robert Lemos

Contributing Writer

October 29, 2021



Impact of attacks on automakers over the past decade.

Source: Upstream's "Global Automotive Cybersecurity Report 2021"

# Cyberattacks on Medical Devices

- Many medical devices are connected to the Internet, or have a wireless interface
- This allows remote attacks, see for example
  - ▣ <https://www.youtube.com/watch?v=THpcAd2nWJ8>

## Hacker Shows Off Lethal Attack By Controlling Wireless Medical Device

BY JORDAN ROBERTSON | FEB. 29, 2012 10:00 AM EDT | POSTED IN HACKERS, MEDICAL PRIVACY, POSTS, SECURITY, VIDEO | 15 COMMENTS



Photographer: David Paul Morris/Bloomberg

Barnaby Jack uses a mannequin equipped with an insulin pump to show the vulnerabilities of wireless medical devices.

# What is a Vulnerability?

27

- A weakness which can be exploited by a threat actor/agent (an attacker) to cross privilege boundaries (i.e. perform unauthorised actions) within a computer system (Wikipedia)
- A flaw or weakness in a system's design, implementation, or operation and management that could be exploited to violate the system's security policy (RFC2828)
- A weakness in the computational logic (e.g., code) found in software and some hardware components (e.g., firmware) that, when exploited, results in a negative impact to confidentiality, integrity, OR availability (<https://cve.mitre.org/>)
- Vulnerabilities can be researched, reverse-engineered, hunted, or exploited using automated tools or customized scripts
- An exploitable vulnerability is one for which at least one working attack or exploit exists

# Hardware Vulnerabilities

28

- ❑ Hardware vulnerabilities are usually the result of hardware design flaws
- ❑ Example DRAM:
  - ▣ DRAM memory requires one capacitor per bit (a capacitor is a component which can hold an electrical charge)
  - ▣ Modern DRAM chips have a very high memory capacity (4 – 32 gigabits) resulting in those capacitors being positioned installed very close to one another
  - ▣ However, it was discovered that due to their close proximity, changes applied to one of these capacitors could influence neighbouring capacitors
  - ▣ Based on this design flaw, an exploit called **Rowhammer** was created
  - ▣ By repeatedly accessing (hammering) a row of memory, the Rowhammer exploit triggers electrical interferences that eventually corrupt the data stored inside the RAM

# Software Vulnerabilities

29

- Software vulnerabilities are usually introduced by errors in the operating system or application code
- Bug: An error that can be rooted to the source code, e.g.
  - ▣ Incorrect implementation of a security protocol
  - ▣ Buffer Overflow: When a program writes more data to a buffer than it can hold, potentially leading to arbitrary code execution
    - Example TLS Heartbleed (will be covered later and in an assignment)
- Flaw: An error at a much deeper level, particularly in the design, and likely in the code level, which may be very difficult and costly to correct; e.g.
  - ▣ Lack of security features, i.e. data encryption, to protect sensitive application data from unauthorised access

# Example: Apple's 'goto fail;' Bug in TLS 1.0 and TLS 1.1 (2014)

30

- ❑ Affected iOS and Mac OS X operation systems
- ❑ This vulnerability allowed attacks on TLS connections

```
if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;

err = sslRawVerify(ctx,
                  ctx->peerPubKey,
                  dataToSign,
                  dataToSignLen,
                  signature,
                  signatureLen);
/* plaintext */
/* plaintext length */

if(err) {
    sslErrorLog("SSLDecodeSignedServerKeyExchange: sslRawVerify "
               "returned %d\n", (int)err);
    goto fail;
}

fail:
SSLFreeBuffer(&signedHashes);
SSLFreeBuffer(&hashCtx);
return err;
```

# The Common Vulnerabilities and Exposures (CVE) Database

31

- CVE (<https://cve.mitre.org/>) is a central repository of all the reported security vulnerabilities associated with a specific software system
- Each CVE entry has a unique identifier which is commonly used by many commercial vulnerability management systems to refer to a specific software vulnerability, e.g.,
  - ▣ Heartbleed, CVE-2014-0160
  - ▣ “goto fail;”, CVE-2014-1266
- CVE ID Syntax: CVE prefix + Year + Arbitrary Digits

# The Common Weakness Enumeration (CWE) Database

32

- ❑ CVE is complemented by CWE (<https://cwe.mitre.org/>)
- ❑ It provides a formal list of software weakness types that serve as a common language for describing software security weaknesses in architecture, design, or code, for example:
  - ❑ Unrestricted upload of files
  - ❑ Improper input validation
  - ❑ Out-of-bound writes (in arrays)
- ❑ CWE describes a generic vulnerability, while CVE has to do with the specific instance within a product or system not the underlying flaw

# Exploit (Wikipedia)

33

- An exploit is a piece of software, data, or a sequence of commands that takes advantage of a vulnerability to cause unintended or unanticipated behaviour to occur on computer software or hardware
- Such behaviour frequently includes things like gaining control of a computer system, allowing privilege escalation, or a denial-of-service attack
- A **remote exploit** works over a network and exploits the security vulnerability without any prior access to the vulnerable system
- A **local exploit** requires prior access to the vulnerable system and usually increases the privileges of the person running the exploit past those granted by the system administrator
- A **zero-day exploit** takes advantage of a vulnerability in software, hardware, or firmware that is unknown to the vendor and for which no patch or fix is available

# Heartbleed Exploit Extract (Python Code)

34

□ <https://gist.github.com/eelsivart/10174134>

Heartbleed (CVE-2014-0160) Test & Exploit Python Script

```
heartbleed.py Raw
1  #!/usr/bin/python
2
3  # Modified by Travis Lee
4  # Last Updated: 4/21/14
5  # Version 1.16
6  #
7  # -changed output to display text only instead of hexdump and made it easier to read
8  # -added option to specify number of times to connect to server (to get more data)
9  # -added option to send STARTTLS command for use with SMTP/POP/IMAP/FTP/etc...
10 # -added option to specify an input file of multiple hosts, line delimited, with or without a port specified (host:port)
11 # -added option to have verbose output
12 # -added capability to automatically check if STARTTLS/STLS/AUTH TLS is supported when smtp/pop/imap/ftp ports are entered and automatically
13 # -added option for hex output
14 # -added option to output raw data to a file
15 # -added option to output ascii data to a file
16 # -added option to not display returned data on screen (good if doing many iterations and outputting to a file)
17 # -added tls version auto-detection
18 # -added an extract rsa private key mode (orig code from epixoip. will exit script when found and enables -d (do not display returned data
19 # -requires following modules: gmpy, pyasn1
20
21 # Quick and dirty demonstration of CVE-2014-0160 by Jared Stafford (jspenguin@jspenguin.org)
22 # The author disclaims copyright to this source code.
23
24 import sys
25 import struct
26 import socket
27 import time
28 import select
29 import re
```

# Attack Surface and Attack Vector

35

- An organisation's attack surface is the sum of all its attack vectors, i.e., vulnerabilities, pathways and methods, that hackers can use to gain unauthorised access to a network or sensitive data, or to carry out a cyberattack
- The smaller the attack surface, the easier it is to protect the system (obviously)
- We distinguish between the
  - ▣ digital attack surface,
  - ▣ physical attack surface (e.g. malicious insiders or device theft),
  - ▣ social engineering attack surface (e.g. phishing)

# The Digital Attack Surface

36

- This includes:
  - Weak passwords, i.e., passwords that are easy to guess or easy to crack via brute-force attacks
  - Misconfiguration, e.g., improperly configured network ports or wireless access points
  - Software, operating system and firmware vulnerabilities
  - Outdated or obsolete devices, data, or applications

# Social Engineering (Recall CT255)

37

- Social engineering is the manipulation of people into performing certain actions or revealing confidential information
- That information might be a password, credit card information, personally identifiable information, confidential data, or anything that can be used for fraudulent acts like identity theft
- There are different types of social engineering attacks:
  - ▣ Pretexting
  - ▣ Phishing
  - ▣ Smishing
  - ▣ Vishing
  - ▣ Tailgating

# Pretexting

38

- This is when an attacker calls an individual and lies to them in an attempt to gain access to privileged data
- Example:



A fraudster **impersonates a trusted authority** and crafts a scenario to reach out to their victims.



The victim **believes the scenario** and shares any information the 'trusted' authority requests.



The fraudster **gains valuable information** from their victim and often uses it maliciously.

# Phishing

39

- ❑ **Phishing** involves sending malicious emails from supposed trusted sources to as many people as possible, assuming a low response rate (shotgun method)
- ❑ In **spear phishing** the perpetrator is disguised as a trusted individual (boss, friend, spouse)
- ❑ **Whaling** uses deceptive email messages targeting high-level decision makers within an organisation, such as CEOs and other executives, who have access to highly valuable information

**Subject: Shhh it's a surprise!**

Clare,

We're planning a virtual baby shower for Amy in financial services and are in a time crunch. Could you buy the gift? Please **share your banking information** and we can transfer the money over.

Thanks,  
Larry Scamington, HR director

# Smishing

40

- Smishing is phishing by SMS or text messaging
- This can be a trusty avenue for pretexting attackers to connect with victims since texting is a more intimate form of communication

Text Message  
Thu, 31 Aug, 12:57

AIB: Due to unusual activity, your card has been placed on hold. Please visit [aibinfo8.com](https://aibinfo8.com) and follow the on-screen instructions to re-activate.

Text Message  
Wed, 20 Sep, 10:18

AnPost: Your package has a €2.38 pending fee. To pay this visit: [anpost-post-servicecharge.com](https://anpost-post-servicecharge.com) If this is not paid the package will be returned to sender.

Text Message  
Thu, 21 Sep, 12:05

AnPost: You've missed our delivery, for the redelivery of your parcel please visit: [anpost-delivery-notice.com](https://anpost-delivery-notice.com) and confirm the settlement of €2.38

Text Message  
Sat, 23 Sep, 19:31

MyGov: Pre-approved 2023 tax repayment available. Follow <https://incometaxcreditrevenue-mygov.com/ie> to verify information. Review may take up to 14 days.

# Vishing

41

- It is the voice counterpart to phishing, e.g.
  - ▣ An email message asks the user to make a telephone call
  - ▣ Victims receive an unsolicited call
- Fraudsters might spoof, or fake caller IDs or use AI generated deepfakes to convince victims they are a trusted source and, ultimately, get victims to share valuable information over the phone
- Many different variations, see for example
  - ▣ <https://www.youtube.com/watch?v=PWVN3Rq4gzw>
  - ▣ <https://www.youtube.com/watch?v=lc7scxvKQOo>

# Tailgating

42

- This is when
  - ▣ an attacker quickly follows an authorized person into a secure, physical location
  - ▣ fraudsters pose in real-life as someone else to gain access to restricted or confidential areas where they can get their hands on valuable information

An “internet service provider” shows up on your doorstep for a routine check. Once inside, they have free reins to snoop through your devices and valuable information.



## TIP

If a service provider arrives without an appointment, don't just let them inside. Verify their legitimacy by asking questions about your plan.

# Quid Pro Quo

43

- “Get something for doing something” (latin: quid pro quo)
- This is when an attacker requests personal information from a person in exchange for something, like a free gift

Congratulations, Steve!

You're eligible for a \$5,000 gift card.  
To redeem, please **share your banking information** for wire transfer and also your **home address**.

Sincerely,  
Notareal Co.

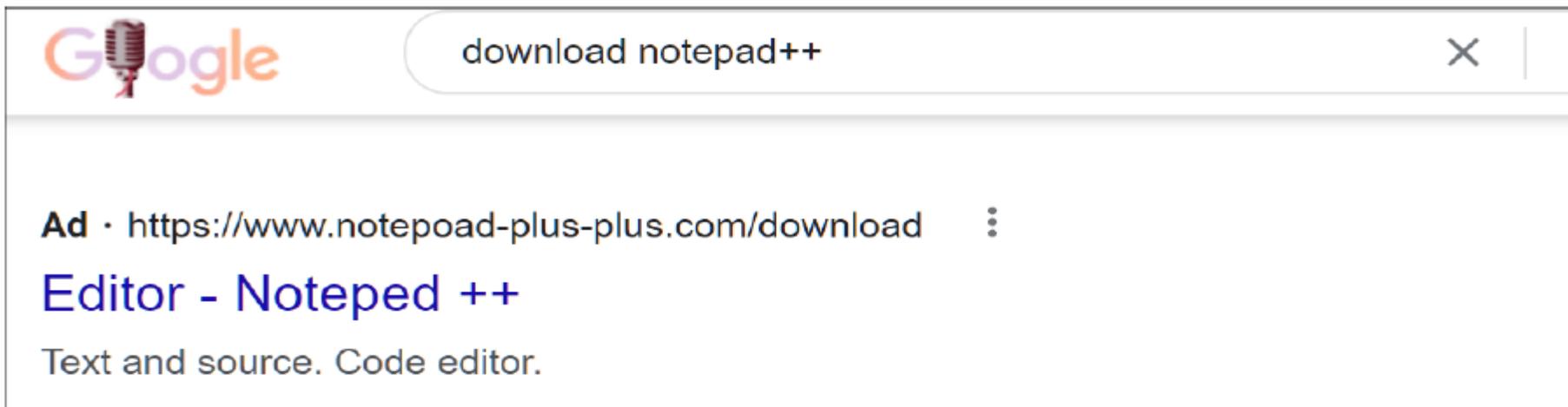
# SEO Poisoning

- ❑ Search engine optimisation (SEO) is about improving an organisation's website visibility in search engine results
- ❑ Search engines such as Google present a list of web pages to users based on their search query. These web pages are ranked according to the relevancy of their content.
- ❑ SEO poisoning is a technique used by threat actors to increase the prominence of their malicious websites, making them look more authentic to consumers
- ❑ The most common goal of SEO poisoning is to increase traffic to malicious sites that may host malware or attempt social engineering

# Typosquatting

45

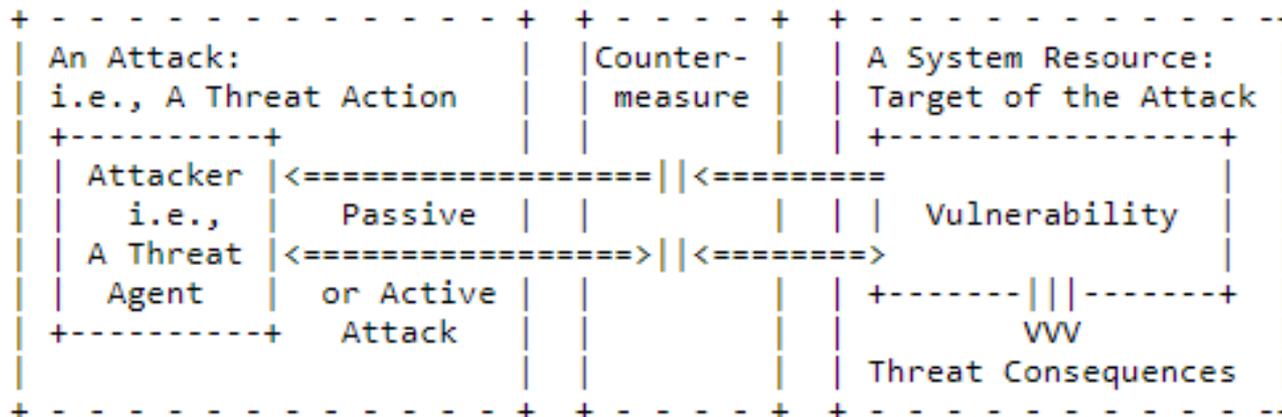
- ❑ Typosquatting targets users who might open their
- ❑ browser and input a website address that has an inadvertent typo or click on a link with a misspelled URL
- ❑ § To exploit these minor user errors, attackers register domain names similar to legitimate ones



# Attack (RFC2828, Internet Security Glossary)

46

- An attack is an assault on system security that derives from an intelligent threat, i.e. a deliberate attempt
- An "active attack" attempts to alter system resources or affect their operation (e.g., a ransomware attack on a file server)
- A "passive attack" attempts to learn or make use of information from the system, but does not affect system resources (e.g., eavesdropping on an unprotected network connection)



# Common Attack Types

47

- ❑ Malware
  - ❑ E.g., in a ransomware attack, an adversary encrypts a victim's data and offers to provide a decryption key in exchange for a payment
- ❑ Denial-of-Service (DoS)
  - ❑ A malicious, targeted attack that floods a network with false requests in order to disrupt business operations
- ❑ Phishing
  - ❑ A type of cyberattack that uses email, SMS, phone, social media, and social engineering techniques to entice a victim to share sensitive information
- ❑ Spoofing
  - ❑ A technique through which a cybercriminal disguises themselves as a known or trusted source
- ❑ Code injection attacks
  - ❑ An attacker injecting malicious code into a vulnerable computer or network to change its course of action (e.g. XSS and SQL injection)

# Countermeasures (RFC2828, Internet Security Glossary)

48

- An action, device, procedure, or technique that
  - ▣ ... reduces a threat, a vulnerability, or an attack
  - ▣ ... by eliminating or preventing it,
  - ▣ ... by minimising the harm it can cause, or
  - ▣ ... by discovering and reporting it so that corrective action can be taken
- For example, a firewall filters unsolicited network traffic

# Threat Consequences

49

- ❑ Threat consequences (as a result from an attack) include
  - ❑ disclosure of information
  - ❑ deception,
  - ❑ disruption of services
  - ❑ usurpation, e.g. unauthorized control of some part of a system

- ❑ Cybercrime: it's all around us
  - Posing a major threat to personal and organizational data and even national security



Personal level

Your identity, data, and computing devices



Organizational level

Reputation, data and customers



Government level

National security, economy and the safety of citizens

# Vulnerability Testing

50

- ❑ Vulnerability testing is a process of evaluating and identifying security weaknesses in a computer system, network, or software application
- ❑ It involves systematically scanning, probing, and analyzing systems and applications to uncover potential vulnerabilities, such as coding errors, configuration flaws, or outdated software components
- ❑ The main goal of vulnerability testing is to discover and address these security gaps before they can be exploited by attackers

# Vulnerability Testing

51

- ❑ **Network-based scanning:** Used to scan networks for open ports, misconfigurations, and other security weaknesses
- ❑ **Web application scanning:** Identify vulnerabilities in web applications, such as SQL injection, cross-site scripting (XSS), and broken authentication
- ❑ **Static application security testing (SAST):** Analyse source code or compiled code to identify potential security vulnerabilities without executing the application
- ❑ **Dynamic application security testing (DAST):** Interact with running applications to identify security weaknesses during runtime
- ❑ **Fuzz testing:** Generate and send malformed or unexpected inputs to applications to identify vulnerabilities related to input validation and error handling
- ❑ **Database vulnerability assessment:** Scan the database management systems for any potential security weaknesses, misconfigurations, or other vulnerabilities that could be exploited
- ❑ **Configuration management and compliance assessment:** Assess system and application configurations against established security best practices or compliance standards
- ❑ **Container and cloud security assessment:** Focus on identifying vulnerabilities and misconfigurations in cloud-based environments and containerized applications

# Vulnerability Testing Steps

52

## Asset discovery



Detect and manage local and remote endpoints, roaming devices, and closed network (DMZ) devices

## Vulnerability scanning



Spot all OS vulnerabilities, third-party vulnerabilities, and zero-day vulnerabilities.

## Vulnerability assessment



Understand the impact of threats, and prioritize vulnerabilities based on severity, age, exploit code disclosure, patch availability, and various infographics for timely risk reduction.

## Vulnerability remediation



Deploy automatically correlated patches to seal vulnerabilities, and leverage alternative mitigation measures if no patch is available.

# Example Nessus: An automatic Network Vulnerability Scanner

53

**nessus Professional** Scans Settings admin

## Basic network Scan

[Back to My Scans](#)

Hosts: 112 | Vulnerabilities: 272 | Remediations: 500 | VPR Top Threats

Filter Search Hosts 112 Hosts

Host	Vulnerabilities
192.168.1.46	147 (Critical), 278 (High), 59 (Medium), 189 (Info)
192.168.1.83	60 (Critical), 333 (High), 86 (Medium), 184 (Info)
192.168.1.10	42 (Critical), 320 (High), 81 (Medium), 186 (Info)
192.168.1.53	28 (Critical), 48 (Medium), 508 (Info)
192.168.1.44	39 (Critical), 293 (High), 78 (Medium), 169 (Info)
192.168.1.66	22 (Critical), 228 (High), 52 (Medium), 174 (Info)
192.168.1.55	113 (Critical), 172 (High), 29 (Medium), 130 (Info)
192.168.1.40	65 (Critical), 154 (High), 89 (Medium), 56 (Info)
192.168.1.56	48 (Critical), 166 (High), 41 (Medium), 66 (Info)
192.168.1.11	15 (Critical), 87 (High), 178 (Info)
192.168.1.12	15 (Critical), 87 (High), 177 (Info)
data.tehgeek.local	12 (Medium), 266 (Info)
sshsrvr.tehgeek.local	26 (Critical), 16 (Medium), 225 (Info)

**Scan Details**

Policy: Basic Network Scan  
Status: Imported  
Severity Base: CVSS v3.0  
Modified: April 1 at 1:00 PM (Live Results)

**Vulnerabilities**

- Critical
- High
- Medium
- Low
- Info

# Outlook Assignment 1

54

- ❑ In assignment 1 you will be doing a manual (non-automated) vulnerability analysis of a VM target, using Metasploit, focusing on a small number of exploits
- ❑ Metasploit is a
  - ▣ widely-used open-source framework for developing, testing, and executing exploits against target systems
  - ▣ powerful tool for penetration testing, enabling security professionals to identify and exploit vulnerabilities in networks, systems, and applications
- ❑ This assignment will reinforce your understanding of pentesting tools, vulnerabilities and exploits

# Vulnerability Scanning versus Penetration Testing

55

- Both are essential components of a comprehensive cybersecurity strategy, but they serve different purposes and involve different methodologies
  - ▣ The primary goal of **vulnerability scanning** is to identify known vulnerabilities in systems, applications, and networks; it provides an automated way to check for security weaknesses
  - ▣ The primary goal of **penetration testing** is to simulate real-world attacks to assess the security posture of a system, application, or network, thereby determining the impact and the effectiveness of existing security measures
    - Pentesters use a combination of automated tools and manual techniques (e.g. social engineering) to find and exploit vulnerabilities by mimicking the actions of real attackers

# Vulnerability Disclosure Options

56

- ❑ Tell no one (No disclosure)
- ❑ Report in full to public immediately (Full disclosure)
- ❑ Report to vendor only and potentially receive bug bounty!

 <b>Amazon Vulnerability Research Program</b> <a href="https://www.amazon.com">https://www.amazon.com</a> Reports resolved: 746 Assets in scope: 35 Average bounty: -	SEVERITY	Amount (in USD)
	Critical	\$10,000 - \$20,000
	High	\$1,500 - \$5,000
	Medium	\$350 - \$500
	Low	\$150

- ❑ Report to vendor, wait for fix, report to public (Responsible disclosure)
- ❑ Sell vulnerability to middleman and don't report to vendor
- ❑ Develop fully weaponized malware and distribute on black market

# CT437 COMPUTER SECURITY AND FORENSIC COMPUTING

THE CIA TRIAD  
REVISION: GDPR AND RAID

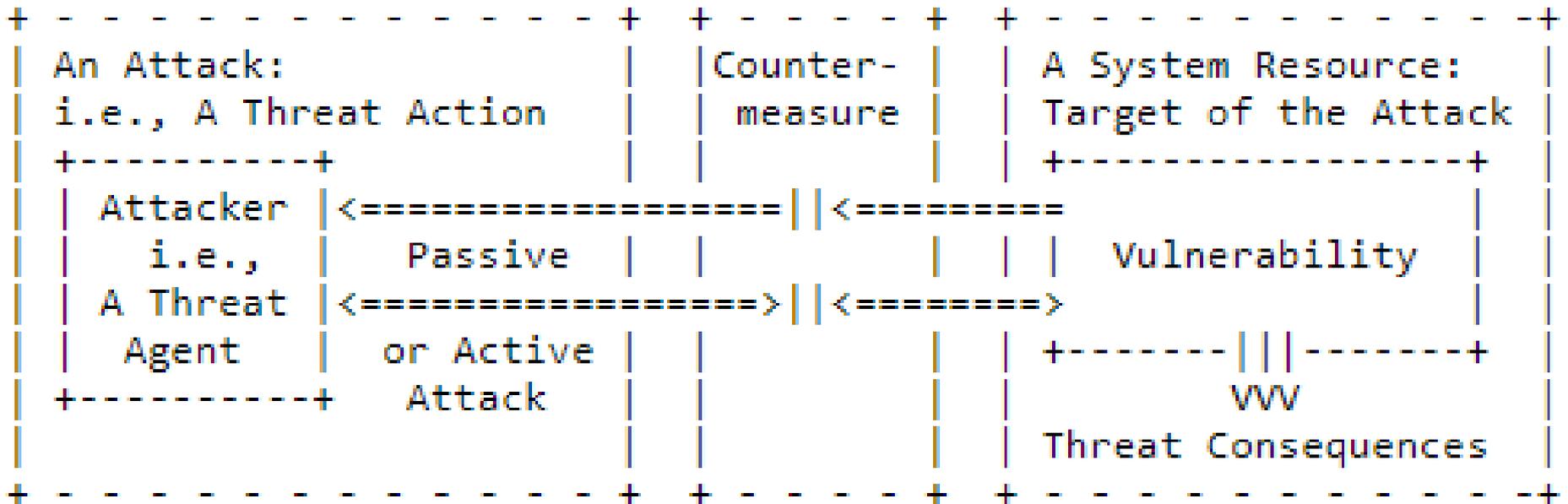
Dr. Michael Schukat



# Recap: RFC2828

2

- RFC2828, Internet Security Glossary
- <https://tools.ietf.org/html/rfc2828>



# Recap: Threat Consequences

3

- ❑ Threat consequences (as a result from a threat action) include
  - ❑ disclosure of information
  - ❑ Deception (i.e. pretending to be another entity)
  - ❑ disruption of services
  - ❑ usurpation, e.g. unauthorized control of some part of a system

- ❑ Cybercrime: it's all around us
  - Posing a major threat to personal and organizational data and even national security



Personal level

Your identity, data, and computing devices



Organizational level

Reputation, data and customers



Government level

National security, economy and the safety of citizens

# The CIA Triad

4

- The three letters in "CIA triad" stand for
  - Confidentiality
  - Integrity, and
  - Availability
- It is a model that forms the basis for the development of security systems



# CIA Triad: Confidentiality

5

- ❑ *“Preserving authorized restrictions on information access and disclosure, including means for protecting personal privacy and proprietary information”*
- ❑ In layman's terms: Keeping things secret that are meant to be secret
- ❑ Protecting data at rest, in transit, and during processing / use



# Recall GDPR: Personal Data

- Any information relating to an identified or identifiable natural person ('data subject')
- An identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person"

# Recall GDPR: Sensitive Personal Data

- This includes
  - ▣ Racial origin
  - ▣ Political opinions
  - ▣ Religious or philosophical beliefs
  - ▣ Trade Union membership
  - ▣ Genetic data (e.g. biological samples)
  - ▣ Biometric data (e.g. fingerprints)
  - ▣ Data concerning health
  - ▣ Data concerning a person's sex life or sexual orientation
  
- GDPR requires explicit consent to process special categories of personal data

# Compromising Confidentiality

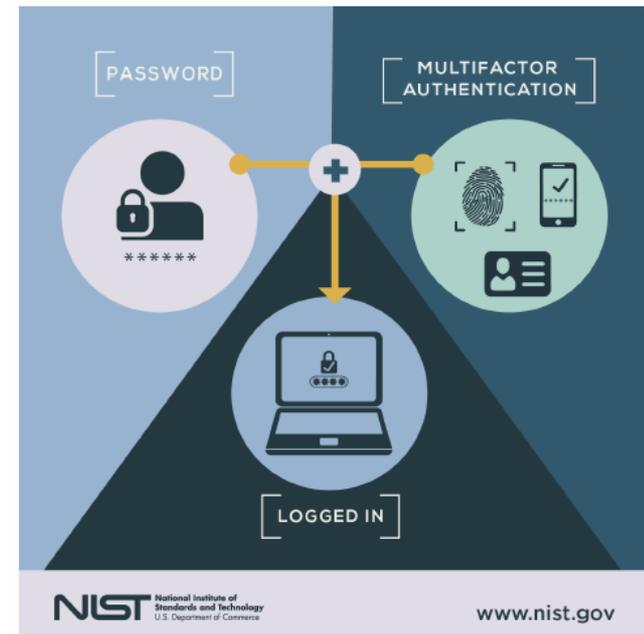
8

- Confidentiality can be compromised by various attacks, including:
  - Man-in-the-middle (MitM) attack
  - Escalation of privileges
  - Human error (weak password, sharing credentials etc.)
  - Insufficient security controls
  
- Can you identify situations where any of those attacks would apply?

# Technologies used to ensure Confidentiality

9

- These include:
  - ▣ Encryption (obviously)
  - ▣ Access Control (e.g. multi-factor authentication)
  - ▣ Secure network protocols
  
- Can you name a technology / protocol / algorithm that ensures confidentiality?



# CIA Triad: Integrity

10

- “Guarding against improper information modification or destruction, and includes ensuring information non-repudiation and authenticity”
  - ▣ Non-repudiation ensures that a party cannot deny having sent or received a message or transaction
  - ▣ Authenticity ensures that information and communication come from a trusted source; this includes protecting against impersonation, spoofing and other types of identity fraud
- In layman’s terms: keeping information accurate, complete, and protected from unauthorised modification
- Integrity makes sure that data is trustworthy and not tampered with
- Think of Revolut; can you provide an example for an attack on data integrity?

# Technologies to protect Integrity

- These include:
  - ▣ Network protocols that validate all data exchanged between end points
  - ▣ Digital signatures
  - ▣ Data hashes
  - ▣ Backup and data recovery strategies
  - ▣ Version control, to prevent the accidental change or deletion of information

# CIA Triad: Availability

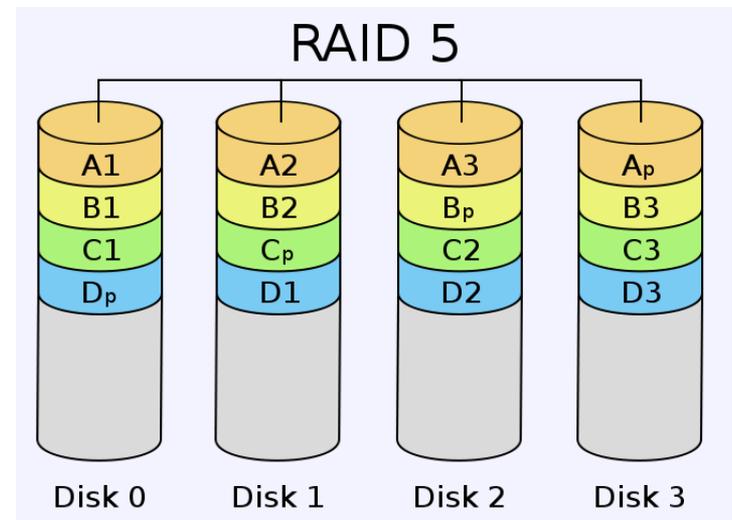
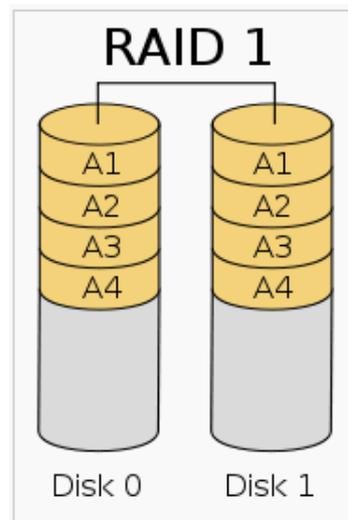
12

- “Ensuring timely and reliable access to and use of information”
- If data is kept confidential and its integrity maintained but it is not available to use, then it is often useless
- Availability can be compromised by various attacks, including:
  - ▣ (Distributed) Denial-of-service (DoS) attacks
  - ▣ Ransomware
  - ▣ Server overload
  - ▣ Physical incident such as a power outage or natural disaster

# Technologies to provide Availability

13

- These include:
  - ▣ RAID – Redundant Array of Independent Disks
  - ▣ Load balancers
  - ▣ Business continuity and disaster recovery plans, e.g., redundancy, failover, etc.

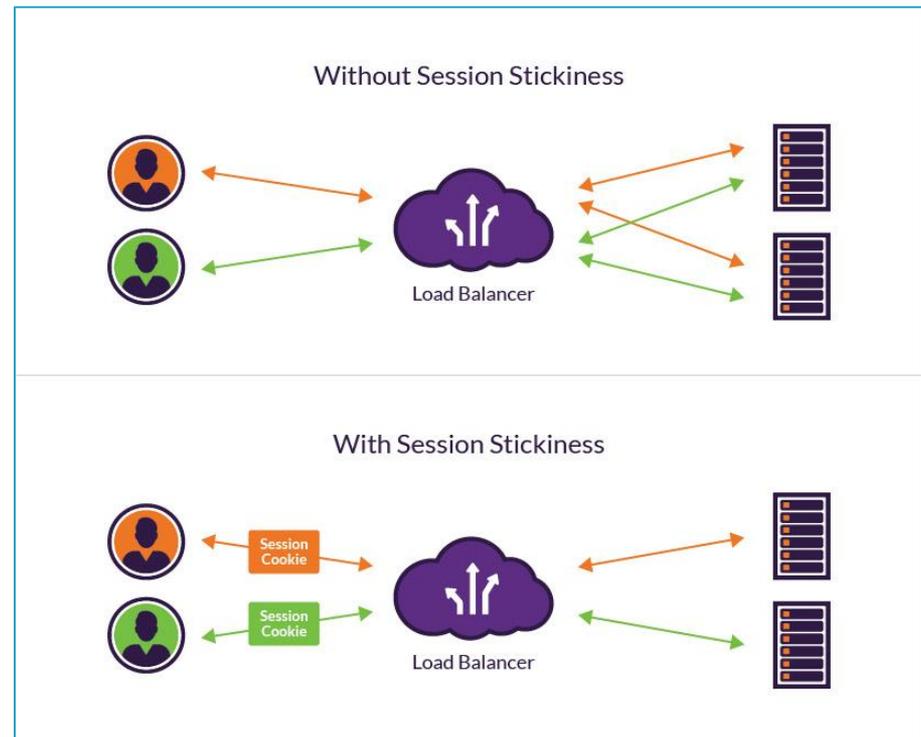


# Load Balancers

- Load balancers are server-side gateways that distribute client traffic between multiple backend (e.g., web-) servers
- They require load-balancing cookies on the client side that associate a client session with a particular server, aka **session stickiness**
- A load balancer **creates an affinity** between a client and a specific network server for the duration of a session using a cookie with a random and unique tracking id
- Subsequently, for the duration of the session, the load balancer routes all of the requests of this client to a specific backend server using the tracking id
- GDPR allows the unsolicited use of such cookies via the **communications exemption**

# Load Balancers

- ◆ Top image:
  - No load balancing at all
- ◆ Bottom image:
  - The LB generates and returns a tracking cookie back to a client when its first session is initiated
  - This cookie is tagged to every subsequent client request and allows the LB to forward the request to always the same server (therefore the stickiness)



# Data Breaches

16

- Despite the best of intentions and all the safeguards one can put in place, protecting organisations from every possible cyber attack may not be feasible
- There is an on-going "arms-race" with cybercriminals constantly finding new ways to attack systems and, very often, they will succeed
- *"A data breach is defined as any breach of security leading to the accidental or unlawful destruction, loss, alteration or unauthorised disclosure of or access to personal data transmitted, stored or otherwise processed"* (article 4.1 2 GDPR).

# CIA Triad Dependencies



17

- Each element connects with the others, and when you implement measures to ensure the protection of one, you must consider the ramifications it has elsewhere
- Example:
  - ▣ As a result of the recent cyberattack UoG implemented multi-factor authentication to access all services (email, student records, etc.)
  - ▣ Doing so protects the confidentiality of sensitive data, making it harder for unauthorised actors to compromise an employee's login credentials and view information using their account
  - ▣ However, without their mobile phone at hand, an employee can't complete the authentication process
  - ▣ This hampers therefore the availability of UoG services

# Risk Assessment

- ISO 27001 certification, GDPR compliance and other frameworks require the adoption of the CIA triad within an organisation
- All these standards mandate that organisations analyse their operations to measures the risks, threats and vulnerabilities in their systems that could compromise sensitive information
- This process is called **risk assessment**
- We may cover risk assessment at a later stage...

# Appendix / Revision

- RAID
- GDPR

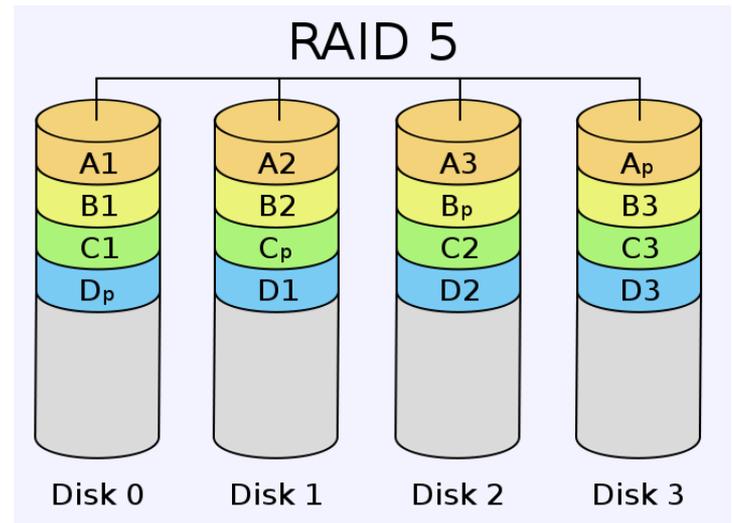
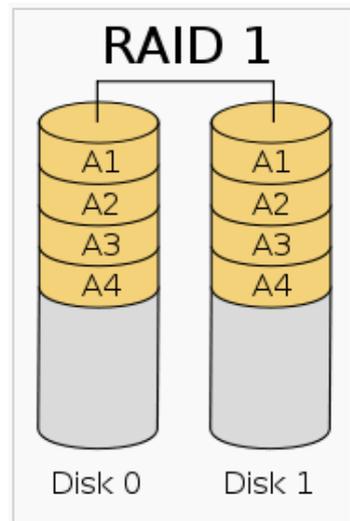
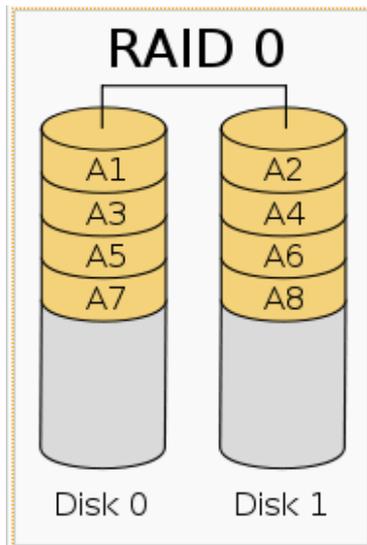
# Mass-Storage Redundancy via RAID

- Background: Hard disks are relatively slow (i.e. seek time + rotational latency) and as mechanical devices may fail
- Redundant Array of Independent Disks (RAID) is a data storage virtualisation technology that combines multiple physical disk drive components into one or more logical units for the purposes of data redundancy and / or performance improvement
- Data blocks are distributed across the drives in one of several ways, referred to as **RAID levels**, depending on the required level of redundancy and performance
- Many RAID levels use a parity-based error protection scheme (see RAID-4), example (with 12 bit / block):
  - Block 1: 010001101001
  - Block 2: 110011011010
  - Block 3: 000100100101
  - P Block: 100111010110 (bitwise EXOR, equivalent to even parity)



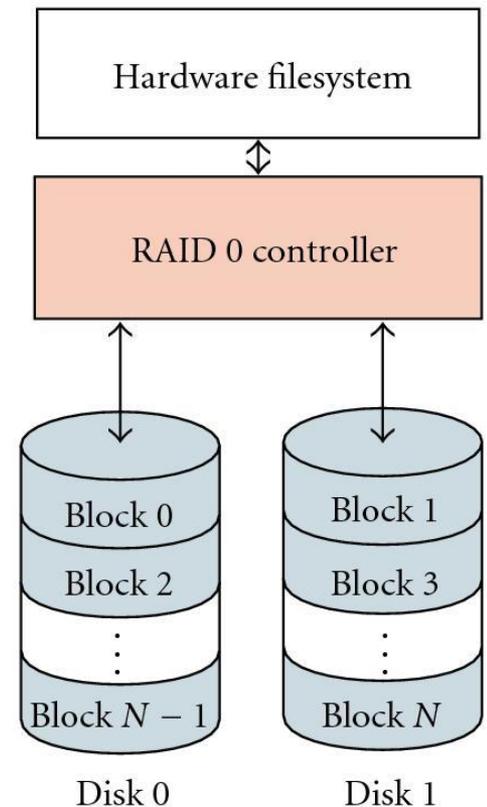
# Mass-Storage Redundancy via RAID

- ❑ RAID storage systems require a dedicated RAID controller, that supports the required RAID level
  - ▣ See also the diagram on the next slide
  - ▣ Normally such controllers are not shown in RAID diagrams



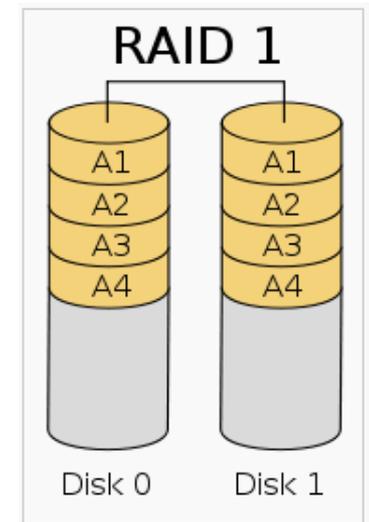
# RAID 0

- **Block-level striping without parity or mirroring**
  - ▣ **data striping** is the technique of segmenting logically sequential data, such as a file, so that consecutive segments are stored on different physical storage devices
- 2 or more drives (n) required
- **No redundancy**, but up to n-times R/W performance increase



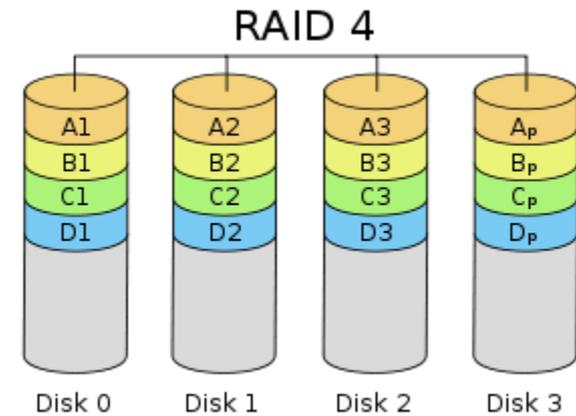
# RAID 1

- Block-level mirroring without parity or striping
- 2 or more drives (n) required
- (n - 1) drive failures can be compensated; here each disk can
  - ▣ diagnose catastrophic failures (e.g. head crash)
  - ▣ detect (but **not** correct) sector-wise bit errors on platters
- No increase in R/W performance



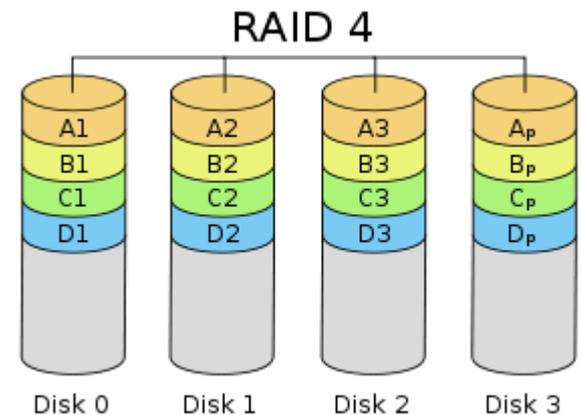
# RAID 4

- ❑ Block-level striping with single parity disk
- ❑ Single catastrophic drive failure can be compensated (any drive can fail)
- ❑ RAID 4 provides good performance of random reads, while the performance of random writes is low due to the need to write all parity data to a single disk (Disk 3 in the diagram above)
- ❑ Minimum of 3 drives required



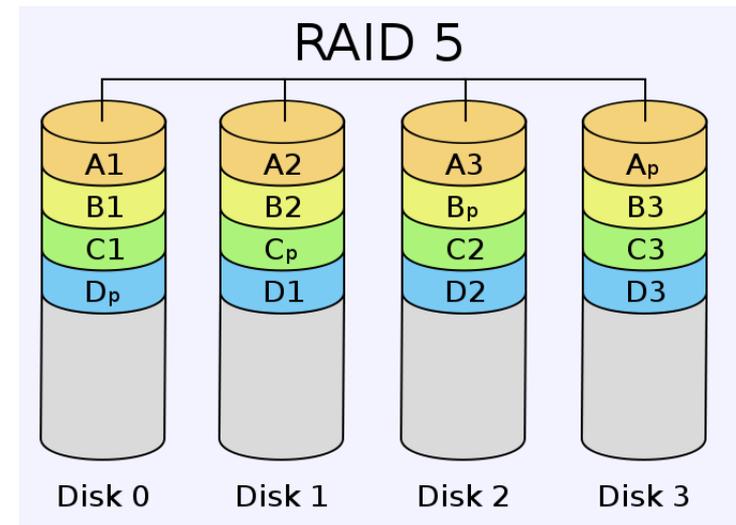
# Drive Hot-Swapping in RAID

- ❑ In RAID a defect drive will be (ASAP)
  - ❑ manually swapped for a new drive (hot-swap), or
  - ❑ replaced by an idle drive (hot-spare) already in the system
- ❑ The new drive's content is rebuild by the RAID controller while the disk set is still operational
- ❑ Example RAID 4 with Disk 0 swapped:
  - ❑  $A_1 = A_2 \text{ EXOR } A_3 \text{ EXOR } A_p$
  - ❑  $B_1 = B_2 \text{ EXOR } B_3 \text{ EXOR } B_p$
  - ❑  $C_1 = C_2 \text{ EXOR } C_3 \text{ EXOR } C_p$
  - ❑  $D_1 = D_2 \text{ EXOR } D_3 \text{ EXOR } D_p$



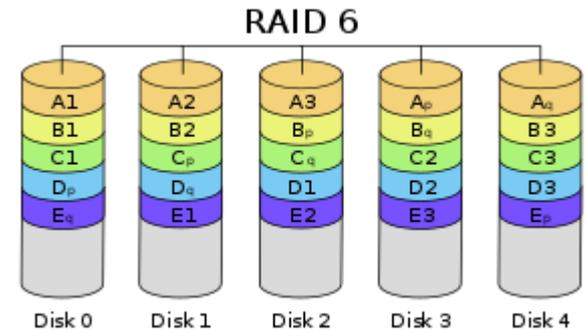
# RAID 5

- Similar to RAID 4, but:
  - ▣ Block-level striping with distributed parity
  - ▣ Distributed parity evens out the stress of a dedicated parity disk among all RAID members
  - ▣ Write performance is increased since all RAID members participate in the serving of write requests
- Minimum of 3 drives required



# RAID 6

- RAID 6 extends RAID 5 by adding another parity block
  - ▣ thus, it uses block-level striping with two parity blocks distributed across all member disks
- Double catastrophic drive failures can be compensated (any two drives can fail)
- Minimum of 4 disks required
- Second parity involves EXOR function (as seen before) and a bit shift function



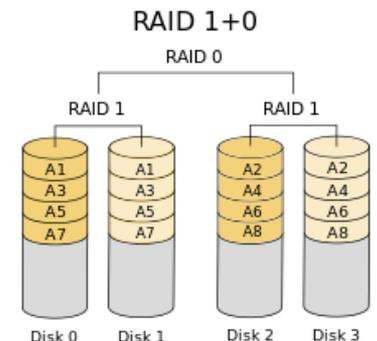
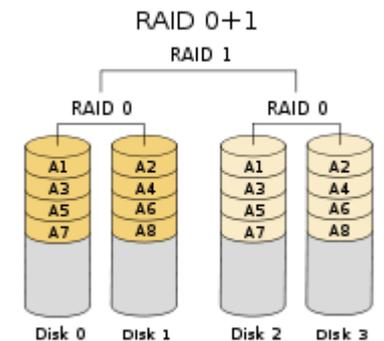
# Other RAID Levels

## ❑ RAID 0+1 (RAID 01)

- ❑ 4 drives minimum
- ❑ Some double catastrophic drive failures can be compensated

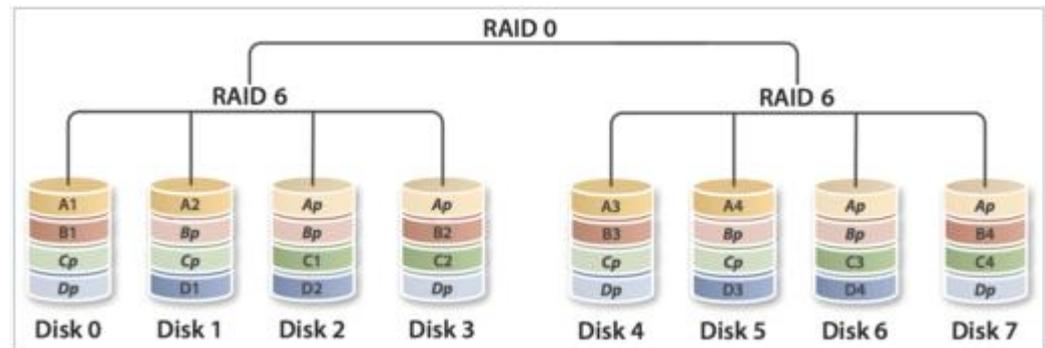
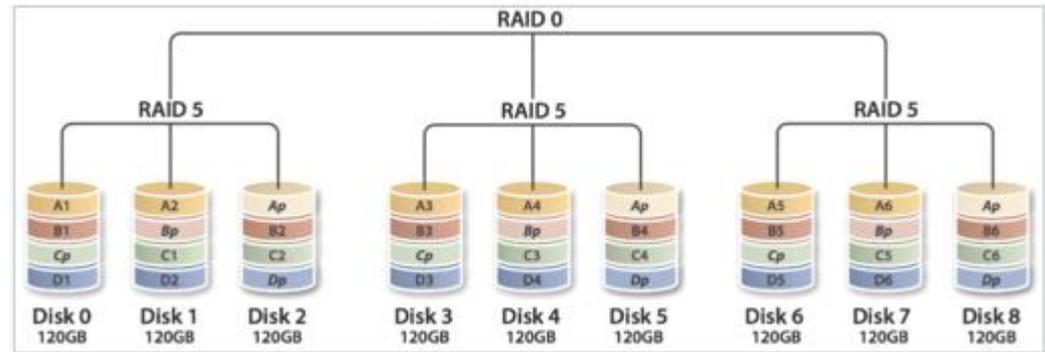
## ❑ RAID 1+0 (RAID 10)

- ❑ 4 drives minimum
- ❑ Some double catastrophic drive failures can be compensated
- ❑ Best throughput (apart from RAID 0), so preferable RAID level for I/O-intensive applications



# Other RAID Levels: RAID 5+0 (RAID 50) and RAID 6+0 (RAID 60)

- Some triple catastrophic drive failures can be compensated
- Rebuild-time after drive swap reduced because of controller hierarchy
  - ▣ Faster turnaround time to restore full I/O capacity
  - ▣ Shorter vulnerable period where a second drive failure would be catastrophic



# Summary RAID 0 – RAID 3

Level	Description	Minimum # of disks	Space Efficiency	Fault Tolerance	Read Benefit	Write Benefit	Image
RAID 0	Block-level striping without parity or mirroring.	2	1	0 (none)	nX	nX	<p>RAID 0</p> <p>Disk 0    Disk 1</p>
RAID 1	Mirroring without parity or striping.	2	1/n	n-1 disks	nX	1X	<p>RAID 1</p> <p>Disk 0    Disk 1</p>
RAID 2	Bit-level striping with dedicated Hamming-code parity.	3	$1 - 1/n \cdot \log_2(n-1)$	RAID 2 can recover from 1 disk failure or repair corrupt data or parity when a corrupted bit's corresponding data and parity are good.			<p>RAID 2</p> <p>Disk 0    Disk 1    Disk 2    Disk 3    Disk 4    Disk 5</p>
RAID 3	Byte-level striping with dedicated parity.	3	$1 - 1/n$	1 disk			<p>RAID 3</p> <p>Disk 0    Disk 1    Disk 2    Disk 3</p>

○

○

# Summary RAID 4 – RAID 6

Level	Description	Minimum # of disks	Space Efficiency	Fault Tolerance	Read Benefit	Write Benefit	Image
RAID 4	Block-level striping with dedicated parity.	3	$1 - 1/n$	1 disk			<p>RAID 4 diagram showing four disks (Disk 0 to Disk 3). Data blocks A1, B1, C1, D1 are striped across Disk 0, 1, 2, 3. Parity blocks A2, B2, C2, D2 are stored on Disk 3. Data blocks A3, B3, C3, D3 are striped across Disk 0, 1, 2, 3. Parity blocks A4, B4, C4, D4 are stored on Disk 3.</p>
RAID 5	Block-level striping with distributed parity.	3	$1 - 1/n$	1 disk	$(n-1)X$	variable	<p>RAID 5 diagram showing four disks (Disk 0 to Disk 3). Data blocks A1, B1, C1, D1 are striped across Disk 0, 1, 2, 3. Parity block A2 is on Disk 0, B2 on Disk 1, C2 on Disk 2, D2 on Disk 3. Data blocks A3, B3, C3, D3 are striped across Disk 0, 1, 2, 3. Parity block A4 is on Disk 1, B4 on Disk 2, C4 on Disk 3, D4 on Disk 0.</p>
RAID 6	Block-level striping with double distributed parity.	4	$1 - 2/n$	2 disks			<p>RAID 6 diagram showing five disks (Disk 0 to Disk 4). Data blocks A1, B1, C1, D1, E1 are striped across Disk 0, 1, 2, 3, 4. Parity blocks A2, B2, C2, D2, E2 are stored on Disk 1, 2, 3, 4, 0. Data blocks A3, B3, C3, D3, E3 are striped across Disk 0, 1, 2, 3, 4. Parity blocks A4, B4, C4, D4, E4 are stored on Disk 2, 3, 4, 0, 1.</p>



# General Data Protection Regulation

- GDPR is a binding regulation in EU law on data protection in the EU and the European Economic Area (EEA), that became enforceable on 25 May 2018
- It also addresses the transfer of personal data outside the EU and EEA areas
- The GDPR's primary aim is to **enhance individuals' control and rights over their personal data and to simplify the regulatory environment for international business**
- The regulation **contains provisions and requirements related to the processing of personal data of individuals** who are located in the EEA, and applies to any enterprise—**regardless of its location and the data subjects' citizenship or residence**—that is processing the personal information of individuals inside the EEA

# GDPR in Ireland

- GDPR applies to the majority of personal data processing tasks, but further rules on certain issues (for example the reasons for, and extent to which, data subject rights may be restricted) are set out in the **Data Protection Act 2018**
- Further on, the **Law Enforcement Directive** concerns the processing of personal data by law enforcement, i.e., the prevention, investigation, detection or prosecution of criminal offences or the execution of criminal penalties

# What is Data Protection?

- Data protection is about an **individual's fundamental right for privacy**
- When an individual gives their personal data to any organisation, the recipient has the duty to keep the data safe and private
- Data protection legislation
  - ▣ governs the way we deal with personal data / information
  - ▣ provides a mechanism for safeguarding privacy rights of individuals in relation to the processing of their data
  - ▣ upholds rights and enforces obligations

# Recall GDPR: Personal Data

- Any information relating to an identified or identifiable natural person ('data subject')
- An identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person"

# Recall GDPR: Sensitive Personal Data

- This includes
  - ▣ Racial origin
  - ▣ Political opinions
  - ▣ Religious or philosophical beliefs
  - ▣ Trade Union membership
  - ▣ Genetic data (e.g. biological samples)
  - ▣ Biometric data (e.g. fingerprints)
  - ▣ Data concerning health
  - ▣ Data concerning a person's sex life or sexual orientation
- GDPR requires explicit consent to process special categories of personal data

# Recap Cookies

- An (HTTP) cookie is a small piece of data stored on the user's computer by the web browser while browsing a website
- Cookies were designed to be a reliable mechanism for websites to remember stateful information (such as items added in the shopping cart in an online store) or to record the user's browsing activity
- They can also be used to remember pieces of information that the user previously entered into form fields
- Authentication cookies are the most common method used by web servers to know whether the user is logged in or not, and which account they are logged in with

# Cookie Implementation

- Cookies are arbitrary pieces of data (i.e. large random strings), usually chosen and first sent by the web server, and stored on the client computer by the web browser
- The browser then sends them back to the server with every request
- Browsers are required to:
  - ▣ support cookies as large as 4,096 bytes in size
  - ▣ support at least 50 cookies per domain (i.e. per website)
  - ▣ support at least 3,000 cookies in total

# Setting a Cookie - Example

- A browser sends its first request for the homepage of [www.example.org](http://www.example.org), resulting in the GET request

```
GET /index.html HTTP/1.1
Host: www.example.org
...
```

- The server responds with

```
HTTP/1.0 200 OK
Content-type: text/html
Set-Cookie: theme=light
Set-Cookie: sessionToken=abc123; Expires=Wed, 09 Jun 2021 10:18:14 GMT
...
```

- Later client requests to this server will contain these cookies:

```
GET /spec.html HTTP/1.1
Host: www.example.org
Cookie: theme=light; sessionToken=abc123
...
```

# Cookie Structure

- A cookie consists of the following components:
  - ▣ Name
  - ▣ Value
  - ▣ Zero or more attributes (name/value pairs)  
Attributes store information such as the cookie's expiration, domain, and flags (such as *Secure* and *HttpOnly*)

```
HTTP/1.0 200 OK
Content-type: text/html
Set-Cookie: theme=light
Set-Cookie: sessionToken=abc123; Expires=Wed, 09 Jun 2021 10:18:14 GMT
...
```

# Session Cookies

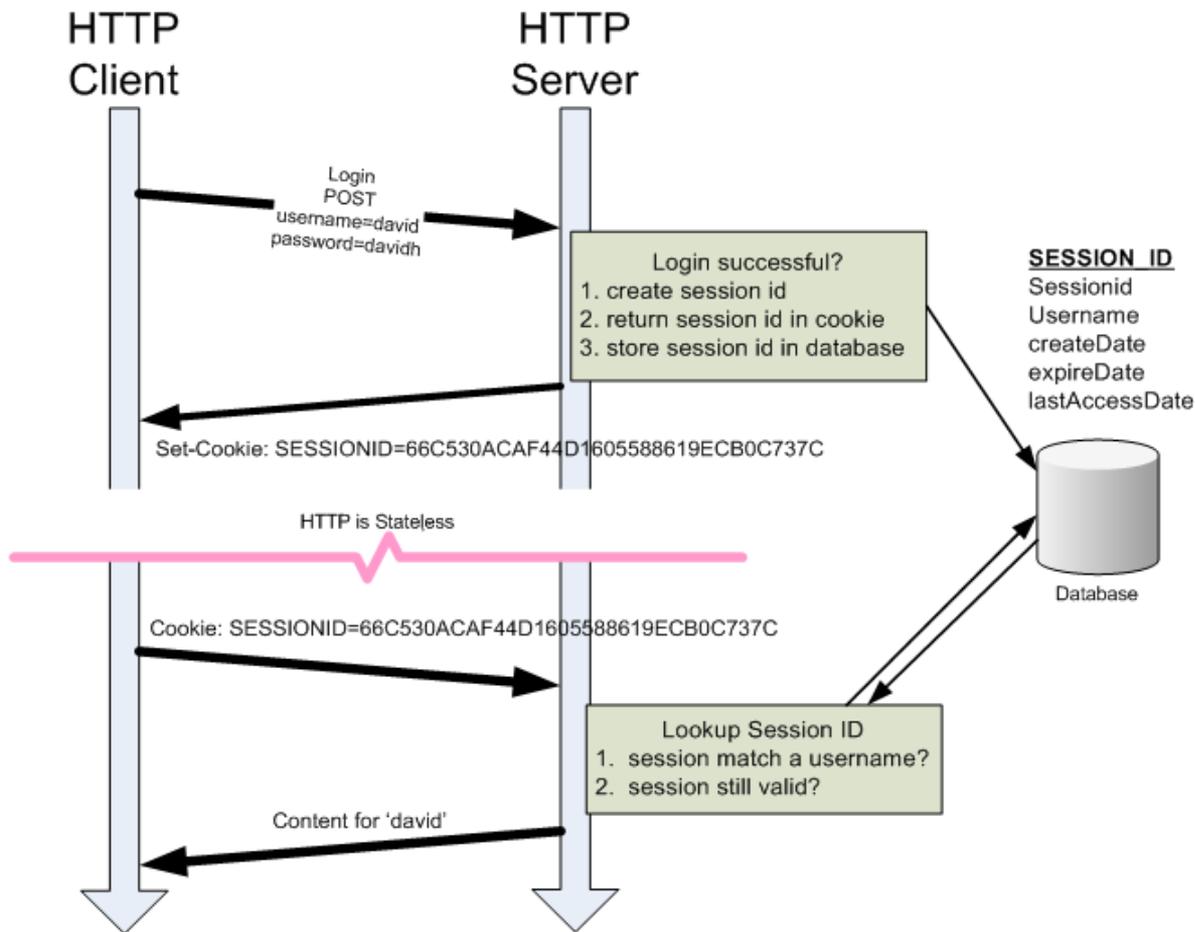
- A session cookie (aka in-memory cookie, transient cookie or non-persistent cookie) exists only in temporary memory while the user navigates its website
- Web browsers normally delete session cookies when the user closes the browser
- Session cookies do not have an expiration date assigned to them, which is how the browser knows to treat them as session cookies
- Example: “theme” cookie on previous slide

# Persistent Cookie

- A persistent cookie expires at a specific date or after a specific length of time
- For the persistent cookie's lifespan set by its creator, its information will be transmitted to the server every time the user visits the website that it belongs to
- ... or every time the user views a resource belonging to that website from another website (such as an advertisement).  
For this reason, persistent cookies are sometimes referred to as tracking cookies because they can be used by advertisers to record information about a user's web browsing habits
- However, they are mainly used for legitimate reasons, such as keeping users logged into their accounts on websites, to avoid re-entering login credentials at every visit
- Example: “sessionToken” cookie in the previous example

# Session Management via Persistent Cookies

44



# Cookie Attributes

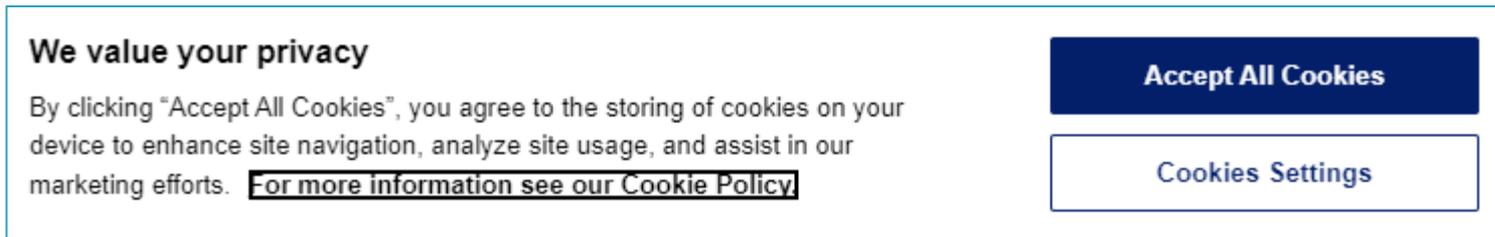
- Consider the following response header sent by a webserver that contains 3 persistent cookies:

```
HTTP/1.0 200 OK
Set-Cookie: LSID=DQAAAK...Eaem_vYg; Path=/accounts; Expires=Wed, 13 Jan 2021 22:23:01 GMT; Secure; HttpOnly
Set-Cookie: HSID=AYQEVn...DKrdst; Domain=.foo.com; Path=/; Expires=Wed, 13 Jan 2021 22:23:01 GMT; HttpOnly
Set-Cookie: SSID=Ap4P...GTEq; Domain=foo.com; Path=/; Expires=Wed, 13 Jan 2021 22:23:01 GMT; Secure; HttpOnly
...
```

- The *Domain* and *Path* attributes define the cookie's scope
- The *Secure* attribute makes sure that the cookie can only be transmitted over an encrypted connection (i.e. HTTPS → later), making it a **secure cookie**
- The *HttpOnly* attribute directs browsers not to expose cookies through channels other than HTTP / HTTPS requests  
This means that this **HttpOnly cookie** cannot be accessed via client-side scripting languages (notably JavaScript)

# GDPR and Cookies

- Generally, a user's consent must be sought before a cookie is installed in a web browser

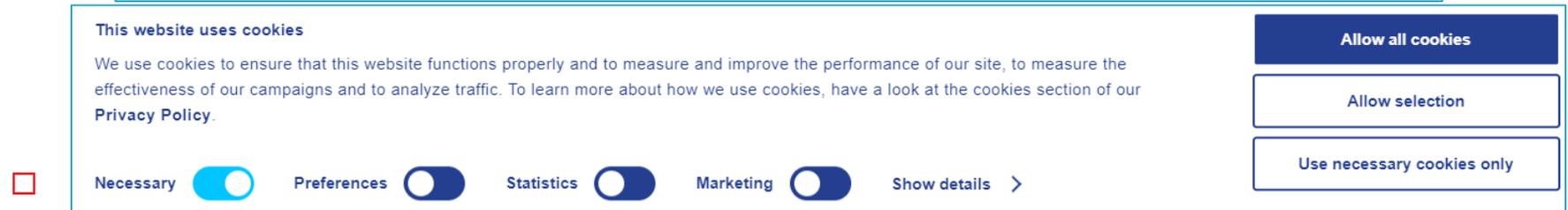


**We value your privacy**

By clicking "Accept All Cookies", you agree to the storing of cookies on your device to enhance site navigation, analyze site usage, and assist in our marketing efforts. [For more information see our Cookie Policy](#)

**Accept All Cookies**

Cookies Settings



- **This website uses cookies**

We use cookies to ensure that this website functions properly and to measure and improve the performance of our site, to measure the effectiveness of our campaigns and to analyze traffic. To learn more about how we use cookies, have a look at the cookies section of our [Privacy Policy](#).

Necessary  Preferences  Statistics  Marketing  Show details >

**Allow all cookies**

Allow selection

Use necessary cookies only

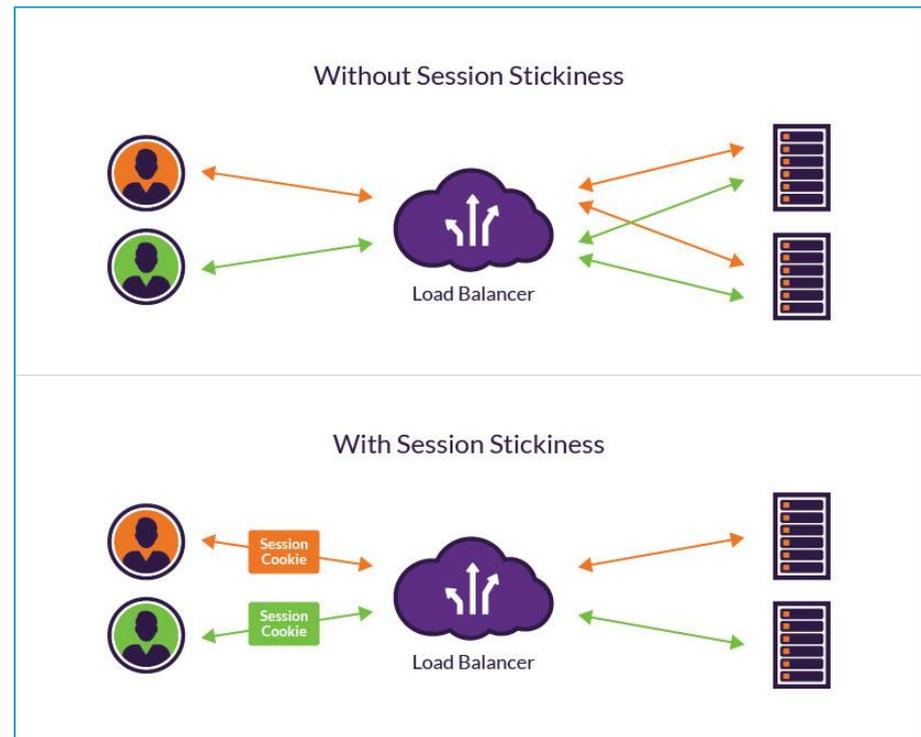
- The communications exemption
- The strictly necessary exemption

# The Communications Exemption

- This applies to cookies whose sole purpose is for carrying out the transmission of a communication over a network, for example to identify the communication endpoints
- Example: load-balancing cookies that distribute network traffic across different backend servers, aka **session stickiness**
  - Here a load balancer **creates an affinity** between a client and a specific network server for the duration of a session using a cookie with a random and unique tracking id
  - Subsequently, for the duration of the session, the load balancer routes all of the requests of this client to a specific backend server using the tracking id

# Session Stickiness

- ◆ Top image:
  - No load balancing at all
- ◆ Bottom image:
  - The LB generates and returns a tracking cookie back to a client when its session is initiated
  - This cookie is tagged to every subsequent client request and allows the LB to forward the request to always the same server (therefore the stickiness)



# The *strictly necessary Exemption*



- Must be linked to a service delivered over the internet, i.e. a website or an app
- This service must have been explicitly requested by the user (i.e. typing in the URL) and the use of the cookie must be restricted to what is strictly necessary to provide that service
- Note that cookies related to advertising are not strictly necessary and must be consented to

# Example for the *strictly necessary* *Exemption*

- A website uses session cookies to keep track of items a user places in an online shopping basket
  - ▣ Assuming this cookie will be deleted once the session is over
- Cookies that record a user's language or country preference

# Data Processing



- Performing any operation on personal data, manually or by automate means, including:
  - ▣ Obtaining
  - ▣ Storing
  - ▣ Transmitting
  - ▣ Recording
  - ▣ Organising
  - ▣ Altering
  - ▣ Disclosing
  - ▣ Erasing

# Entities in GDPR



- GDPR distinguishes between:
  - ▣ The Data Subject
  - ▣ The Data Protection Officer (DPO)
  - ▣ The Data Controller
  - ▣ The Data Processor

# The Data Subject

- This is the (living!) person to whom the data relates
- Under GDPR, businesses have a legal obligation to keep their data up-to-date, which means that, theoretically, data of deceased should be removed
- Deceased persons, who predate the introduction of GDPR, may be covered by national legislation active at the time of death (e.g. the Data Protection Acts 1988 and 2003 in Ireland)
- Access by a next-of-kin to the personal data (e.g. emails) of a deceased person may be possible under Irish **Freedom of Information laws**

# The Data Protection Officer (DPO)

- ❑ The primary role of the DPO is to ensure that her organisation processes the personal data of its staff, customers, and other data subjects in compliance with the applicable data protection rules
- ❑ It is a mandatory role within three different scenarios:
  - ❑ When the processing is undertaken by a public authority or body
  - ❑ When an organisation's main activities require the frequent and large-scale monitoring of individual people
  - ❑ Where large scale processing of special categories of data or data relating to criminal records forms the core activities
- ❑ The Data Protection Officer is required to be an expert within this field, along with the requirement for them to report to the highest management level.
  - ❑ With this being a challenging aspect of GDPR compliance for smaller organisations, there is the option to make an external appointment of a third-party

# The Data Controller



- The Data Controller is the company or an individual who has overall control over the processing of personal data
- The Data Controller takes on the responsibility for GDPR compliance
- A Data Controller needs to have had sufficient training and be able to competently ensure the security and protection of data held within the organisation

# The Data Processor

- The Data Processor is the person who is responsible for the processing of personal information
- Generally, this role is undertaken under the instruction of the data controller
  - ▣ This might mean obtaining or recording the data, its adaption and use. It may also include the disclosure of the data or making it available for others
- Generally, the Data Processor is involved in the more technical elements of the operation, while the interpretation and main decision making is the role of the Data Controllers

# Cloud Services and GDPR



- A Cloud Service Provider will be considered a **Data Processor** under GDPR if it provides data processing services (e.g. storage) on behalf of the Data Controller even without determining the purposes and means of processing
- A Cloud Service Provider that offers personal data processing services directly to Data Subjects will be **Data Controller**

# Some Key Benefits for Data Subjects

- More information must be given to data subjects (e.g. how long data will be kept, right to lodge a complaint)
- Must explain and document legal basis for processing personal data
- Tightens the rules on how consent is obtained (must be distinguishable from other matters and in clear plain language)
- Must be as easy to withdraw consent as it is to give it
- Mandatory notification of security breaches without undue delay
  - To data protection commissioner within 72 hours

# Personal Data Security Breaches

- ❑ Disclosure of confidential data to unauthorised individuals
- ❑ Loss or theft of data or equipment on which data is stored
- ❑ Hacking, viruses or other security attacks on IT equipment/ systems / networks
- ❑ Inappropriate access controls allowing unauthorised use of information
- ❑ Emails containing personal data sent in error to wrong recipient
- ❑ Applies to paper and electronic records

# Some Key Benefits for Data Subjects



- Right of Access (copy to be provided within one month)
- Right to erasure (i.e. right to be forgotten)
- Right to restriction of processing
- Right to object to processing
- Right not to be subject to a decision based solely on automated processing

# GDPR Key Principles

- The GDPR sets out several key principles:
  1. Lawfulness
  2. Fairness and transparency
  3. Purpose limitation
  4. Data minimisation
  5. Accuracy
  6. Storage limitation
  7. Integrity and confidentiality (security)
  8. Accountability

# GDPR Principle: Lawfulness

- You must **identify valid grounds** under the GDPR (known as a ‘lawful basis’) for collecting and using personal data
- Processing shall be lawful only if and to the extent that at least one of the following applies:
  - ▣ Consent
  - ▣ Necessary for the performance of a contract
  - ▣ Necessary for compliance with a legal obligation
  - ▣ Necessary to protect the vital interests of the data subject or another person
  - ▣ Necessary for the performance of a task carried out in the public interest
  - ▣ Necessary for the purpose of the legitimate interests

# GDPR Principle: Fairness and Transparency

- ❑ You must **use personal data in a way that is fair**; this means you must not process the data in a way that is unduly detrimental, unexpected or misleading to the individuals concerned
- ❑ You must be **clear, open and honest with people** from the start about how you will use their personal data
- ❑ At the time personal data is being collected from data subjects, they must be informed via a "Data Protection Notice"

# Data Protection Notice

- A data protection notice entails the following:
  - ▣ The identity and contact details of the data controller
  - ▣ The contact details of the data protection officer
  - ▣ The purpose of the processing and the legal basis for the processing
  - ▣ The recipients or categories of recipients of the data
  - ▣ Details of any transfers out of the EEA, safeguards in place and the means by which to obtain a copy of them
  - ▣ The data retention period used or criteria to determine same
  - ▣ The individual's rights (access, rectification and erasure, restriction, complaint)

# GDPR Principle: Purpose Limitation



- ❑ You must be **clear about what your purposes** for processing are from the start
- ❑ You need to **record your purposes** as part of your documentation obligations and specify them in your privacy information for individuals
- ❑ You **can only use the personal data for a new purpose** if either this is compatible with your original purpose, you get consent, or you have a clear basis in law

# GDPR Principle: Data Minimisation

- You must ensure the personal data you are processing is:
  - **adequate** – sufficient to properly fulfil your stated purpose
  - **relevant** – has a rational link to that purpose
  - **limited** to what is necessary – you do not hold more than you need for that purpose

# GDPR Principle: Accuracy

- ❑ You should take all reasonable steps to ensure the personal data you hold **is not incorrect or misleading** as to any matter of fact
- ❑ You may need to **keep the personal data updated**, although this will depend on what you are using it for
- ❑ If you **discover that personal data is incorrect or misleading**, you must take reasonable steps to correct or erase it as soon as possible
- ❑ You must **carefully consider any challenges to the accuracy** of personal data

# GDPR Principle: Storage Limitation

- ❑ You must not keep personal data **for longer than you need it**
- ❑ You need to think about – and be able to justify – **how long you keep personal data**; this will depend on your purposes for holding the data
- ❑ You need a policy **setting standard retention periods** wherever possible, to comply with documentation requirements
- ❑ You should also **periodically review the data you hold**, and erase or anonymise it when you no longer need it
- ❑ You must **carefully consider any challenges to your retention of data**; individuals have a right to erasure if you no longer need the data
- ❑ You can **keep personal data for longer if you are only** keeping it for public interest archiving, scientific or historical research, or statistical purposes

# GDPR Principle: Accountability and Governance

- Accountability is one of the data protection principles - it makes you responsible for complying with the GDPR and says that **you must be able to demonstrate your compliance**
- You need to put in place appropriate technical and organisational measures to meet the requirements of accountability

# GDPR Principle: Accountability and Governance

- Accountability requires controllers to maintain records of processing activities in order to demonstrate how they comply with the data protection principles, i.e.
  - ▣ Inventory of personal data
  - ▣ Providing assurance about compliance
  - ▣ Need to document
    - Why it is held
    - How it is collected
    - When it will be deleted
    - Who may gain access to it

# GDPR Principle: Integrity and Confidentiality

- A key principle of the GDPR is that you process personal data securely by means of ‘appropriate technical and organisational measures’ – this is the ‘security principle’
- Doing this requires you to consider things like risk analysis, organisational policies, and physical and technical measures
- Where appropriate, you should look to use measures such as **pseudonymisation and encryption**
- Your measures must ensure the ‘**confidentiality, integrity and availability**’ of your systems and services and the personal data you process within them
- The measures must also enable you to **restore access and availability** to personal data in a timely manner in the event of a physical or technical incident

# CT437 COMPUTER SECURITY AND FORENSIC COMPUTING

HISTORY OF CRYPTOGRAPHY  
CRYPTOGRAPHIC CONCEPTS

Dr. Michael Schukat



# Lecture Outline and Motivation

2

- Recap: Technologies used to ensure confidentiality:
  - ▣ **Encryption (obviously)**
  - ▣ Access Control (e.g. multi-factor authentication)
  - ▣ Secure network protocols
- Therefore, this lecture provides:
  - ▣ A summary of terms linked to cryptography
  - ▣ An overview of historic cryptographic algorithms (recap CT255)
  - ▣ Some context for the next topic, **modern cryptography**



# Basic Terminology



## □ Cryptography

- The art of encompassing the principles and methods of transforming an intelligible message into one that is unintelligible, and then retransforming that message back to its original form
  - Intelligible means “able to be understood” or comprehensible

# Basic Terminology

- Plaintext
  - ▣ The original intelligible message, e.g. “THIS IS A SECRET MESSAGE”
- Ciphertext
  - ▣ The transformed message, e.g. “XPHDSYUEGSD68G4AS8AG56”
- Cipher
  - ▣ An algorithm for transforming an intelligible message into one that is unintelligible
- Key
  - ▣ Some critical information used by the cipher, known only to the sender & receiver
  - ▣ Selected from a **keyspace**  $K$  (i.e., a set of all possible keys)

# Basic Terminology

- Encipher (encode)
  - ▣ The process of converting plaintext to ciphertext using a cipher and a key
- Encryption
  - ▣ The mathematical function  $E_K()$  mapping plaintext  $P$  to ciphertext using the specified key  $K$ :

$$C = E_K(P)$$

# Basic Terminology

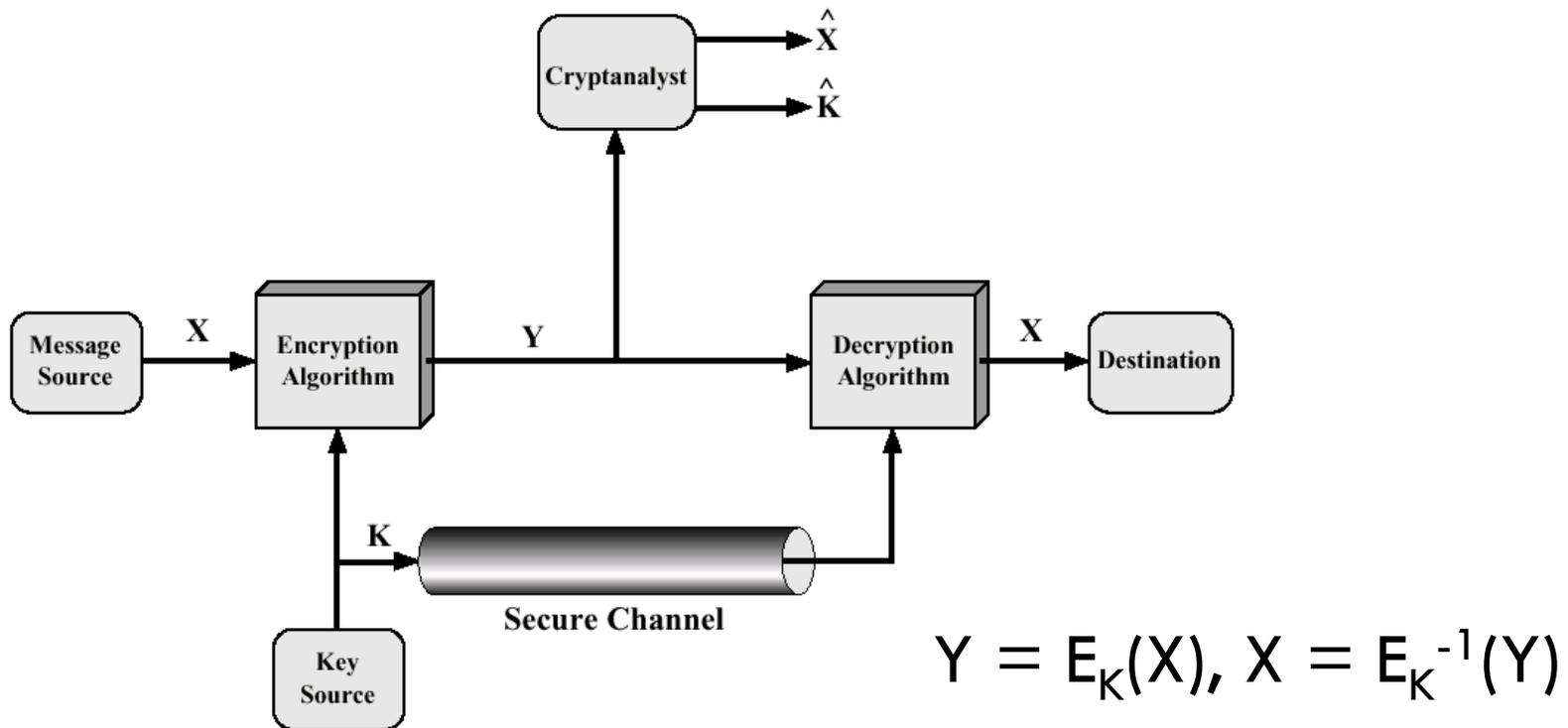
- Decipher (decode)
  - ▣ The process of converting ciphertext back into plaintext using a cipher and a key
- Decryption:
  - ▣ The mathematical function  $E_K^{-1}()$  mapping ciphertext  $C$  to plaintext  $P$  using the specified key  $K$ :

$$P = E_K^{-1}(C)$$

# Basic Terminology

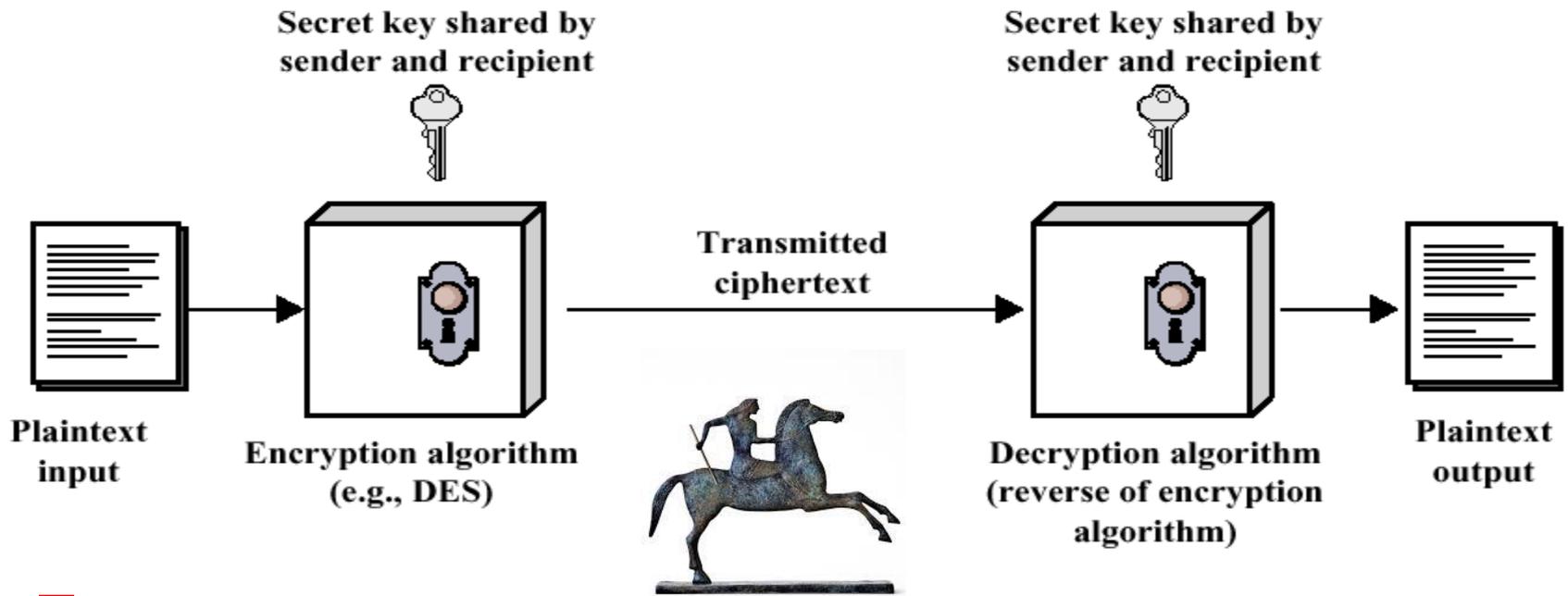
- Cryptanalysis
  - ▣ The study of principles and methods of transforming an unintelligible message back into an intelligible message without knowledge of the key
- Cryptology
  - ▣ The field encompassing both cryptography and cryptanalysis

# Model of Conventional Cryptosystem



# Classical Cryptography

- ❑ Ancient ciphers have been in use for over 5,000 years
- ❑ Already used by ancient Egyptians, Hebrews and Greeks
- ❑ Normally they would follow the following scheme:



# Caesar Cipher

- 2000 years ago, Julius Caesar used a simple substitution cipher, now known as the Caesar cipher
- First attested use in military affairs (Gallic Wars)
- Replace each letter by 3rd letter on, e.g.  
L FDPH L VDZ L FRQTXHUHG ->  
I CAME I SAW I CONQUERED
- We can describe this mapping (or translation alphabet) as:  
Plain: ABCDEFGHIJKLMNOPQRSTUVWXYZ  
Cipher: DEFGHIJKLMNOPQRSTUVWXYZABC

# Generalised Caesar Cipher

- More generally can use any shift from 1 to 25, i.e. replace each letter of message by a letter a fixed distance away
- Specify key letter as the letter a plaintext A maps to,
  - e.g. a key letter of F means  
A maps to F, B to G, ... Y to D, Z to E  
e.g. shift letters by 5 places
- Hence have 26 (25 useful) ciphers
- Note that with this and all other historic ciphers punctuation and spaces are ignored and all text is converted to capital letters

# How to break the generalised Caesar Cipher

- Try all 25 possibilities until you recover some meaningful text

KEY	PHHW	PH	DIWHU	WKH	WRJD	SDUWB
1	oggv	og	chvgt	vjg	vqic	rcvta
2	nffu	nf	bgufs	uif	uphb	qbsuz
3	meet	me	after	the	toga	party
4	ldds	ld	zesdq	sgd	snfz	ozqsx
5	kcoo	kc	ydrop	rfo	rmey	nyprw
6	jbbq	jb	xoqbo	qeb	qldx	mxoqv
7	iaap	ia	wbpan	pda	pkcw	lwnpu
8	hzzo	hz	vaozm	ocz	ojbv	kvmot
9	gyyn	gy	uznyl	nby	niau	julns
10	fxxm	fx	tymxk	max	mhzt	itkmr
11	ewwl	ew	sxlwj	lzw	lgys	hsjlg
12	dvvk	dv	rwkvi	kyv	kfxr	grikp
13	cuuj	cu	qvjuh	jxu	jewq	fqhjo
14	btti	bt	puitg	iwt	idvp	epgin
15	assh	as	othsf	hvs	hcuo	dofhm
16	zrrg	zr	nsgre	gur	gbtn	cnegl
17	yqqf	yq	mrfqd	ftq	fasm	bmdfk
18	xppe	xp	lqepc	esp	ezrl	alcej
19	wood	wo	kpdob	dro	dyqk	zkbdi
20	vnnc	vn	jocna	oqn	cxpj	yjach
21	ummb	um	inbmz	bpm	bwoi	xizbg
22	tlla	tl	hmaly	aol	avnh	whyaf
23	skkz	sk	glzcx	znk	zumg	vgxze
24	rjyy	rj	fkyjw	ymj	ytlf	ufwyd
25	qiix	qi	ejxiv	xli	xske	tevxc

# The Mono-Alphabetic (or simple) Substitution Cipher

- In the mono-alphabetic (or simple substitution) cipher each letter of the plain text is replaced with another letter of the alphabet
- It uses a fixed key which consist of the 26 letters of a “shuffled alphabet”.
- Example:
  - Plaintext alphabet (obviously): ABCDEFGHIJKLMNOPQRSTUVWXYZ
  - Ciphertext alphabet (i.e. the key): ZEBRASCDFGHIJKLMNOPQTUVWXY
  - Plaintext message:  
FLEEATONCEWEAREDISCOVERED
  - Ciphertext message:  
SIAAZQLKBAVAZORFPBLUAOAR
- This ciphers allows for  $26!$  ( $= 4.0329146e+26$ ) possible key combinations ...
  - This is too many combinations for a brute force attack where the attacker tries every single possible key!
    - This of course assumes that the attacker can identify the correctly decoded cyphertext (e.g., a text written in English)
- **Is this cipher therefore unbreakable?**

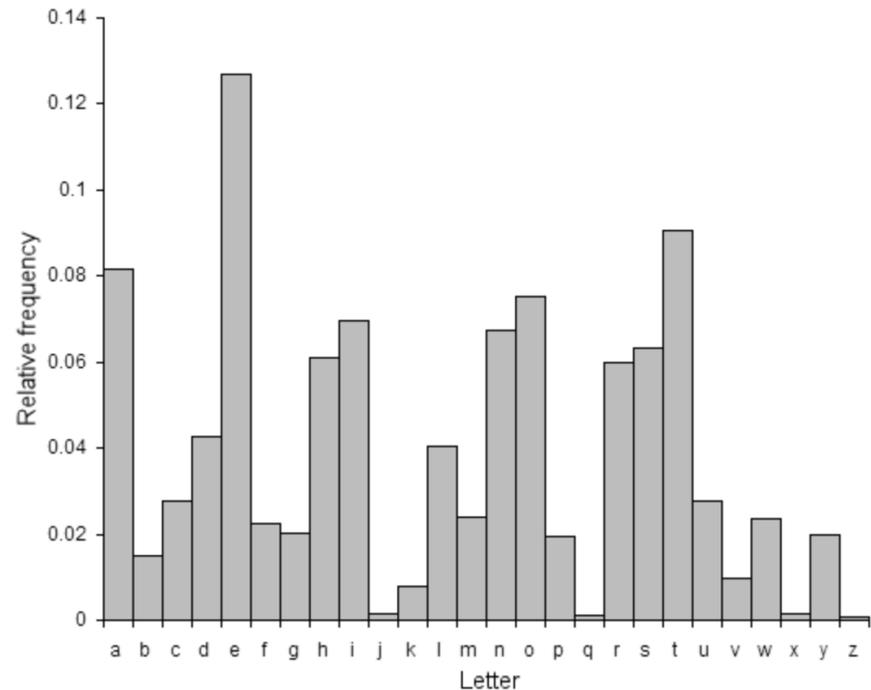
# Non-trivial Cryptanalysis Attacks Against Substitution Ciphers

14

- **Frequency Analysis:**  
This attack relies on the fact that certain letters or symbols occur more frequently in the English language than others. By analysing the frequency of characters in the ciphertext, one can make educated guesses about the substitutions made in the cipher, ultimately revealing the plaintext  
See also next slides
- **Pattern Recognition:**  
Cryptanalysts may also exploit patterns in the ciphertext to deduce information about the key. Recognisable patterns, such as common word endings or repeating character sequences, can provide valuable clues about the substitutions used in the cipher
- **Known-Plaintext Attack:**  
See next slides

# Cryptanalysis via Letter Frequency Analysis

- In most written languages, letters are not equally commonly used
- For example, in the English language:
  - ▣ E is by far the most common letter followed by T,R,N,I,O,A,S
  - ▣ Other letters like Z,J,K,Q,X are fairly rare
  - ▣ See frequency table on the right
- There are tables of single, double & triple letter frequencies for all common languages
- There is an example for the cryptanalysis of a ciphertext via letter frequency analysis at the end of this slide deck



# C-Program for Letter Frequency Analysis

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

int main(int argc, char *argv[])
{
    FILE *fp;
    int data[26];
    char c;
    int i;

    memset(data, 0, sizeof(data));

    if (argc != 2)
        return(-1);
```

```
    if ((fp = fopen(argv[1], "r")) == NULL)
        return(-2);

    while (!feof(fp))
    {
        c = toupper(fgetc(fp));

        if ((c >= 'A') && (c <= 'Z'))
            data[c - 65]++;
    }

    for (i = 0; i < 26; i++)
        printf("%c: %i\n", i + 65, data[i]);

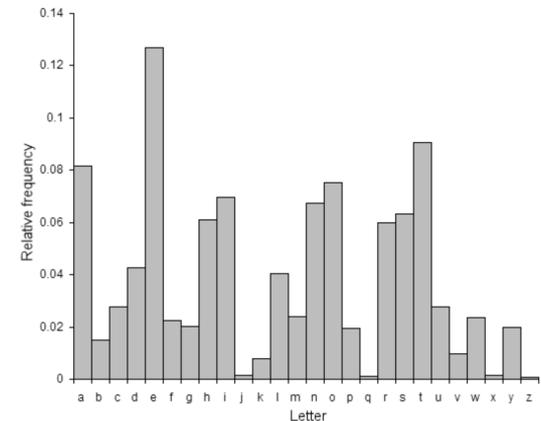
    fclose(fp);
    return(1);
}
```

# Known Plaintext Attacks

- The known-plaintext attack (KPA) is an attack model for cryptanalysis where the attacker has access to both
  - ▣ (some of) the plaintext (this is called a crib),
  - ▣ and its encrypted version
- See the example on the next slide

# Example: Combined known-Plaintext and Pattern Recognition Attack

- You are presented with the following ciphertext which is based on a substitution cipher:  
**JEPOUMJWFIFSFCVUNZIPNFJTNZDBTUMFGVMMTUPQ**
- You know the original plaintext message consists of capital letters only (no spaces) and contains the following plaintext **crib**:  
**MYHOMEISMYCASTLE**
- How could you tackle this?



# Playfair Cipher

- Not even the large number of keys in a monoalphabetic substitution cipher provides security!
- One approach to improving security was to encrypt multiple letters at once
- The **Playfair Cipher** is an example for such an approach
- Algorithm was invented by Charles Wheatstone in 1854, but named after his friend Baron Playfair

# Playfair Cipher

## □ How it works:

- Create a 5x5 grid of letters; insert the keyword as shown, with each letter only considered once; fill the grid with the remaining letters in alphabetic order
- Letters I and J are treated the same (see example above with keyword IRELAND)
- Letters are encrypted in pairs
- Repeats have an X inserted:  
BALLOON -> BA LX LO ON
- Letters that fall in the same row are each replaced with the letter on the right (OK becomes GM)
- Letters in the same column are replaced with the letter below (FO becomes OU)
- Otherwise, each letter gets replaced by the letter in its row but in the other letters column (QM becomes TH)

I/J	R	E	L	A
N	D	B	C	F
G	H	K	M	O
P	Q	S	T	U
V	W	X	Y	Z

# Robustness of Playfair Cipher

- The algorithm's complexity was much improved over the simple monoalphabetic cipher, since we have  $26 \times 26$  ( $= 676$ ) character combinations we have to deal with
- This requires a 676-entry frequency table for analysis (versus 26 for a monoalphabetic cipher) and substantially more ciphertext for a cryptanalysis
- Therefore, it was widely used for many years, e.g., by US & British military in WW1
- However, it **can** be broken, given a few hundred letters

# Example Playfair Cipher

- ❑ Consider the Playfair Cipher and the key “PRUNEJUICE”
- ❑ Encipher the following plaintext: “KENSENTMEX”
- ❑ What is the resulting ciphertext?

# Vigenère Cipher

- Blaise de Vigenère is generally credited as the inventor of the "polyalphabetic substitution cipher"
  - ▣ A monoalphabetic cipher is any cipher in which the letters of the plain text are mapped to cipher text letters based on a single alphabetic key
  - ▣ A polyalphabetic substitution ciphers uses multiple substitution alphabets
- To improve security, use many monoalphabetic substitution alphabets
- Hence each letter can be replaced by many others
- Use a key to select which alphabet is used for each letter of the message
- $i^{\text{th}}$  letter of key specifies  $i^{\text{th}}$  alphabet to use
- Use each alphabet in turn
- Repeat from start after end of key is reached



# Example Vigenère Cipher

- Encode the plaintext “**KENSENTME**” using the Vigenère cipher and the keyword “BABA”

- Plaintext:            KENSENTME
- Key:                    BABABABAB
- Ciphertext:           MFPTGOVNG

# How to break the Vigenère Cipher

- Search the ciphertext for repeated strings of letters; the longer strings you find the better
- For each occurrence of a repeated string, count how many letters are between the first letters in the string and add one
- Factor the number you got in the above computation (e.g., 2, 5 and 10 itself are factors of 10)
- Repeat this process with each repeated string you find and make a table of common factors. The most common factor is probably the length of the keyword that was used to encipher the ciphertext. Call this number 'n'
- Do a frequency count on the ciphertext, on every nth letter. You should end up with n different frequency counts
- Compare these counts to standard frequency tables to figure out how much each letter was shifted by
- Undo the shifts and read off the message!

# Example Breaking the Vigenère Cipher

Key: ABCDABCDABCDABCDABCDABCDABCD (not known to attacker)

Plaintext: **CRYPTO**ISSHORTFOR**CRYPTO**GRAPHY (not known to attacker)

Ciphertext: CSASTPKVSIQUTGQUCSASTPIUAQJB

- Our search reveals the following repetition:  
**CSASTP** KV SIQUT GQU **CSASTP**IUAQJB
- The distance is 16, therefore the key length  $n$  is either 2, 4, 8 or 16 characters
- Do four different frequency counts on the ciphertext, i.e., on every  $n^{\text{th}}$  letter
- Continue as shown before

# In-Class Activity: Breaking Vigenère

- Consider the following ciphertext that has been encoded using a Vigenère Cipher:

DYDUXRMHTVDVNQDQNWLDYDUXRMHARTJGWNQD

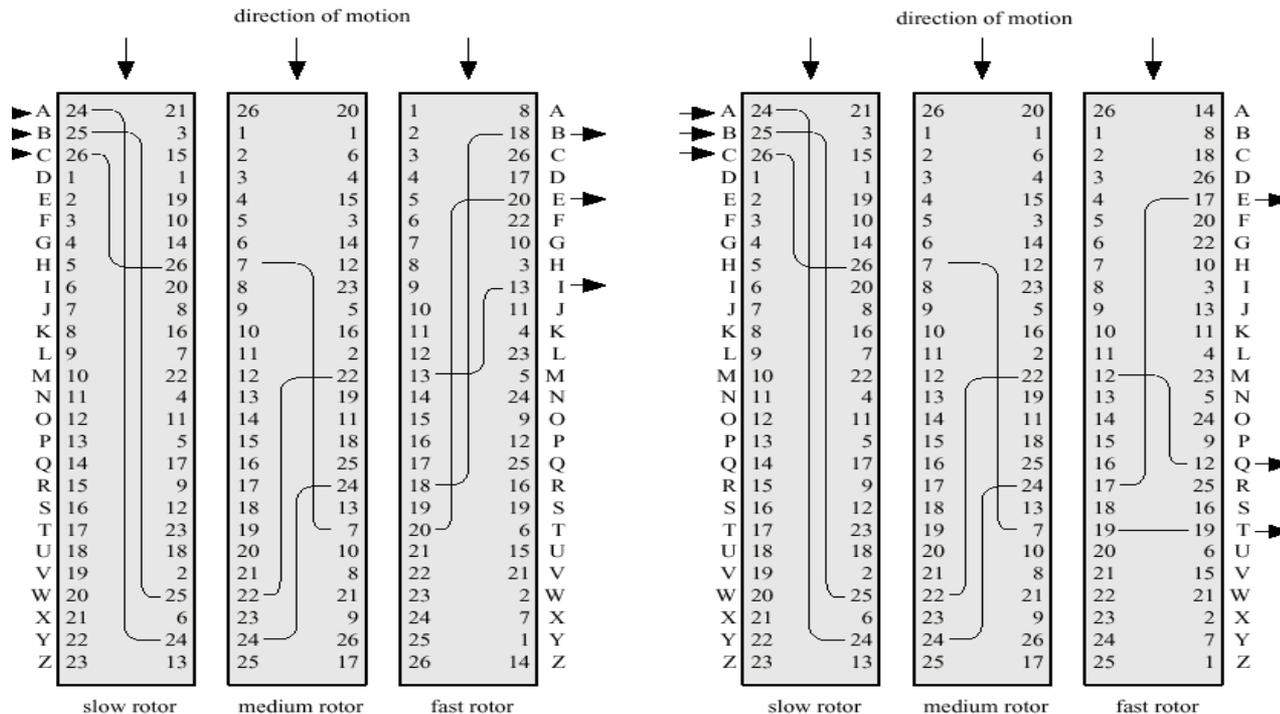
- Q1: Which repeating strings can you identify?
- Q2: What is the distance of their appearances?
- Q3: Subsequently, what is the probable key length?

# Rotor Machines

- These allowed for the mechanisation / automation of message encryption and decryption and were widely used in the 20<sup>th</sup> century (until the 1970s)
- The primary components of a rotor machine are
  - ▣ a set of rotors
  - ▣ a keyboard for inputting text
  - ▣ A dashboard (e.g. array of letter-coded lamps) to show the output
- Rotors are rotating disks with an array of electrical contacts on either side
- The wiring between the contacts implements a fixed substitution of letters, replacing them in some complex fashion
- After encrypting of a letter, the rotors advance positions, changing the substitution (to be applied to the next letter that is typed in)
- By this means, a rotor machine produces a complex polyalphabetic substitution cipher, which changes with every key press

# Rotor Machines

- The example below shows schematically  $N = 3$  rotors including some of their internal wiring
- Keyboard and dashboard are not shown
- The medium rotor advances its position after a full turn of the fast rotor, and the slow rotor advances its position after a full turn of the medium rotor
- Therefore, we have a  $N$ -stage polyalphabetic substitution algorithm
- For  $N = 5$ , there are  $26^N (= 11881376)$  steps before a substitution is repeated!



(a) Initial setting

(b) Setting after one keystroke

# Example: The Enigma Machine



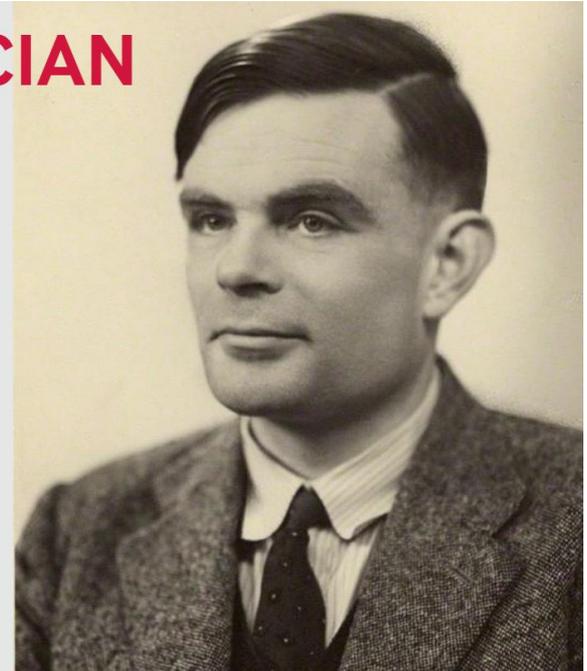
[https://www.youtube.com/watch?v=-mdSvGUd0\\_c](https://www.youtube.com/watch?v=-mdSvGUd0_c)

# How Alan Turing broke the Enigma Code

- <https://www.iwm.org.uk/history/how-alan-turing-cracked-the-enigma-code>
- The Imitation Game (Film, 2014)
- <https://www.youtube.com/watch?v=nuPZUUED5uk>

## MATHEMATICIAN

Alan Turing was a brilliant mathematician. Born in London in 1912, he studied at both Cambridge and Princeton universities. He was already working part-time for the British Government's Code and Cypher School before the Second World War broke out. In 1939, Turing took up a full-time role at Bletchley Park in Buckinghamshire – where top secret work was carried out to decipher the military codes used by Germany and its allies.



# Breaking Enigma using Cribs

36

- Breaking Enigma was based on the following observations:
  - ▣ Plaintext messages were likely to contain certain phrases, e.g.
    - Weather reports contained the term "WETTER VORHERSAGE"
    - Military units often sent messages containing "KEINE BESONDEREN EREIGNISSE", i.e. "nothing to report"
  - ▣ A plaintext letter was never mapped onto the same ciphertext letter

# Breaking Enigma using Cribs (Wikipedia)

37

- While the cryptanalysts did not know where exactly these cribs were placed in an intercepted message, they could exclude certain positions (i.e. Position 1 and 3):

Ciphertext	O	H	J	Y	P	D	O	M	Q	N	J	C	O	S	G	A	W	H	L	E	I	H	Y	S	O	P	J	S	M	N	U
Position 1			K	E	I	N	E	B	E	S	O	N	D	E	R	E	N	E	R	E	I	G	N	I	S	S	E				
Position 2			K	E	I	N	E	B	E	S	O	N	D	E	R	E	N	E	R	E	I	G	N	I	S	S	E				
Position 3				K	E	I	N	E	B	E	S	O	N	D	E	R	E	N	E	R	E	I	G	N	I	S	S	E			
<p>Positions 1 and 3 for the possible plaintext are impossible because of matching letters.</p> <p>The red cells represent these <i>crashes</i>. Position 2 is a possibility.</p>																															

- From here on, possible rotor start positions and rotor wiring would be systematically examined using a “the bombe”, an electromechanical device designed by Alan Turing

# Transposition Ciphers

- Now consider classical **transposition** or **permutation** ciphers
- These hide the message by rearranging the letter order without altering the actual letters used
- This can be recognised since ciphertext has the same frequency distribution as the original text

# Rail Fence (Zigzag) Cipher

- Write message letters out diagonally up and down over a number of rows, then read off cipher row by row
- Example (Wikipedia): WE ARE DISCOVERED. RUN AT ONCE:

```
W . . . E . . . C . . . R . . . U . . . O . . .  
. E . R . D . S . O . E . E . R . N . T . N . E  
. . A . . . I . . . V . . . D . . . A . . . C .
```

- **Resulting ciphertext:**

WECRUOERDSOEERNNTNEAIVDAC

# Row Transposition Ciphers

- Write letters of message out in rows over a specified number of columns
- Then reorder the columns according to some key before reading off the columns
- Example:

```
Key:          4 3 1 2 5 6 7
Plaintext:   A T T A C K P
              O S T P O N E
              D U N T I L T
              W O A M X Y Z
```

```
Ciphertext: TTNA APTM TSUO AODW COIX KNLY PETZ
```

**Note that spaces are inserted to improve readability**

# Combined Ciphers

- Ciphers using substitutions or transpositions are not very strong because of language characteristics
- Hence consider using several ciphers in succession to make harder:
  - ▣ two substitutions make a more complex substitution
  - ▣ two transpositions make more complex transposition
  - ▣ but a substitution followed by a transposition makes a new much harder cipher
- Similar approaches are implemented in modern ciphers

The header consists of a red bar on the left and a teal bar on the right, both extending horizontally across the top of the slide. The word "Steganography" is written in white on the teal bar.

# Steganography

# Steganography

- An alternative to encryption
- Steganography hides the existence of a message, by:
  - ▣ Using only a subset of letters / words in a longer message marked in some way
  - ▣ Using invisible ink
  - ▣ Hiding single bits at a time in suitable computer files (e.g., images or sound files)
- Drawback:
  - ▣ Not very economical in terms of overheads to hide a message (see also examples)

# Example for Steganography



- Assume an  $x$ -by- $y$  pixel wide image is stored in RGB format
- For each pixel the colour component (R, G and B) intensity is represented by a byte
- The image can be stored in a byte array of size  $[x][y][3]$
- For each entry we change the least significant bit to hide bitwise a message, e.g.


	R	G	B	becomes	R	G	B
	01010110	11100101	10110000		01010111	11100100	10110000
	11111111	10101001	00101010		11111111	10101000	00101011
	11001101	10011001	11001010		11001100	10011001	11001010
	...				...		
- This transformation allows the storage of the bit pattern 100101010, while causing minimal image distortions (invisible for the human eye)
- However, this method doesn't work in combination with image compression (e.g. JPEG compression)

# Annex

45

1. Example cryptanalysis of a simple substitution cipher

# Example Cryptanalysis of Simple Substitution Cipher

- Assume one intercepts the ciphertext below
- We know (out of the context) that
  - ▣ the plaintext message is written in English
  - ▣ The message has been encoded using the simple substitution cipher
- Intercepted ciphertext:  
UZQSOVUOHXMOPVGPOZPEVSGZWSZOPFPESXUDBMETSXA  
IZVUEPHZHMDZSHZOWSFPAPPDTSVPPQUZWYMXUZUHSXEPYE  
POPDZSZUFPOMBZWPFUPZHMDJUDTMOHMQ
- We do a frequency analysis of the ciphertext and begin with the most common letters in the English language, e and t
- Guess ciphertext letters P & Z are plaintext letters e and t (we use small letters to distinguish between both):  
UtQSOVUOHXMOeVGOteEVSGtWStOeFeESXUDBMETSXAtV  
UEeHtHMDtSHtOWSFeAeeDTSVeQUZWYMXUtUHSXEeYEeOeD  
tStUFeOMBtWeFUetHMDJUDTMOHMQ

# Example Cryptanalysis

- Guess (!) Z?P means *the*:

UtQSOVUOHXMOeVGeOteEVSGtWStOeFeESXUDBMET  
SXAltVUEeHtHMDtSHtOWSFeAeeDTSVeQUZWYMXUtUH  
SXEeYEeOeDtStUFeOMBtWeFUetHMDJUDTMOHMQ

- Assume W is *h*:

UtQSOVUOHXMOeVGeOteEVSGthStOeFeESXUDBMETS  
XAltVUEeHtHMDtSHtOhSFeAeeDTSVeQUZWYMXUtUHSX  
EeYEeOeDtStUFeOMBtheFUetHMDJUDTMOHMQ

# Example Cryptanalysis

- Guess word *that*, translating S into a:

UtQSOVUOHXMOeVGeOteEVSGthStOeFeESXUDBMET  
SXAltVUEeHtHMDtSHtOhSFeAeeDTSVeQUZWYMXUtUH  
SXEeYEeOeDtStUFeOMBtheFUetHMDJUDTMOHMQ

- Ciphertext becomes:

UtQaOVUOHXMOeVGeOteEVaGthatOeFeEaXUDBMET  
aXAltVUEeHtHMDtaHtOhsFeAeeDTaVeQUZWYMXUtUH  
aXEeYEeOeDtatUFeOMBtheFUetHMDJUDTMOHMQ

# Example Cryptanalysis

- Guess that AeeD means *been*:

UtQaOVUOHXMOeVGeOteEVaGthatOeFeEaXUDBM  
ETaXAltVUEeHtHMDtaHtOhsFeAeeDTaVeQUZWYMXU  
tUHaxEeYEeOeDtatUFeOMBtheFUetHMDJUDTMOHM  
Q

- Resulting in (with  $A \rightarrow b$  and  $D \rightarrow n$ ):

UtQaOVUOHXMOeVGeOteEVaGthatOeFeEaXUnBM  
ETaX**b**ltVUEeHtHM**n**taHtOhsFe**been**TaVeQUZWYMXUt  
UHaxEeYEeOe**n**tatUFeOMBtheFUetHM**n**JU**n**TMOHMQ

# Example Cryptanalysis

- Is HMntaHt meaning *contact*?

UtQaOVUOHXMOeVGeOteEVaGthatOeFeEaXUnBMET  
aXbltVUEeHtHMntaHtOhsFebeenTaVeQUZWYMXUtUH  
aXEeYEeOentatUFeOMBtheFUetHMnJUnTMOHMQ

- Therefore (with  $H \rightarrow c$  and  $M \rightarrow o$ ):

UtQaOVUOcXoOeVGeOteEVaGthatOeFeEaXUnBoETa  
XbltVUEectcontactOhaFebeenTaVeQUZWYoXUtUcaXEe  
YEeOentatUFeOoBtheFUetconJUnToOcoQ

# Example Cryptanalysis

- Does VUEect mean *direct*?

UtQaOVUOcXoOeVGeOteEVaGthatOeFeEaXUnBoETaX  
blt **VUEect**contactOhaFebeenTaVeQUZWYoXUtUcaXEeY  
EeOentatUFeOoBtheFUetconJUnToOcoQ

- Therefore (with  $V \rightarrow d$ ,  $U \rightarrow i$  and  $E \rightarrow r$ ):

**it**QaO**di**OcXoOe**d**GeOter**da**GthatOeFe**ra**X**i**nBorTaXblt  
**direct**contactOhaFebeenTadeQ**i**ZWYoX**it****ica**X**re**Y**re**Oent  
at**i**FeOoBthe**F****i**etcon**J****i**nToOcoQ

# Example Cryptanalysis

- Does GeOterdaG mean yesterday?

itQaOdiOcXoOedGeOterdaGthatOeFeraXinBorTaXblt  
directcontactOhaFebeenTadeQiZWYoXiticaXreYreOent  
atiFeOoBtheFietconJinToOcoQ

- Therefore (with  $G \rightarrow y$  and  $O \rightarrow s$ ):

itQasdiscxosedyesterdaythatseFeraXinBorTaXbltdirect  
contactshaFebeenTadeQiZWYoXiticaXreYresentatiFeso  
BtheFietconJinToscoQ

# Example Cryptanalysis

- Moscow calling?

itQasdiscXosedyesterdaythatseFeraXinBorTaXbltdirectco  
ntactshaFebeenTadeQiZWYoXiticaXreYresentatiFesoBth  
eFietconJin**ToscoQ**

- Therefore (with  $T \rightarrow m$  and  $Q \rightarrow w$ ):

itwasdiscXosedyesterdaythatseFeraXinBormaXbltdirectc  
ontactshaFebeenmadewiZWYoXiticaXreYresentatiFesoB  
theFietconJinmoscow

# Example Cryptanalysis

- X means l, F means v, B means f?

itwas**discXosed**yesterdaythatse**FeraXinBormaX**bltdirectcontactshaFebeenmadewiZWYoXiticaXreYrepresentatiFesoBtheFietconJinmoscow

- Therefore:

itwas**disclosed**yesterdaythat**severalinformal**bltdirectcontactshavebeenmadewiZWYoliticalreYrepresentativesofthevietconJinmoscow

# Example Cryptanalysis

- I means u, Z means t, W means h, Y means p?

it was disclosed yesterday that several informal **but** direct contacts have been made **with** political representatives of the vietcon in moscow

- Therefore:

it was disclosed yesterday that several informal **but** direct contacts have been made **with** political representatives of the vietcon in moscow

# Example Cryptanalysis

- Finally: J means g:  
itwasdisclosedyesterdaythatseveralinformalbutdirectc  
ontactshavebeenmadewithpoliticalrepresentativesofth  
**evietconJ**inmoscow
- Therefore (with spaces added):  
it was disclosed yesterday that several informal but  
direct contacts have been made with political  
representatives of the vietcong in moscow

CT437

# COMPUTER SECURITY AND FORENSIC COMPUTING

## BLOCK CIPHERS

Dr. Michael Schukat

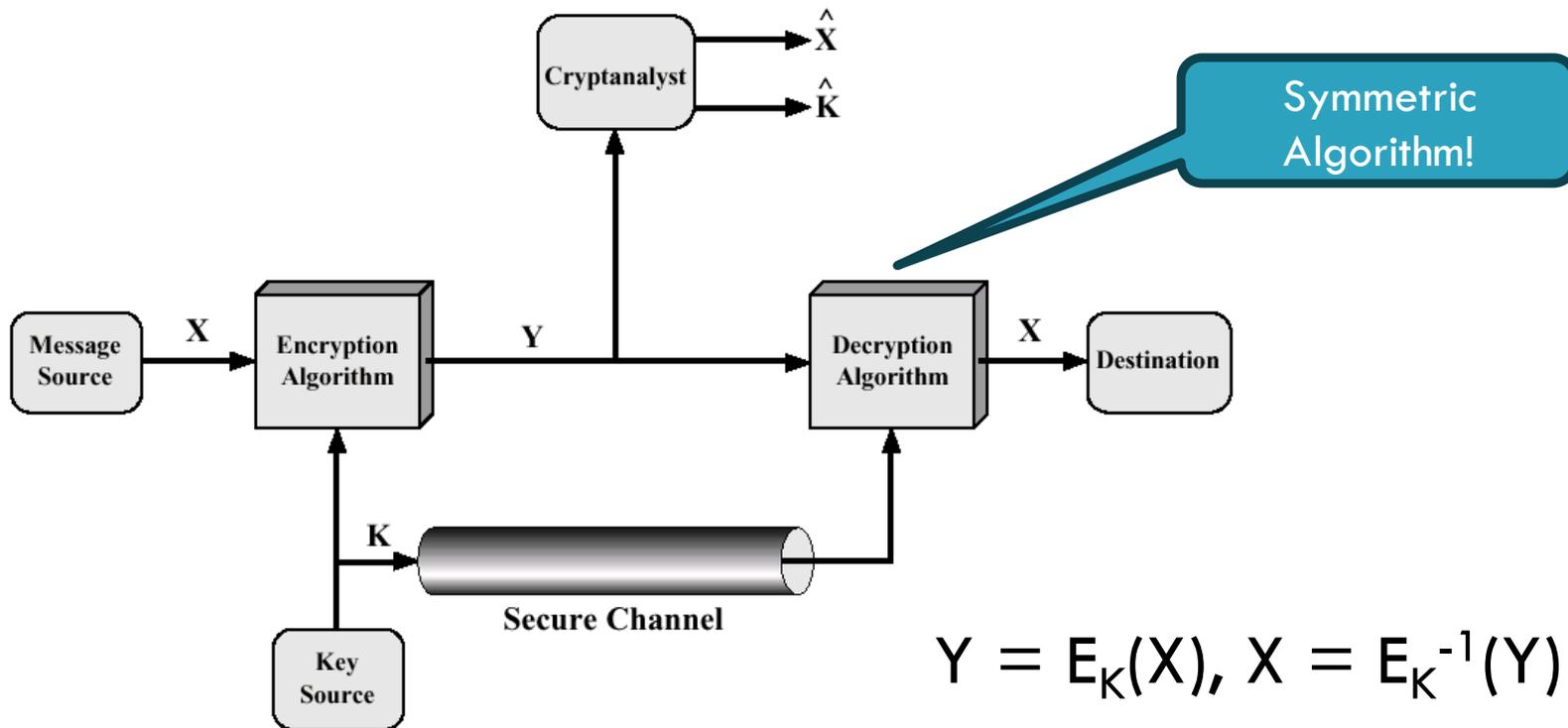


# Lecture Overview

2

- This lecture provides an introduction to one of the fundamental building blocks to provide confidentiality, namely **block ciphers**, thereby covering the following:
  - Symmetric versus Public Key Algorithms
  - Block ciphers versus Stream Ciphers
  - Building Blocks of modern Block Ciphers
  - Modes of operation of block ciphers
  - Examples for modern block ciphers

# Recall: Model of Conventional Cryptosystem



# Symmetric Key Algorithms

4

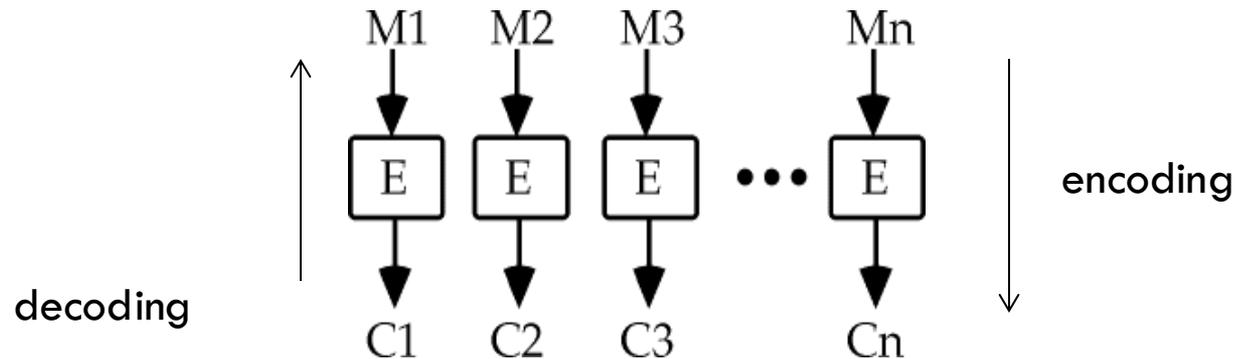
- Also called ciphers for traditional / conventional / single key / private key encryption
- Here the encryption key can be calculated from the decryption key and vice versa
  - ▣ Normally both keys are the same
- The algorithm / cipher itself is public, i.e. is not a secret
- If the key is disclosed, communications are compromised
- The key is also **symmetric**, parties are equal
- Hence methods does not protect sender from receiver forging a message & claiming is sent by sender  
→ nonrepudiation is usually not provided

# Public-Key Algorithms

- Also called ciphers for two key / asymmetric cryptography
- These involve the use of two keys:
  - ▣ a **public key**, which may be known by anybody, and can be used to encrypt messages, and verify signatures (later!)
  - ▣ a **private key**, known only to the recipient/owner, used to decrypt messages, and sign (create) signatures
- The keys are **asymmetric**, because they are not equal
- While the public and its private key are interlinked, it is mathematically very hard to recover the private key via its public key
- Public key algorithms are generally significantly slower than symmetric algorithms, therefore these are often used to securely convey symmetric algorithm' (session) keys
- More later!

# Block Ciphers versus Stream Ciphers

- In a block cipher the data (e.g. text, video, or a network packet) to be encrypted is broken into blocks  $M_1, M_2$ , etc. of  $K$  bits length, each of which is then encrypted
- The encryption process is like a substitution on very big characters – 64 bits or more



- In contrast, **stream ciphers** ( $\rightarrow$  next lecture) only process one bit or one byte at a time

# Example Block Cipher Transformation

7

P: 0000000000000000 .... 1111111111111111



C: 0101001010100101 .... 0110110110110010

- Block size  $K$  is 16 bits
- If there wasn't a cipher available for this transformation, we'd require a table with  $2^{16}$  entries
  - Not feasible
- Note that there are  $(2^{16})!$  possible substitutions

# Block Ciphers and Padding

- Messages are usually not multiples of  $K$  bits
- Padding is a way to take data that may or may not be a multiple of the block size for a cipher and extend it out so that it is
  - ▣ It is only applied to the last block that is being encrypted
- Padding must be reversable, i.e., one must be able to distinguish between relevant content and padding bytes in a block

# Padding Algorithms

9

- Let  $N$  be the number of bytes required to make a final block of data the same size as the block size
  - ▣ PKCS7 padding works by appending  $N$  bytes with the binary value of  $N$ ; example:

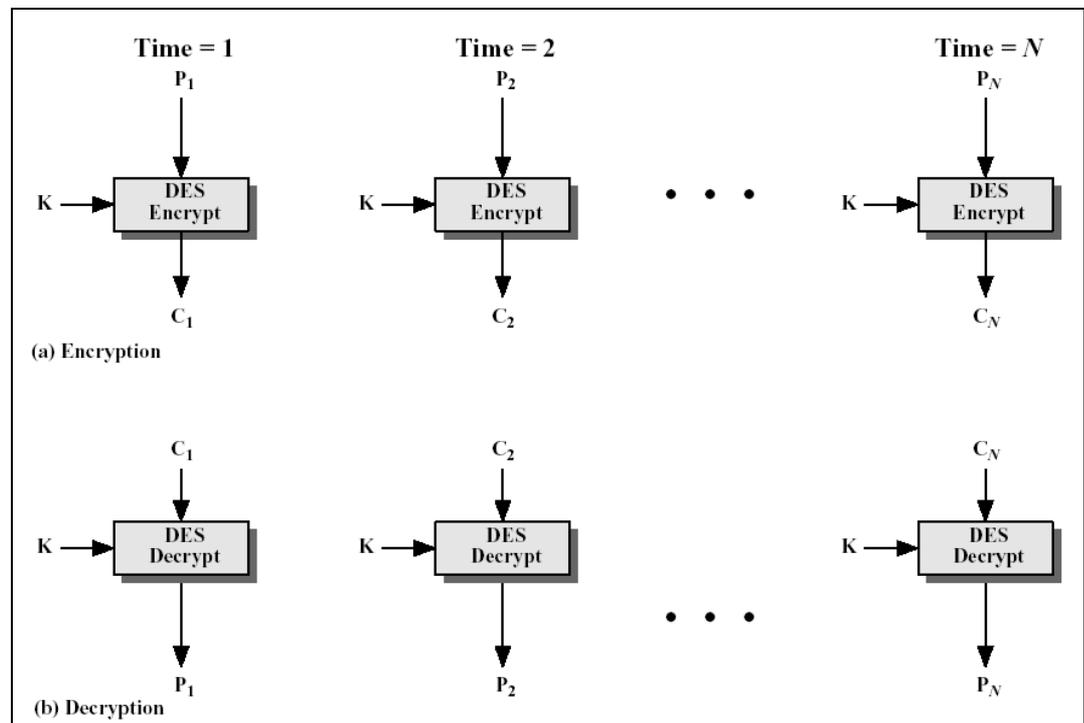
```
... | DD DD DD DD DD DD DD DD | DD DD DD DD 04 04 04 04 |
```

- ▣ ANSI X9.23 padding works by appending  $N-1$  bytes with the value of 0 and a last byte with the value of the binary value of  $N$ ; example:

```
... | DD DD DD DD DD DD DD DD | DD DD DD DD 00 00 00 04 |
```

# Modes of Operation: Electronic Codebook (ECB) Mode

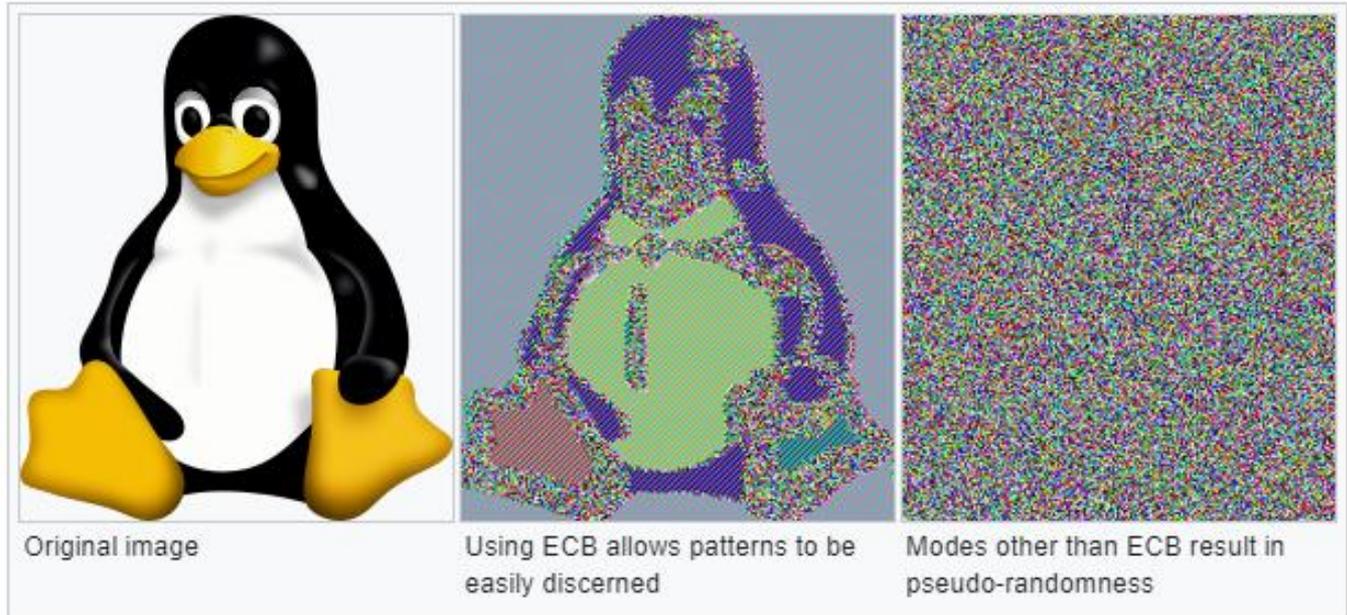
- These modes comprise different strategies on how to use block ciphers (to encode messages)
- What are the advantages / disadvantages of the ECB mode?
- Note that “DES” in the diagram on the right is just an example for a block cipher



# Characteristics and Limitations of ECB Mode

11

Source:  
Wikipedia



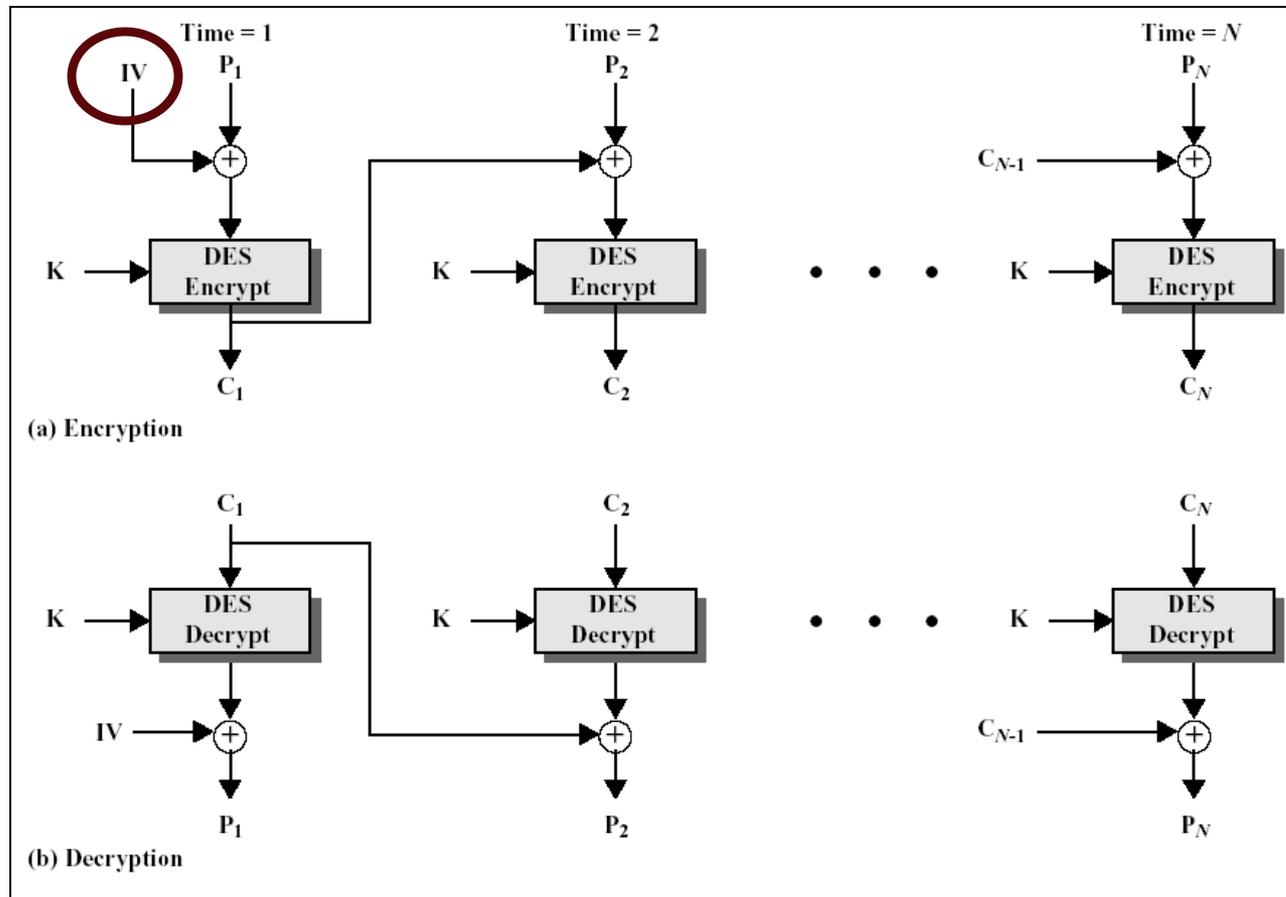
ECB	
Electronic codebook	
Encryption parallelizable	Yes
Decryption parallelizable	Yes
Random read access	Yes

# Why would one avoid the Electronic Codebook Mode?

12

- In ECB mode identical plaintext blocks result in identical ciphertext blocks
- An attacker, while not able to decode the ciphertext blocks, would conclude that the encoded data is repetitive / structured, i.e. could be an image
- However, random data (e.g. long cryptographic keys) could be still encoded in ECB
  - ▣ E.g., a 512-bit key could be encrypted using a 128-bit block cipher using ECB mode

# Modes of Operation: Cipher Block Chaining (CBC) Mode



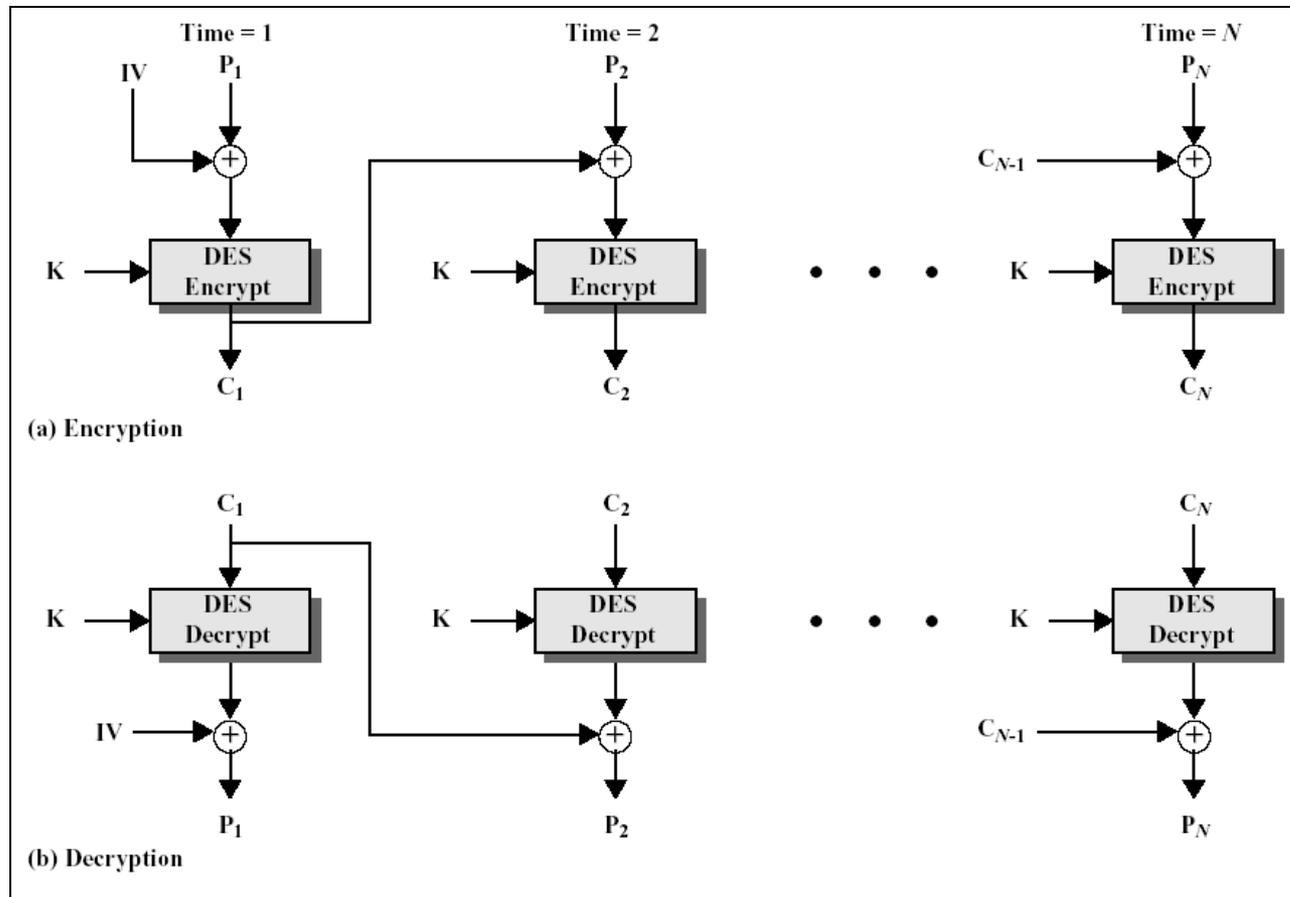
# The Initialisation Vector (IV)

14

- ❑ An IV is a block of bits that is used by several modes (including CBC) to randomise the encryption
- ❑ An initialization vector has different security requirements than a key, so the IV usually does not need to be secret
- ❑ For most block cipher modes it is important that an initialisation vector is never reused under the same key, i.e. it must be a **cryptographic nonce**
  - ❑ Hence distinct ciphertexts are generated even if the same plaintext is encrypted multiple times using the same key
- ❑ In data communication, the IV may be attached as plaintext to the encrypted data, and send to the receiver

# Modes of Operation: Cipher Block Chaining (CBC) Mode

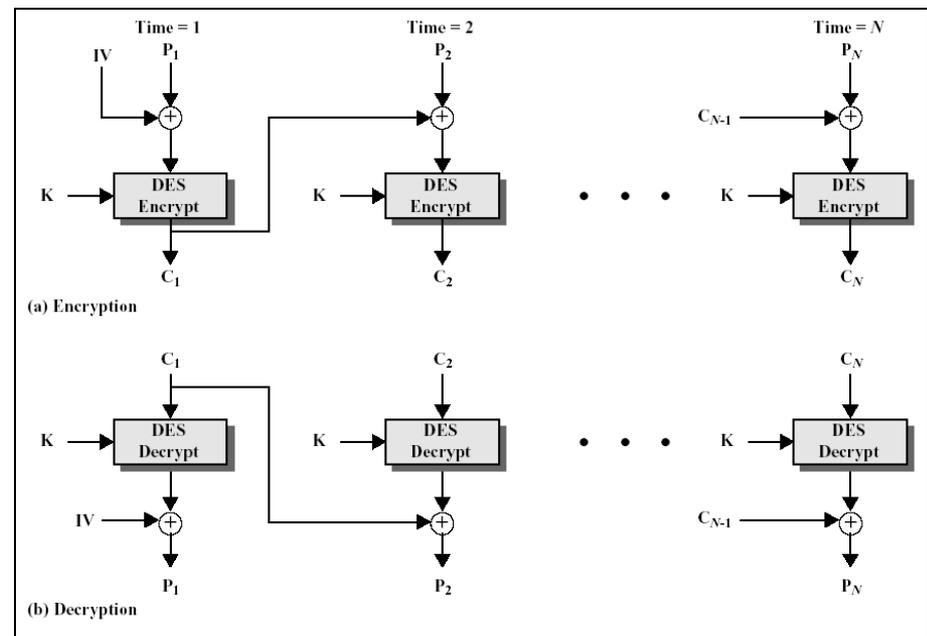
- What are the characteristics of the CBC mode?



# Modes of Operation: Cipher Block Chaining (CBC) Mode

- ❑ For encryption, a one-bit change in a plaintext or IV affects all following ciphertext blocks
- ❑ Decrypting with the incorrect IV causes the first block of plaintext to be corrupt but subsequent plaintext blocks will be correct
  - ▣ This could be problematic if the IV was kept a secret and is used to expand the algorithm's key space

CBC	
Cipher block chaining	
Encryption parallelizable	No
Decryption parallelizable	Yes
Random read access	Yes

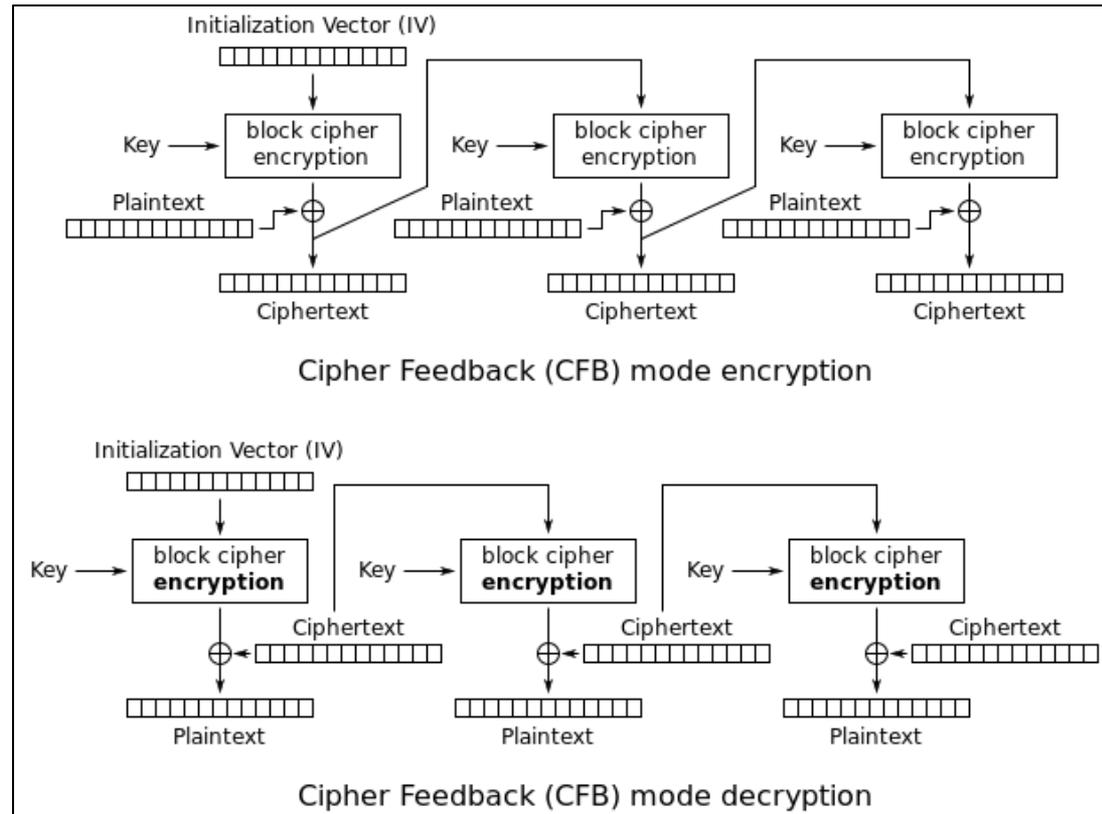


# Full Block Cipher Feedback (CFB) Mode

## Mode

17

- Note that CFB only requires block encryption for both encoding and decoding

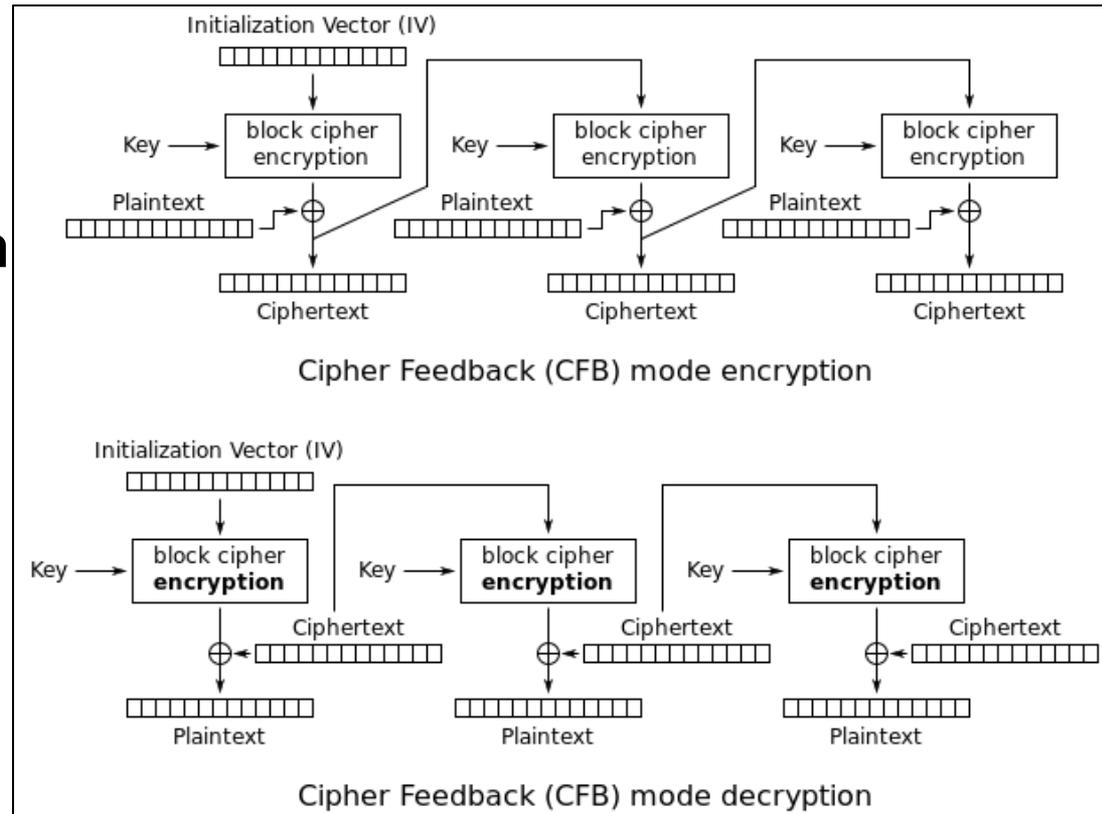


# Full Block Cipher Feedback (CFB) Mode

18

- This mode is particularly useful, when decryption needs to be fast (i.e., parallelisable)
- Note that CFB only requires block encryption for both encoding and decoding

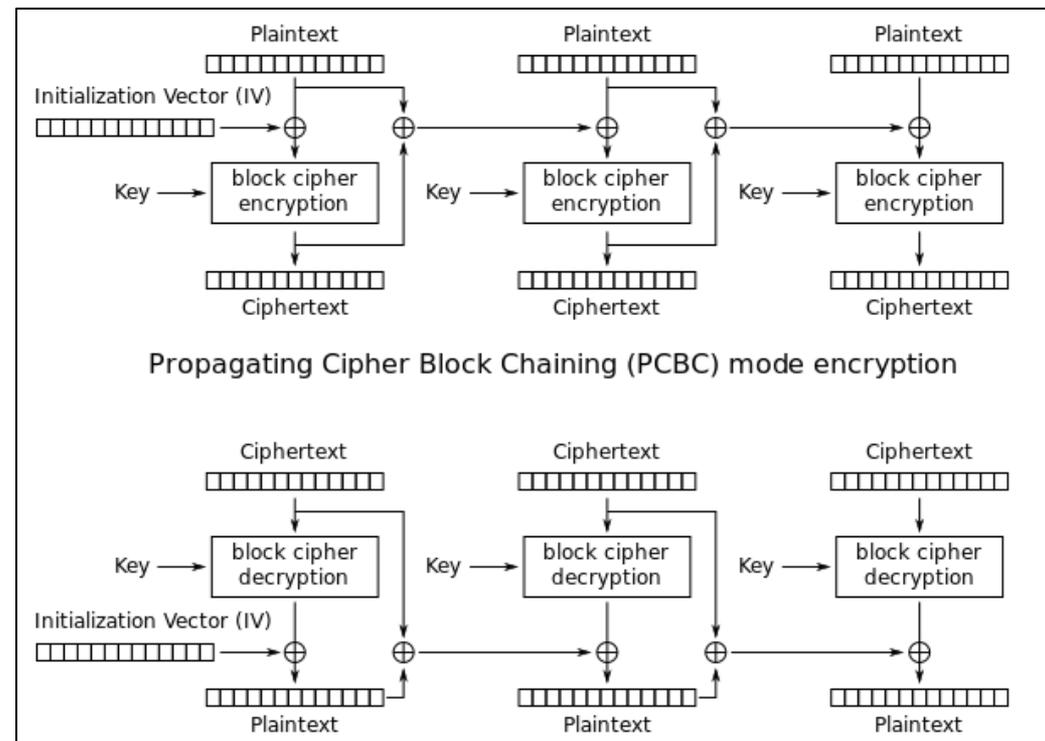
CFB	
Cipher feedback	
Encryption parallelizable	No
Decryption parallelizable	Yes
Random read access	Yes



# Propagating Cipher Block Chaining (PCBC) Mode

19

- This mode fixes the IV problem of CBC, i.e., decrypting PCBC with the incorrect IV causes all blocks of plaintext to be corrupt

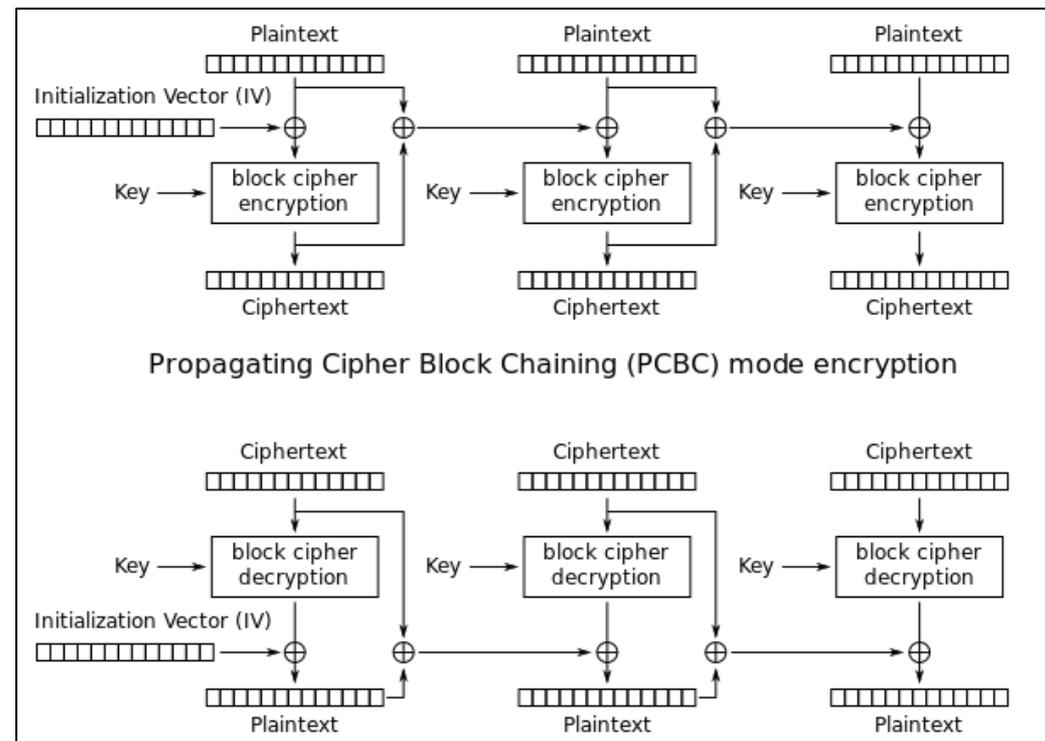


# Propagating Cipher Block Chaining (PCBC) Mode

20

- ❑ This mode fixes the IV problem of CBC, i.e., decrypting PCBC with the incorrect IV causes all blocks of plaintext to be corrupt

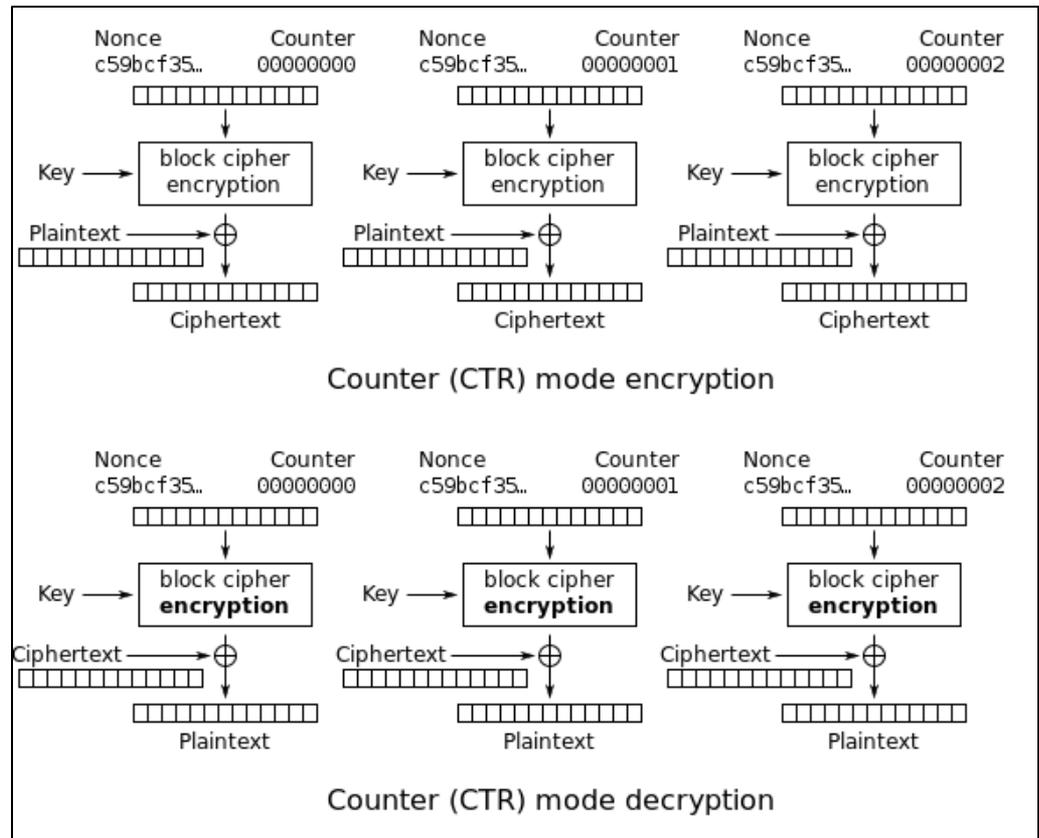
PCBC	
Propagating cipher block chaining	
Encryption parallelizable	No
Decryption parallelizable	No
Random read access	No



# Counter (CTR) Mode

21

- Here the random nonce (which is equivalent to an IV) is complemented with an incremented counter value
- Note that CTR only requires block encryption for both encoding and decoding

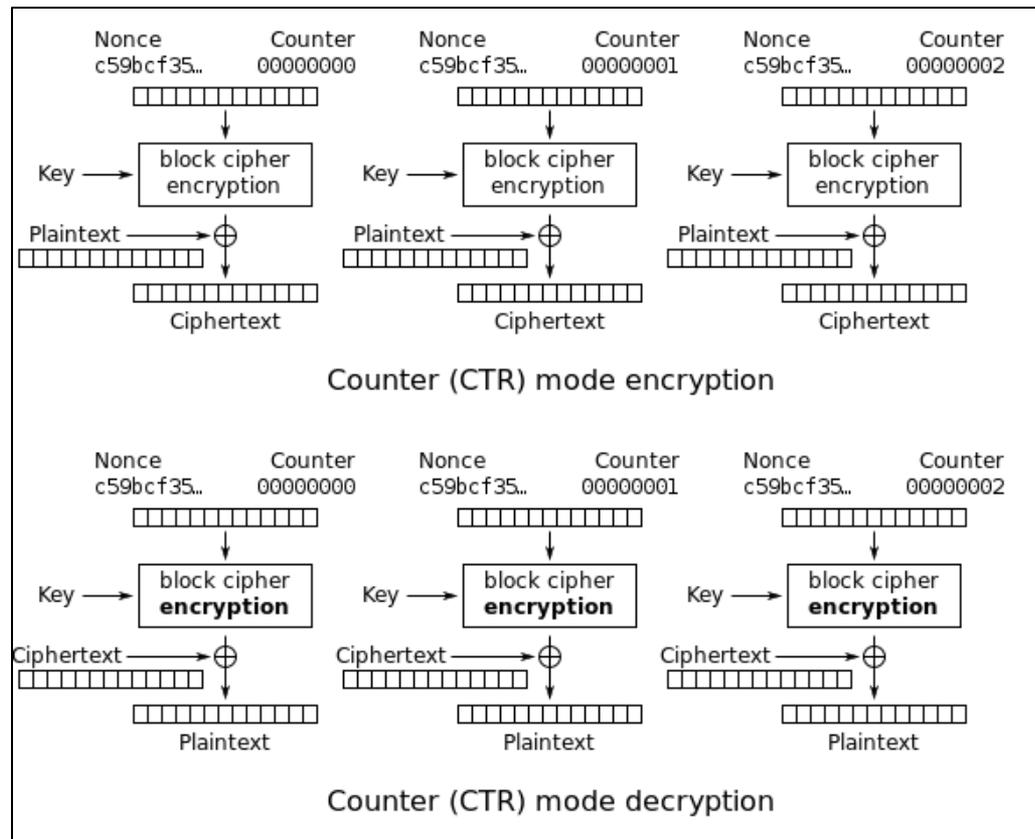


# Counter (CTR) Mode

22

- Here the random nonce (which is equivalent to an IV) is complemented with an incremented counter value
- Note that CFB only requires block encryption for both encoding and decoding

CTR	
Counter	
Encryption parallelizable	Yes
Decryption parallelizable	Yes
Random read access	Yes



# Another Question ...

23

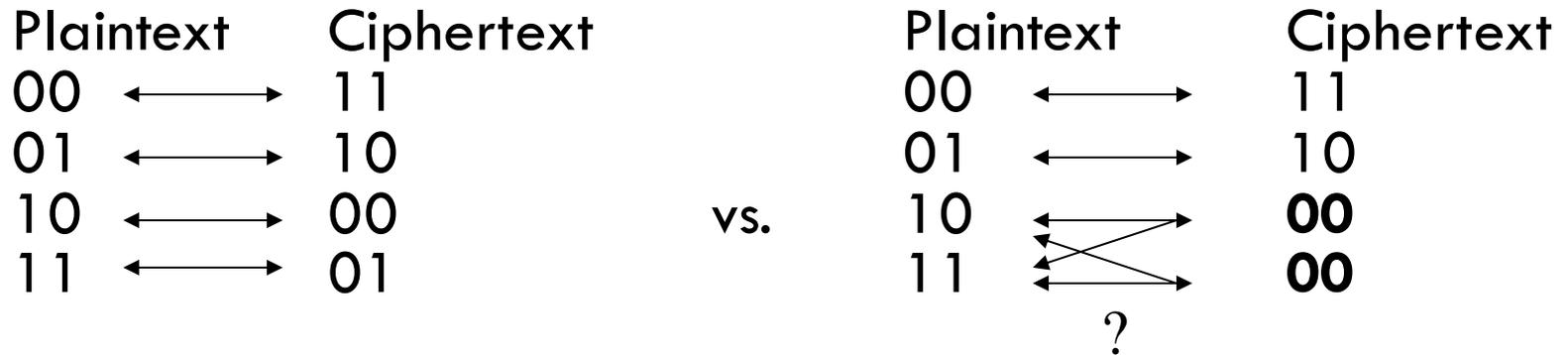
- Assume you have a large data file stored on your computer that needs to be encrypted / decrypted on-the-fly, potentially with random block access
- You pick a suitable block cipher for this task
- **Which mode of operation would you choose?**

# Building Block Ciphers

- Confusion, diffusion and the avalanche effect
- Using SP-Networks
- Using Feistel Networks

# Important Block Cipher Principle

- Transformations must be reversible (“non-singular”), e.g.,



- There must be a 1:1 association between a n-bit plaintext and an-n-bit ciphertext, otherwise mapping (encryption) is irreversible

# Confusion and Diffusion

- A block cipher needs to completely obscure the statistical properties of original message (obviously)
- Claude Shannon introduced two terms:
  - ▣ **Diffusion** seeks to make the statistical relationship between the plaintext and ciphertext as complex as possible
  - ▣ **Confusion** seeks to make the relationship between the statistics of the ciphertext and the value of the encryption key as complex as possible
- Both thwart attempts to deduce the key
- This can be achieved, by applying a cryptographic operation iteratively multiple times, i.e. over multiple rounds
  - ▣ See also the avalanche effect (next slide)

# The Avalanche Effect

- Practically, confusion and diffusion provide for encryption algorithms, where a slight change in either the key or the plaintext results in a significant change in the ciphertext
- The table shows some characteristics of the DES algorithm (later), that encrypts a block over 16 rounds
- A swap of a single bit either in the key or in the plaintext results in an incrementally growing change in the ciphertext (avalanche effect)

(a) Change in Plaintext		(b) Change in Key	
Round	Number of bits that differ	Round	Number of bits that differ
0	1	0	0
1	6	1	2
2	21	2	14
3	35	3	28
4	39	4	32
5	34	5	30
6	32	6	32
7	31	7	35
8	29	8	34
9	42	9	40
10	44	10	38
11	32	11	31
12	30	12	33
13	30	13	28
14	26	14	26
15	29	15	34
16	34	16	35

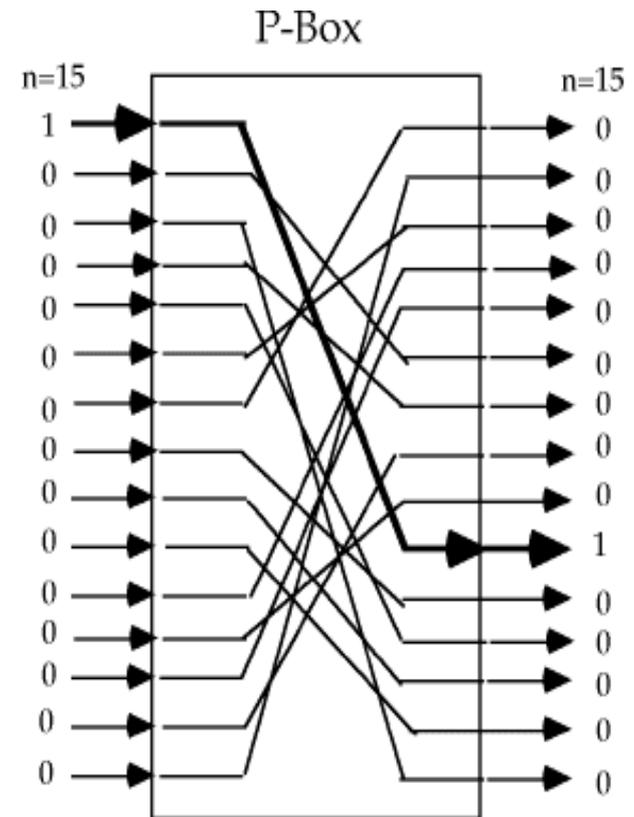
# Block Cipher Building Blocks

28

- In order to build a block cipher efficiently, one needs robust building blocks, that can be easily implemented and tested
  - ▣ The robustness of the block cipher depends on the robustness of its components
- These blocks are combined or iterated through creating a block cipher
- The most common building blocks are:
  - ▣ P-Boxes
  - ▣ S-Boxes
  - ▣ Feistel ciphers

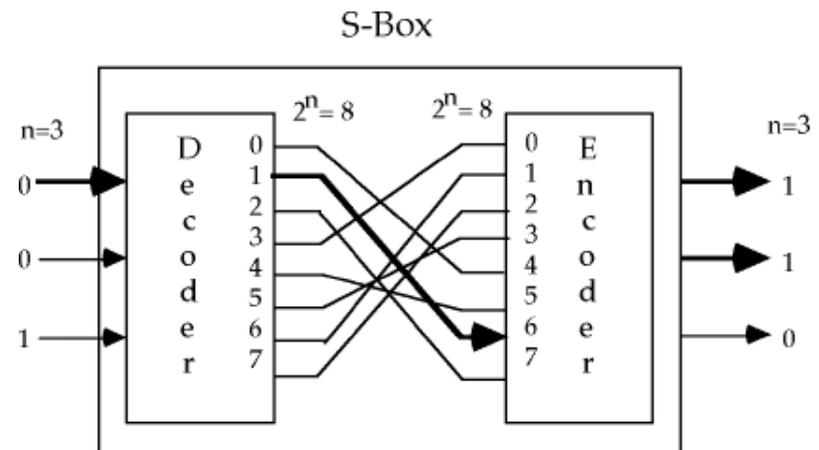
# The Permutation Operation

- A binary word (i.e. block) has its bits reordered (permuted)
  - ▣ Similar to classical transposition ciphers
- This operation is represented by a **P-box** (see diagram)
- Here the re-ordering / internal wiring forms the key
- The example shown allows for  $15! = 1,307,674,368,000$  combinations

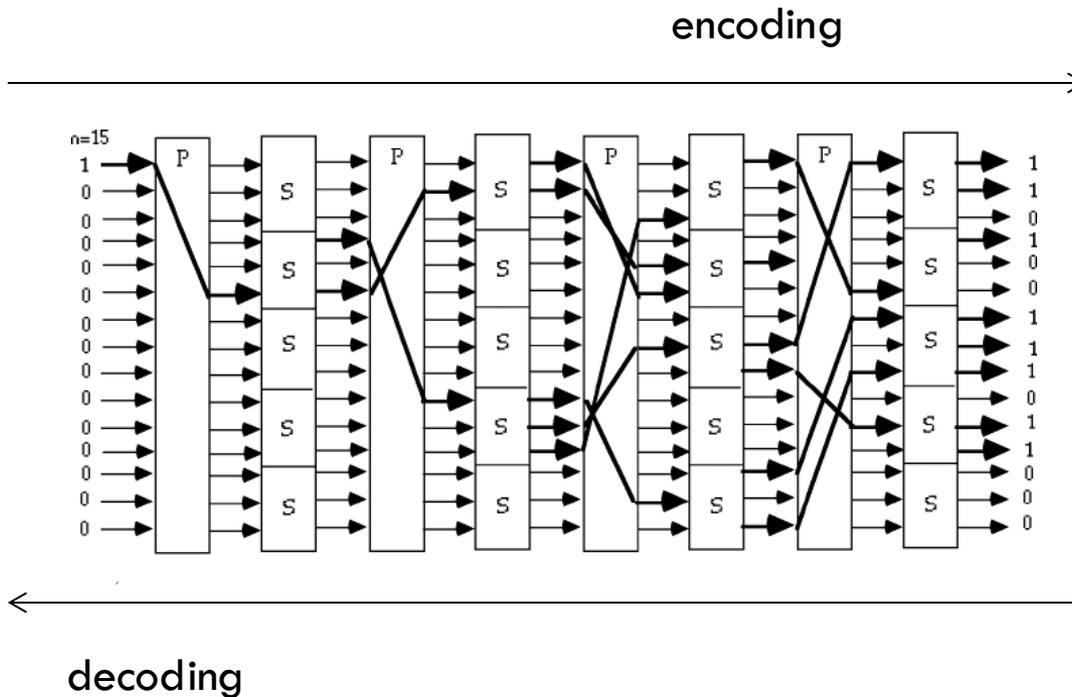


# The Substitution Operation

- A binary word is replaced by some other binary word
- Similar to classical substitution ciphers
- This operation is represented by an **S-box**
- Here the re-ordering / internal wiring forms the key
- The box shown allows for  $8! = 40320$  combinations



# Substitution-Permutation Network



- ◆ The key describes the internal wiring of all S-boxes and P-boxes
- ◆ The same key can be used for encoding and decoding, hence it is a **private key encryption algorithm**
- ◆ The direction of the process determines encoding / decoding

Question:

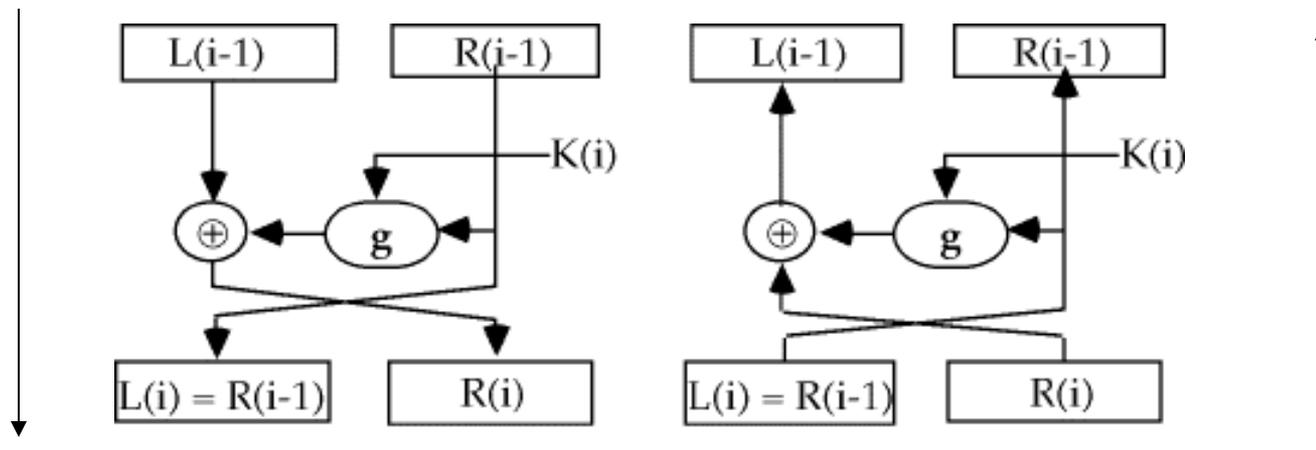
- How big is the key space for this arrangement?
- How many bits are needed to describe a single S or P box?
- What is the total number of bits required to describe all boxes?

# The Feistel Cipher

- In practice, we need algorithms that can decrypt and encrypt messages using similar code / hardware for both
- An S-P network as seen in the example cannot be easily reversed when implemented in hardware / software
  - ▣ i.e. one needs different functions for encoding / decoding
- In contrast, a **Feistel cipher** is an invertible cipher structure which adapts Shannon's S-P network in an easily invertible structure for encoding and decoding
  - ▣ In fact, it can use other cryptographic building blocks
- It is based on the concept of the **invertible product cipher**
- It was invented by Horst Feistel, who worked at IBM Thomas J Watson Research Labs in early 70's

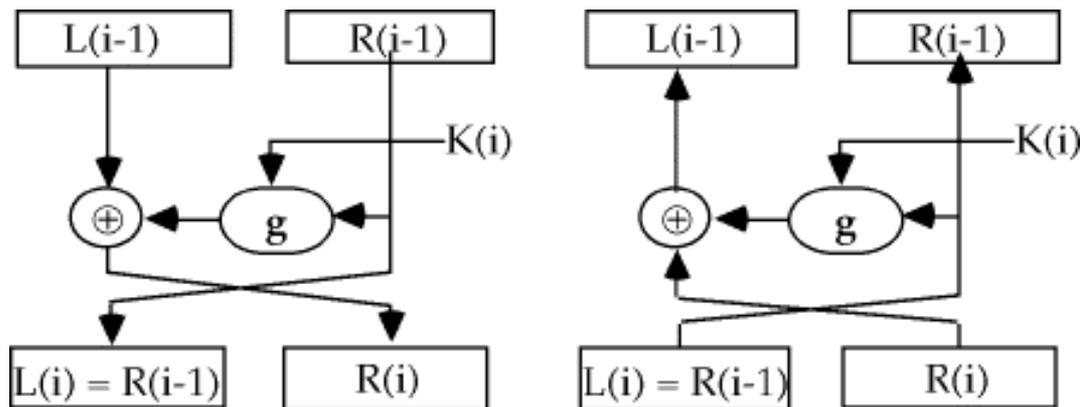
# The Feistel Cipher – A Single Round

- The idea is to partition the input block into two halves,  $L(i-1)$  and  $R(i-1)$ , and use only  $R(i-1)$  in the  $i^{\text{th}}$  round (part) of the cipher
- The function  $g$  incorporates the equivalent of one stage of the S-P network, controlled by part of the key  $K(i)$  known as the  $i^{\text{th}}$  subkey



# The Feistel Cipher – A single Round

- A round of a Feistel cipher can be described functionally as:
  - ▣  $L(i) = R(i-1)$
  - ▣  $R(i) = L(i-1) \text{ EXOR } g(K(i), R(i-1))$



# Recap: Symmetry of Bitwise EXOR

- $A \text{ EXOR } B = C$
- $A \text{ EXOR } C = B$
- $C \text{ EXOR } B = A$

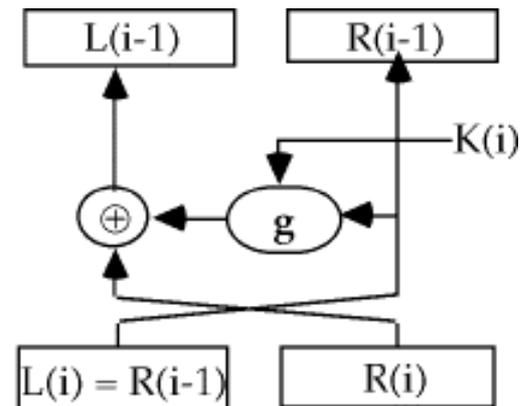
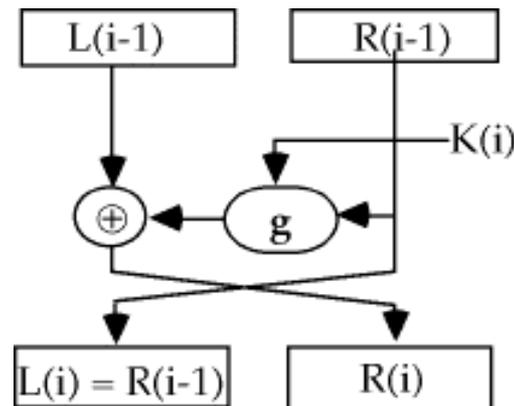
	0	1
0	0	1
1	1	0

# In-Class Activity: Feistel Cipher – Single Round

- Encoding of 01011110:
  - ▣  $L(i - 1) = 0101$
  - ▣  $g(K(i), R(i-1)) = 1001$
  - ▣  $R(i) = ?$
  - ▣ Therefore 01011110 becomes ?

$$R(i - 1) = 1110$$

$$L(i) = ?$$



# Example Feistel Cipher – Single Round

## □ Encoding of 01011110:

$$\square L(i - 1) = 0101 \qquad R(i - 1) = 1110$$

$$\square g(K(i), R(i-1)) = 1001 \qquad L(i) = 1110$$

$$\square R(i) = 0101 \text{ XOR } 1001 = 1100$$

□ Therefore 01011110 becomes 11101100

## □ Decoding of 11101100:

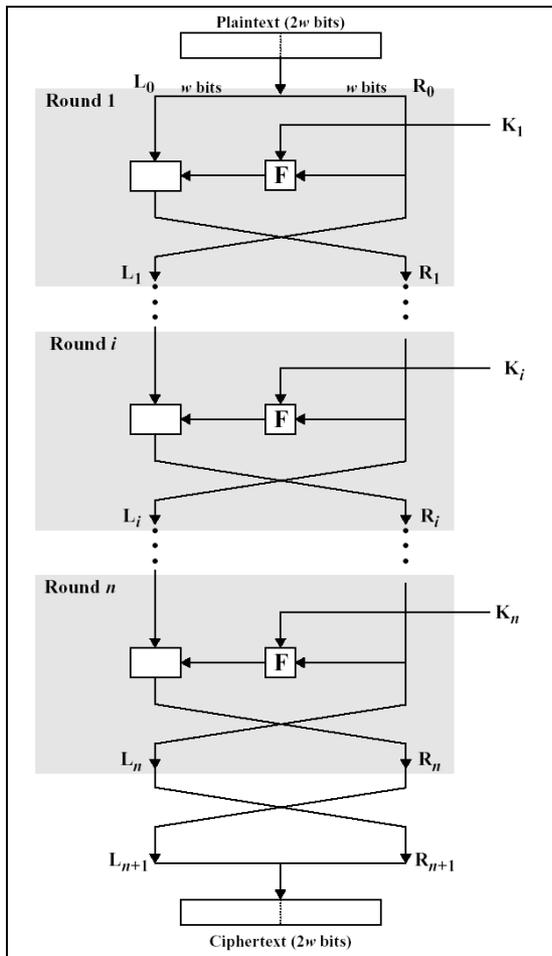
$$\square L(i) = 1110 \qquad R(i) = 1100$$

$$\square g(K(i), R(i-1)) = 1001 \qquad R(i - 1) = 1110$$

$$\square L(i - 1) = 1100 \text{ XOR } 1001 = \underline{0101}$$

□ Therefore 1110 1100 becomes 01011110

# Feistel Network



- Common structure of many modern block ciphers
- It performs multiple transformations (single rounds) sequentially, whereby the output of the  $i^{\text{th}}$  round becomes the input of the  $(i+1)^{\text{th}}$  round
- Every round gets its own subkey, which is derived from the master key
- Decryption process goes from bottom to top

# Feistel Cipher Design Elements

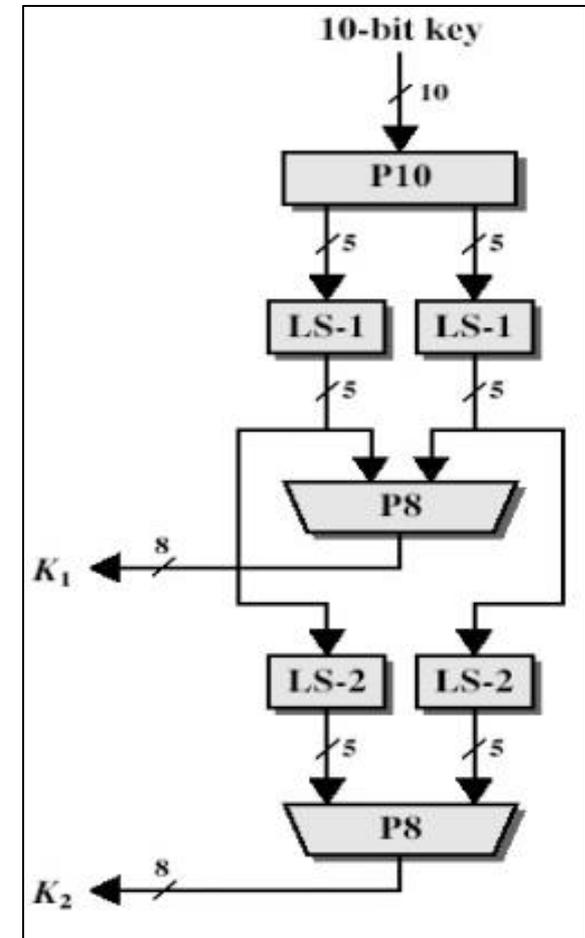


These include

- ❑ Block size (typically 64 – 256 bits)
- ❑ Key size (typically 80 – 256 bits)
- ❑ Number of rounds (typically  $> 16$ )
- ❑ Subkey generation algorithm
- ❑ Round function

# Simple Methods for Subkey Generation

- Here two 8-bit round keys ( $K_1$  and  $K_2$ ) are derived from a 10-bit (master) key:
  - The 10-bit master enters the permutation box (P10)
  - The output is split into 2 parts
  - Each part is left-rotated by one bit (LS-1)
  - Both parts are concatenated and passed into a permutation box (P8)
  - P8 has eight outputs, which make  $K_1$



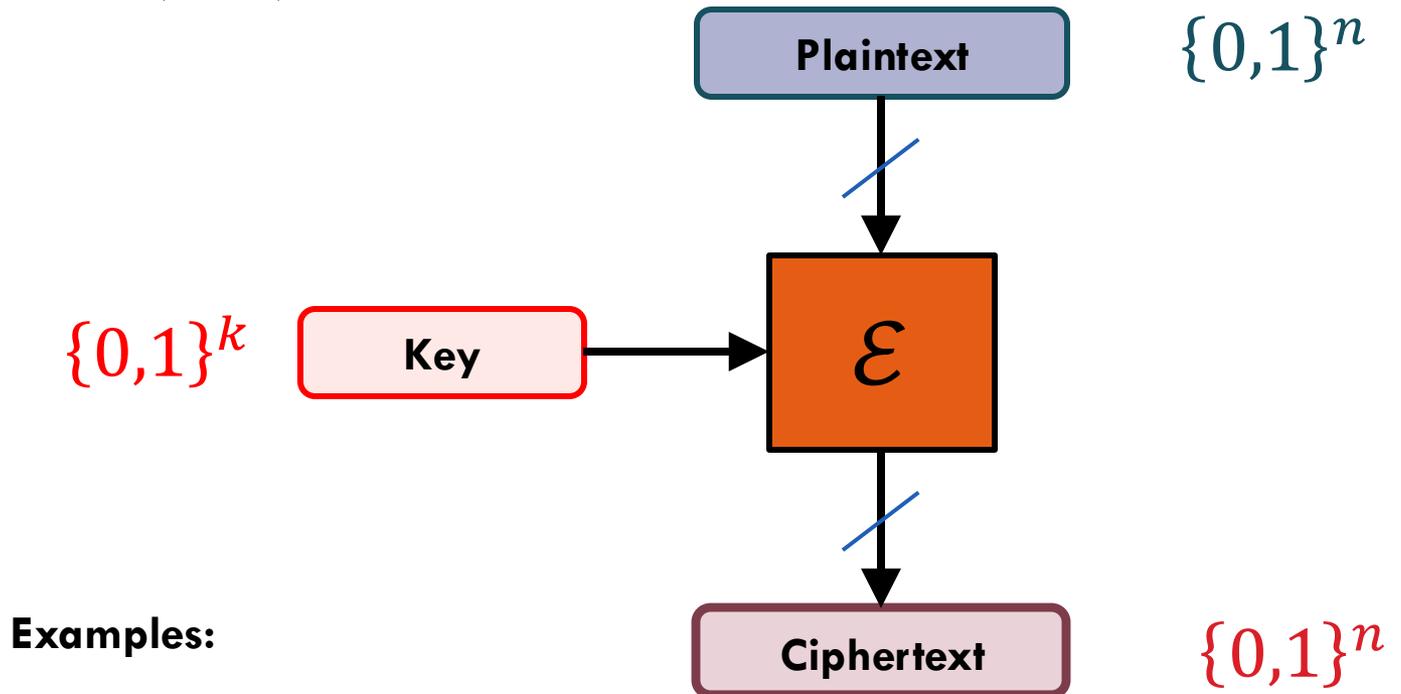
# Block Cipher Examples

- Data Encryption Standard (DES)
- AES

# Common Block Cipher Key and Block Lengths

Key length  $k = 80, 128, 192, 256$

Block length  $n = 64, 128, 256$



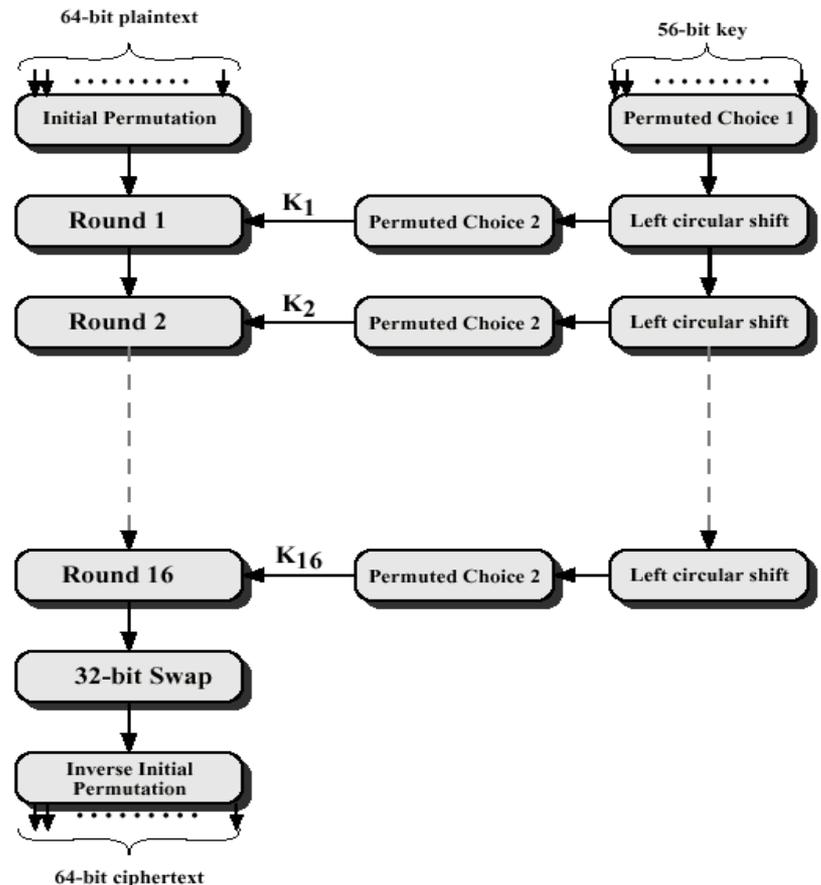
**Examples:**

DES:  $k = 56, n = 64$

AES:  $k = 128, 192, 256, n = 128$

# Data Encryption Standard (DES)

- ❑ DES was the first block cipher widely used in industry
- ❑ Introduced in 1976
- ❑ 64-bit block length
- ❑ 56-bit key length
- ❑ Feistel network with 16 rounds and 48-bit subkeys



# The DES Challenge

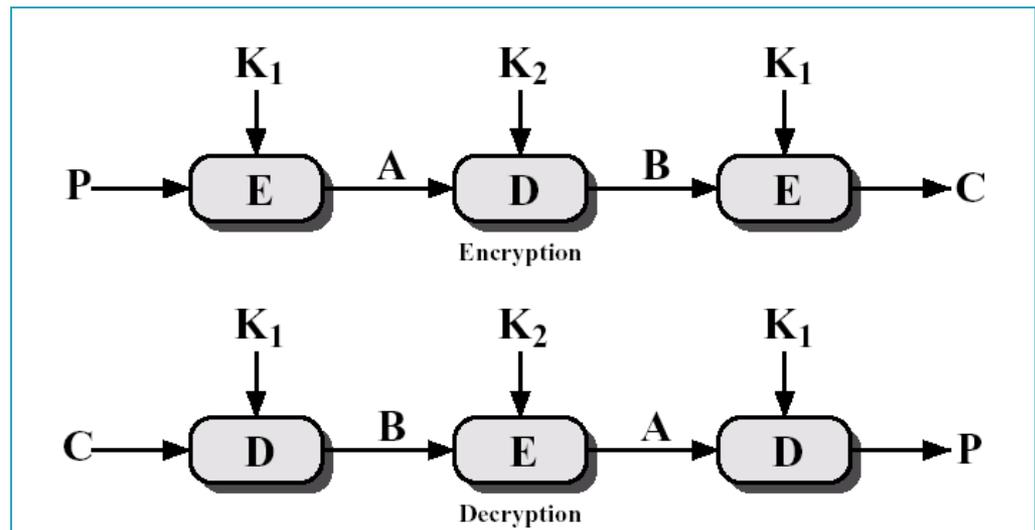
45

- Contest to demonstrate to the US government that 56-bit DES is an ineffective form of encryption
- The goal of the challenge was to decrypt secret messages which had been encrypted with DES

Name	When	Duration	Hardware used
DES I Challenge	June 1997	140 days	Up to 70,000 PC
DES II Challenge	February 1998	41 days	?
DES Challenge II-2	July 1998	56 hours	Custom FPGA Design
DES Challenge III	January 1999	22 hours	~100,000 PC

# Triple DES

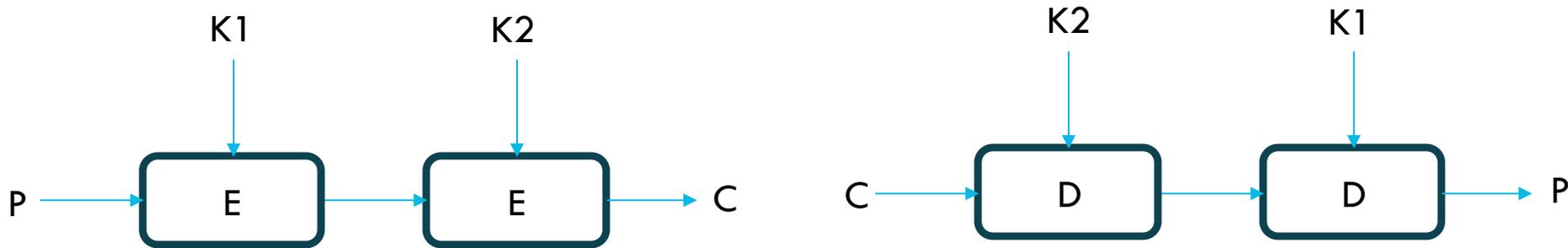
- ❑ DES has been widely used and implemented across many OS and crypto libraries, so attempts were made to increase its active life span
- ❑ This resulted in Triple-DES
- ❑ It is based on three processing stages
- ❑ Note the symmetry in the encoding and decoding process
- ❑ In principal, this concatenation can be applied to every private key block cipher
- ❑ There are 2 common keying options:
  - ▣ 2 keys (as shown in the figure)
  - ▣ 3 keys (one for each stage)



# Double-DES and the Meet-in-the-Middle Attack

47

- Double DES uses two instances of DES with different keys



- While this algorithm uses two independent keys, it is not as sound as it looks
- It is vulnerable to the meet-in-the-middle attack, where an attacker has access to  $P$  and  $C$ , and tries to determine  $K1$  and  $K2$

# Double-DES and the Meet-in-the-Middle Attack

48

- This attack is an example of a space-time tradeoff, where the adversary does the following:
  1. Encrypt  $P$  using every possible key, and copy each key and the resulting cyphertext into a table  $T1$ 
    - $T1$  will have 2 columns and  $2^{56}$  rows
  2. Decrypt  $C$  using every possible key, and copy each key and the resulting plaintext into a table  $T2$ 
    - Again,  $T2$  will have 2 columns and  $2^{56}$  rows
  3. Check for identical cyphertext / plaintext entries in  $T1$  and  $T2$
  4. Their corresponding keys  $K1$  and  $K2$  are key candidates and can be further validated using other plaintext/cyphertext pairs
- Overall, this process requires  $2^{56}$  encryption and  $2^{56}$  decryption attempts, so overall  $2 \times 2^{56} = 2^{57}$  attempts (rather than  $2^{112}$  attempts) are required
- Note that this attack can also be applied to Triple DES, but it would require  $2^{2 \times 56}$  attempts

# Advanced Encryption Standard (AES)

- Successor of DES since 2002
- Based on a S-P network
- Block size is 128-bit
- Key length is configurable can be 128, 192 or 256 bit
- Stronger & faster than Triple-DES
  - ▣  $2 * 56! \ll 128!$
- Envisaged active life until ~2030
- Full specification & design details public
- Algorithm has reference implementations across many programming languages

50

# Breaking Block Ciphers

# Why does Block and Key Length matter?

- Cryptographic algorithms with short block length can be tackled as seen with the substitution cipher
- Large keys and large blocks prevent **brute-force attacks / searches**
  - ▣ Take the ciphertext and try all possible key combinations (or block permutations), until the text is successfully decoded (e.g. until the decryption provides meaningful text)

# Brute Force Search / Attacks

- DES uses 56-bit key has a key space that contains  $2^{56}$  ( $= 7.2 \times 10^{16}$ ) keys
  - ▣ Deemed unsafe since the 1990s
- Triple-DES uses two 56-bit keys. and its key space contains  $2^{112}$  ( $= 5.1 \times 10^{33}$ ) keys
  - ▣ Its use will be prohibited from 2024!
- AES-128 key space contains  $2^{128}$  ( $= 3.4 \times 10^{38}$ ) keys
  - ▣ Generally accepted minimum key length today
- Top secret information requires the use of either AES-192 or AES-256

# Brute Force Search / Attacks

- Always possible to simply try every key
- Most basic attack, effort proportional to key size
- Assume that you either know or recognise plaintext
- GPUs are very good at this task, for example a single RTX 3070 GPU can crack a DES key in  $\sim 215$  days

Key Size (bits)	Number of Alternative Keys	Time required at 1 decryption/ $\mu$ s	Time required at $10^6$ decryptions/ $\mu$ s
32	$2^{32} = 4.3 \times 10^9$	$2^{31} \mu\text{s} = 35.8$ minutes	2.15 milliseconds
56	$2^{56} = 7.2 \times 10^{16}$	$2^{55} \mu\text{s} = 1142$ years	10.01 hours
128	$2^{128} = 3.4 \times 10^{38}$	$2^{127} \mu\text{s} = 5.4 \times 10^{24}$ years	$5.4 \times 10^{18}$ years
168	$2^{168} = 3.7 \times 10^{50}$	$2^{167} \mu\text{s} = 5.9 \times 10^{36}$ years	$5.9 \times 10^{30}$ years
26 characters (permutation)	$26! = 4 \times 10^{26}$	$2 \times 10^{26} \mu\text{s} = 6.4 \times 10^{12}$ years	$6.4 \times 10^6$ years

# Side-Channel Attacks

54

- ❑ AES is cryptographically sound and there is no practical cryptographic "break" that is faster than a brute-force attack
- ❑ However, there are possible side-channel attacks
- ❑ Generally, these are attacks on implementations of a cipher on hardware or software systems that inadvertently leak data, e.g.
  - ▣ Timing information (how long does an encryption take)
  - ▣ Cache and memory content (→ HeartBleed)

# Timing Attacks

- Here the attacker attempts to compromise a cryptosystem by analysing the time taken to execute a cryptographic algorithm
- Every logical operation in a computer takes time to execute, and the time can differ based on the input
- With precise measurements of the time for each operation, an attacker can work backwards to the input

# Example: Insecure String Comparison (Wikipedia)

56

- Spot the difference?

```
bool insecureStringCompare(const void *a, const void *b, size_t length) {
    const char *ca = a, *cb = b;
    for (size_t i = 0; i < length; i++)
        if (ca[i] != cb[i])
            return false;
    return true;
}
```

versus

```
bool constantTimeStringCompare(const void *a, const void *b, size_t length) {
    const char *ca = a, *cb = b;
    bool result = true;
    for (size_t i = 0; i < length; i++)
        result &= ca[i] == cb[i];
    return result;
}
```

- Note that many such functions in normal (rather than crypto-) libraries are unsafe
  - ▣ Example memcpy() as used in C

# Timing Attacks

57

- In principal, timing attacks can be performed
  - ▣ remotely (e.g. a client measures the response time of a server that encrypts a message)
  - ▣ locally (i.e. in the host machine itself)
- Remote timing attacks are not practical, as variable OS and network latencies effect any measurement
- Local attacks are better, but require the exploit to be installed on the host under attack
- Saying this, many modern CPUs have built-in hardware instructions for AES, which protect against timing-related side-channel attacks

# FYI: More Side-Channel Attacks

58

- **Transient execution CPU vulnerabilities** are vulnerabilities in a computer system in which a speculative execution optimisation implemented in a microprocessor is exploited to leak secret data to an unauthorized party
  - ▣ Example Meltdown and Spectre attack
- In **cache timing attacks** an attacker process deliberately causes page faults and/or cache misses in the target process, and monitors the resulting changes in access times
  - ▣ This can be done despite both processes being otherwise isolated

CT437

# COMPUTER SECURITY AND FORENSIC COMPUTING

## STREAM CIPHERS

Dr. Michael Schukat



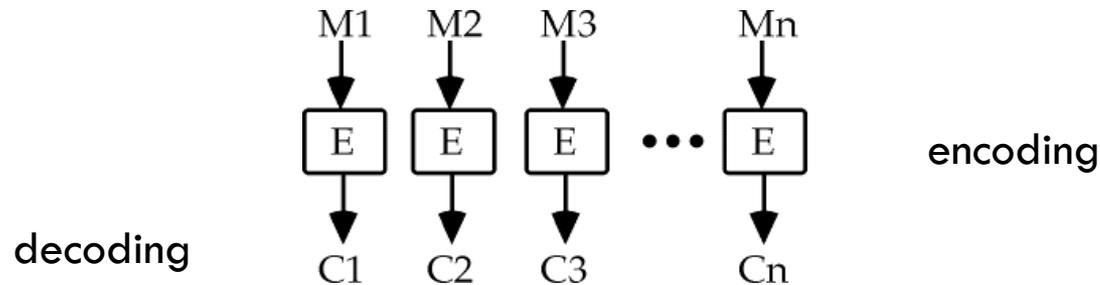
# Lecture Overview

2

- This slide decks covers the following topics:
  - Stream Ciphers and their implementation in
    - LFSR
    - NLFSR
    - RC4
  - Pseudorandom number generation principles

# Recap: Block Ciphers versus Stream Ciphers

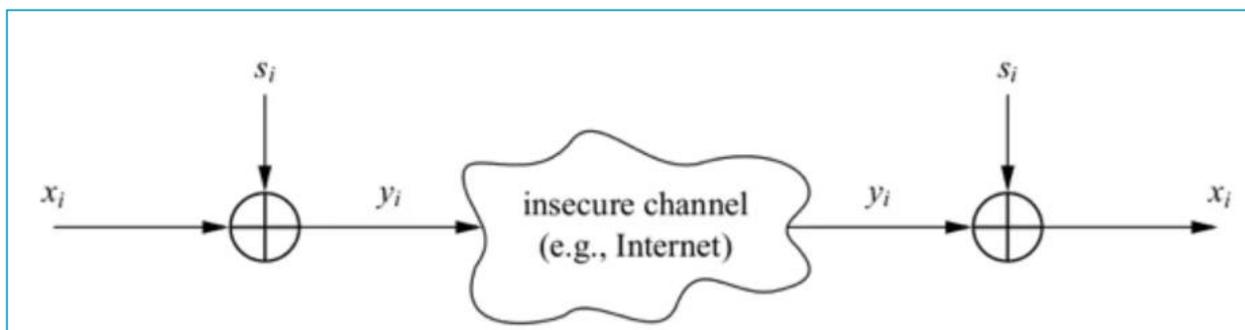
- In a block cipher the data (e.g. text, video, or a network packet) to be encrypted is broken into blocks  $M_1, M_2$ , etc. of  $K$  bits length, each of which is then encrypted
- The encryption process is like a substitution on very big characters – 64 bits or more



- In contrast, a **stream cipher** is a symmetric key cipher where plaintext digits are combined with a pseudorandom cipher digit stream (the **keystream**)
- Normally,
  - ▣ stream ciphers only process one bit or one byte at a time
  - ▣ the combining operation is an exclusive-or (XOR)

# Stream Ciphers

- Stream ciphers typically provide a (pseudo) random stream key generator that produces a pseudo-random digit sequence  $s_i$  ( $i = 1, 2, \dots$ )
- This stream is XORed digit-by-digit with the plaintext  $x$ :
$$y_i = x_i \text{ XOR } s_i$$
- The plaintext stream can be recovered by reapplying the XOR operation
- In modern stream ciphers, a digit is one bit (or one byte  $\rightarrow$  later)
- A random stream key completely destroys any statistically properties in the plaintext message
  - ▣ For a perfectly random keystream  $s_i$ , each  $y_i$  has a 50% chance of being 0 or 1
- But how can a pseudo-random sequence  $s_i$  be generated?



$x_i$	$s_i$	$y_i$
0	0	0
0	1	1
1	0	1
1	1	0

# Stream Cipher Performance

5

- Since an XOR operation of a single bit or byte can be done in a single CPU cycle,
  - ▣ the code size and complexity of a stream cipher mainly depends on the code size and complexity of the random number generator
  - ▣ the speed of a stream cipher mainly depends on the speed of the random number generator
- For comparison (based on some Intel Pentium architecture):

Cipher	Key length	Mbit/s
DES	56	36.95
3DES	112	13.32
AES	128	51.19
RC4 (stream cipher)	(choosable)	211.34

- Size and speed make stream ciphers very suitable for resource constrained devices (e.g., mobile phones, IoT devices)

# One-Time Pad

- The OTP is an encryption requires the use of a single-use pre-shared key that is equal to the size of the message being encrypted
- For the resulting ciphertext to be impossible to decrypt, the key must...
  - ▣ be at least as long as the plaintext (think of Vigenère and its weakness)
  - ▣ be
    - random (uniformly distributed in the set of all possible keys and independent of the plaintext)
    - entirely sampled from a non-algorithmic, chaotic source such as a hardware random number generator
    - pattern-less
  - ▣ never be reused in whole or in part (Coincidence counting -> next slide)
  - ▣ be kept completely secret by the communicating parties
- OTPs are not practical for practical reasons, therefore pseudo-random generators (PRG) are used
- PRGs are often based on Linear Feedback Shift Registers (LFSRs)

# Example Coincidence Counting

8

- ❑ Coincidence counting allows predicting the length of the key of a stream cipher, by comparing the ciphertext against itself with different offsets
- ❑ Assume ciphertext CXEKCWCOZKUCAYZEKW that has been encoded using a stream cipher with an unknown key
- ❑ Count the number of identical characters (matches) using different displacements of ciphertext:

- ❑ Displacement = 1

```
CXEKCWCOZKUCAYZEKW  
  CXEKCWCOZKUCAYZEKW
```

Matches: 0

- ❑ Displacement = 2

```
CXEKCWCOZKUCAYZEKW  
  CXEKCWCOZKUCAYZEKW
```

Matches: 1

- ❑ Displacement = 3

```
CXEKCWCOZKUCAYZEKW  
    CXEKCWCOZKUCAYZEKW
```

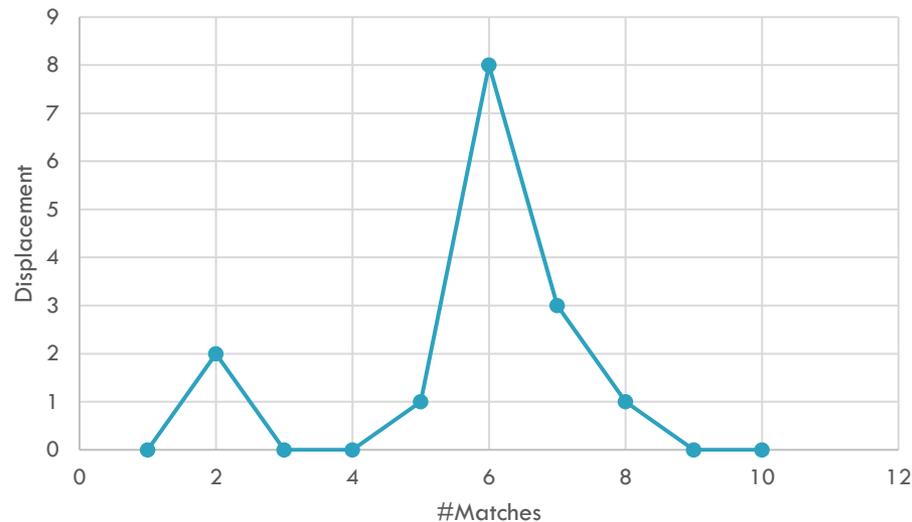
Matches: 0

- ❑ ...

# Example Coincidence Counting

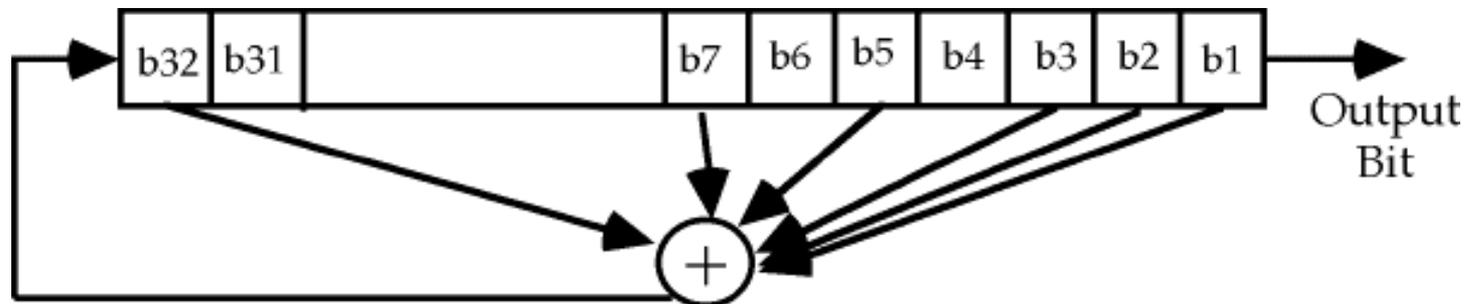
9

- If you line up the ciphertext with itself displaced by  $k$  ( $=$  key length) characters, then you get a match in the ciphertext (offset by  $k$  places) if there is a match in the plaintext (offset by  $k$  places)
  - ▣ With the non-uniformity of the frequency distribution of English letters there's about a 6% chance that those two positions have the same letter (the index of coincidence)
- In contrast, when you line up the ciphertext using a different displacement, the index of coincidence is much smaller, i.e.,  $1/256$ , if ciphertexts are bytes
- By counting the displacement over a long ciphertext stream,  $k$  can be determined



# Linear Feedback Shift Registers (LFSR)

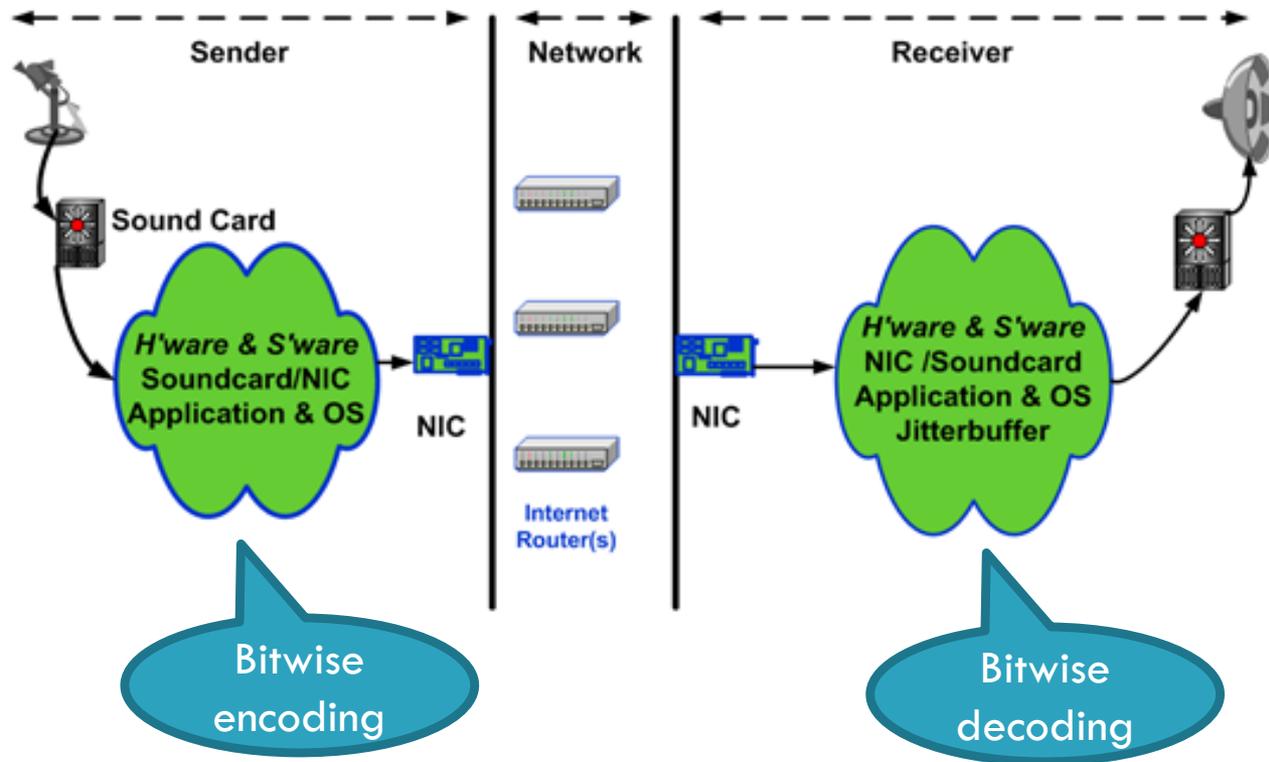
- A LFSR consists of a binary shift register of some length along with a linear feedback function (LFF) that operates on some of those bits
  - ▣ The most commonly used LFF is the XOR operation
- To get started the register is preset with a secret initialisation vector
- Each time a bit is needed,
  - ▣ a new bit is formed from the linear feedback function
  - ▣ all bits are shifted by one position (shifted right in the example below) with the new bit being shifted in
- The bit shifted out is used as the (pseudo-random) output of the LFSR
- A well-designed n-bit LFSR generates a pseudo-random sequence whose length correlates to n



# Example for an 8-Bit LFSR

- Initialisation vector:  $\underline{1}0\underline{1}0\underline{0}1\underline{1}0$  ( $B_7 \dots B_0$ )
- Feedback Function:  $B_7 \text{ XOR } B_4 \text{ XOR } B_1$
- Right shift after each cycle ( $B_0$  shifted out)
- Iteration 0:  $10100110$
- Iteration 1:  $\underline{0}10\underline{1}00\underline{1}1 \ggg 0$
- Iteration 2:  $\underline{0}01\underline{0}100\underline{1} \ggg 1$
- Iteration 3:  $\underline{0}00\underline{1}01\underline{0}0 \ggg 1$
- Iteration 4:  $10001010 \ggg 0$
- ...

# Example VoIP Encoding using a Stream Cipher



# Stream Ciphers in Practice

13

- In practice, one key is used to encrypt many messages
  - ▣ Example: Wireless communication
  - ▣ Solution: Use Initial vectors (IV)
  - ▣  $E_{\text{key}}[M] = [IV, M \oplus \text{PRNG}(\text{key} \parallel IV)]$ 
    - IV is sent in clear to receiver
    - IV needs integrity protection, but not confidentiality protection
    - IV ensures that key streams do not repeat, but does not increase cost of brute-force attacks
    - Without key, knowing IV still cannot decrypt
  - ▣ Need to ensure that IV never repeats! How?

# Example for a 16-bit LFSR written in C

14

```
#include <stdint.h>
#include <stdio.h>
int main(void) {
    uint16_t start_state = 0xACE1u; /* Any non-zero start state will work. */
    uint16_t lfsr = start_state;
    uint16_t bit, input, period = 0;
    printf("Enter LFSR IV as integer: "); scanf("%d", &input);
    if (input > 0) {
        start_state = input;
        lfsr = start_state;
    }
    do
    { /* LFF: B15 XOR B13 XOR B12 XOR B10 */
        bit = ((lfsr >> 0) ^ (lfsr >> 2) ^ (lfsr >> 3) ^ (lfsr >> 5)) & 1u;
        lfsr = (lfsr >> 1) | (bit << 15);
        printf("%d", bit);
        ++period;
    } while (lfsr != start_state);
    printf("\nPeriod of output sequence: %d \n", period);
    return 0;
}
```

# What is the Maximum Sequence Length of a single LFSR?

15

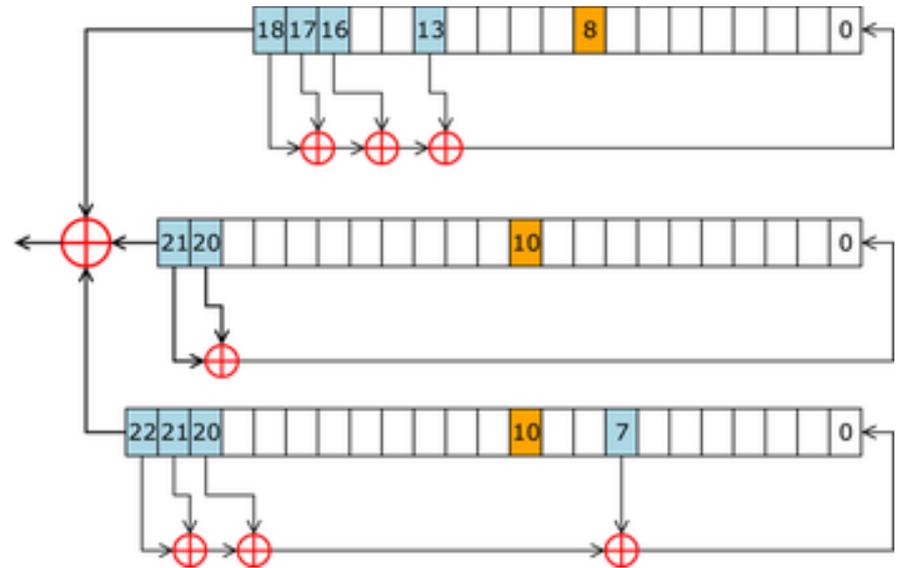
- Consider a single  $n$ -bit LFSR with some feedback function
- Each bit that is shifted out is intrinsically linked to the content of the LFSR
- Each shift operation maps the register content to another (different) pattern, as seen in the example, resulting in another bit shifted out
- An  $n$ -bit LFSR allows for  $2^n$  different register content variations, with each variation pushing out a 0 or a 1
- Therefore, the longest cycle of non-repeating patterns is  $2^n - 1$  iterations, with  $2^n$  the maximum length of the sequence
  - ▣ Think of a 1-bit LFSR ( $n = 1$ ):
    - There are 2 different LFSR contents (“0” or “1”) possible
    - The longest possible patterns are “10” or “01”; both have a length of  $2^n$
    - It just takes one iteration ( $2^{n-1}$ ) to reach all possible register contents ( $1 \rightarrow 0$  or  $0 \rightarrow 1$ )
- However,
  - ▣ poorly designed LFSR may result in cycles that are shorter
  - ▣ the Index of Coincidence problem also applies to LFSR (and in fact to all stream ciphers)

# The Combined LFSR

- A combined LFSR uses multiple LFSR in parallel, and combines their respective outputs to generate a key stream
- They work well on resource-constrained devices too
- Example: A5/1, which was used for GSM voice communication:
  - The Global System for Mobile Communications (GSM) was a mobile phone standard back in the 1990s
  - In GSM, digitised phone conversations are sent as sequences of frames
  - One frame is sent every 4.6 milliseconds and is 228 bits in length
    - Voice samples are collected / digitised over 4.6 milliseconds and send in a block
  - A5/1 is a combined LFSR-based algorithm that is used to produce 228 bits of key stream which is XORed with the frame
  - It is initialised using a 64-bit key

# Example A5/1

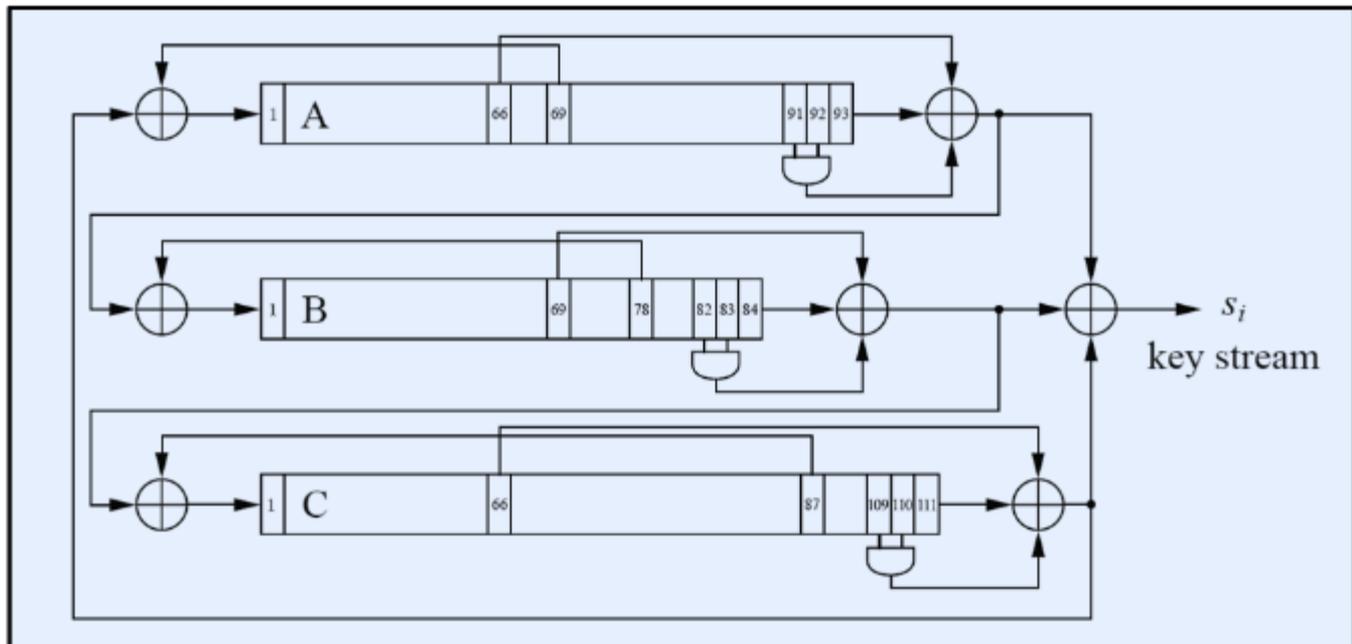
- 3 independent LFSRs:
  - LFSR 1
    - 19 bits
    - LFF: B18 XOR B17 XOR B16 XOR B13
  - LFSR 2:
    - 22 bits
    - LFF: B21 XOR B20
  - LFSR 3:
    - 23 bits
    - LFF: B22 XOR B21 XOR B20 XOR B7
- The output bit is the XORed output of all 3 LFSRs
- A LFSR is only shifted to the left, if their clocking bit (B8, B10, and B10 respectively) matches the output bit; otherwise, there is no shift, and the same output bit value is used again in the next cycle



# Non-Linear Feedback Shift Registers (NLFSR)

18

- NLFSR contain AND gates as well as XOR gates in their feedback function
- Example Trivium: A, B and C are three shift registers with bit lengths of 93, 84 and 111 bits respectively



# Example for a 16-bit NLFSR in C

19

```
#include <stdint.h>
#include <stdio.h>
int main(void)
{
    uint16_t start_state = 0xACE1u; /* Any non-zero start state will work. */
    uint16_t lfsr = start_state;
    uint16_t bit, period = 0;
    do
    { /* FBF: B15 XOR B13 XOR B12 XOR B10 XOR (B2 and B1)*/
        bit = ((lfsr >> 0) ^ (lfsr >> 2) ^ (lfsr >> 3) ^ (lfsr >> 5) ^ ((lfsr >> 13) & (lfsr >> 14))) & 1u;
        lfsr = (lfsr >> 1) | (bit << 15);
        printf("%d", bit)
        ++period;
    } while (lfsr != start_state);
    printf("\nPeriod of output sequence: %d \n", period);
    return 0;
}
```

# Pseudo-Random Number generation: RC4

- Instead of single bits, a generator algorithm can also produce one byte (or one word) at a time
- RC4 is an example for such an algorithm, it returns one pseudorandom byte at a time
- It was designed by Ron Rivest of RSA Security in 1987
- RC4 was initially a trade secret, but in 1994 a description of it was anonymously posted on the Internet
- RC4 consists of a
  - ▣ key-scheduling algorithm (KSA) and a
  - ▣ pseudo-random generation algorithm (PRGA)

# RC4: The Key-Scheduling Algorithm (KSA)

- The KSA requires a key (stored in `key[]`) of length `keylength`
  - ▣ `keylength` is somewhere between 1 and 256
- Using the keyword, a 256-byte long permutation vector `S[]` is generated:

```
for i from 0 to 255
    S[i] := i;
j := 0;
for i from 0 to 255
    j := (j + S[i] + key[i mod keylength])
        mod 256;
    swap(S[i], S[j]);
```

# RC4: The Pseudo-Random Generation Algorithm (PRGA)

- PRGA returns one byte at a time:

```
i := 0;
```

```
j := 0;
```

```
while GeneratingOutput:
```

```
    i := (i + 1) mod 256;
```

```
    j := (j + S[i]) mod 256;
```

```
    swap(S[i], S[j]);
```

```
    output S[(S[i] + S[j]) mod 256];
```

# Security of RC4

- ❑ Obviously not an LFSR-based design, but a more general pseudo-random number generator design
- ❑ Can also be efficiently implemented in software
  - ▣ Very compact algorithm
- ❑ However, it is not deemed safe anymore!

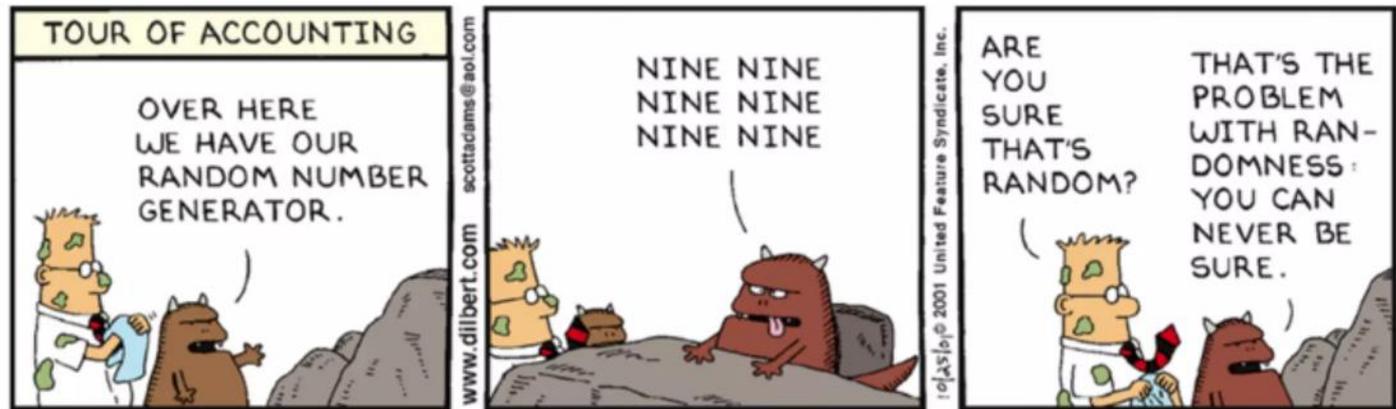
## 3 Security

- 3.1 Roos's biases and key reconstruction from permutation
- 3.2 Biased outputs of the RC4
- 3.3 Fluhrer, Mantin and Shamir attack
- 3.4 Klein's attack
- 3.5 Combinatorial problem
- 3.6 Royal Holloway attack
- 3.7 Bar-mitzvah attack
- 3.8 NOMORE attack

# Background: Pseudorandom Number Generators

24

- Cryptographically strong pseudorandom number generation is essential!



- Pseudorandom number generators (PRNG) are used in a variety of cryptographic and security applications, including
  - ▣ Stream cipher encryption → 802.11 WEP
  - ▣ Encryption keys (both for symmetric and public key algorithms)

# Obvious Requirements for Random Number Generators

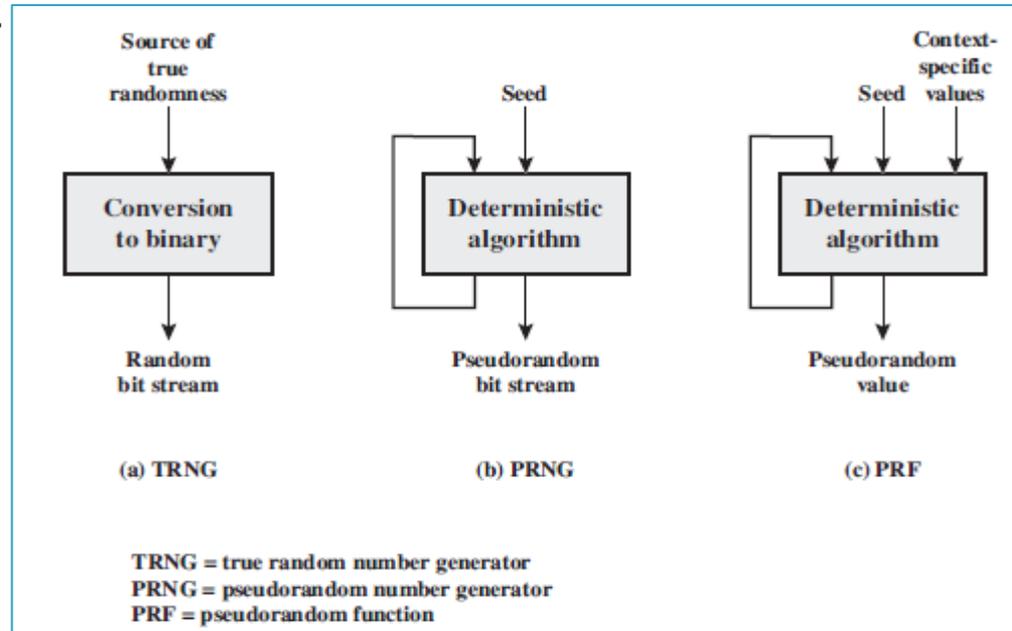
25

- Assume we toss a fair coin or throw a fair dice multiple times. We expect the following from the resulting sequence:
- Randomness, i.e. uniform distribution
  - ▣ The distribution of values in the sequence (e.g. “head or tail”) should be uniform; that is, the frequency of occurrence of possible outputs should be approximately equal
- Unpredictability, i.e. independence
  - ▣ Successive members of the sequence are unpredictable; no subsequence in the sequence can be inferred from the others

# Types of Random Generators

26

- A TRNG takes as input a source that is effectively random
  - ▣ The source is often referred to as an **entropy source**
  - ▣ The entropy source is drawn from the physical environment of the computer, e.g. a combination of keystroke timing patterns, CPU temperature changes and mouse movements
- A PRNG uses just a seed (e.g. LFSR)
- A PRF often also takes in a context-specific value, e.g.
  - ▣ A secure end-to-end communication via TCP/IP may take in the endpoints' IP addresses
- However, PRNG and PRF are based on deterministic algorithms, therefore the “P”



# Formal Requirements for Pseudorandom Generators

27

## □ Randomness

The generated bit stream must “appear” random even though it is deterministic

This can be validated by applying a sequence of tests to the generator, which determine (among others) the following characteristics:

- **Uniformity:** At any point in the generation of a sequence of random or pseudorandom bits, the occurrence of a zero or one is equally likely; The expected number of zeros (or ones) is  $n/2$ , with  $n$  being the sequence length
- **Scalability:** Any test applicable to a sequence can also be applied to sub-sequences extracted at random; if a sequence is random, then any such extracted subsequence should also be random
- **Consistency:** The behavior of a generator must be consistent across many starting values (seeds); it is inadequate to test a PRNG based on the output from a single seed

# Formal Requirements for Pseudorandom Generators

28

## □ Unpredictability

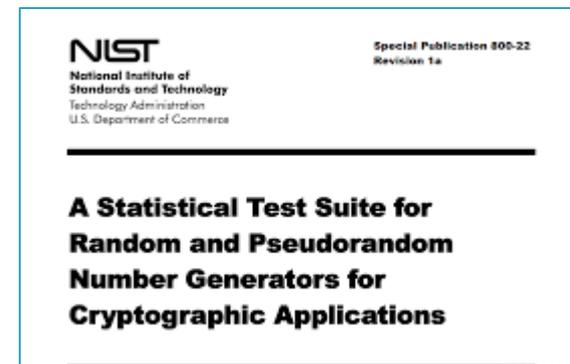
A stream of pseudorandom numbers should exhibit two forms of unpredictability

- ▣ Forward unpredictability: If the seed is unknown, the next output bit in the sequence should be unpredictable in spite of any knowledge of previous bits in the sequence
- ▣ Backward unpredictability: It should not be feasible to determine the seed from knowledge of any generated values; no correlation between a seed and any value generated from that seed should be evident; each element of the sequence should appear to be the outcome of an independent random event whose probability is 0.5

# NIST SP 800-22

29

- The National Institute of Standards and Technology (NIST) published the above report, “*A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*”
- It lists 15 separate tests of randomness and unpredictability
- <https://github.com/terrillmoore/NIST-Statistical-Test-Suite>



CT5191

CRYPTOGRAPHY AND NETWORK SECURITY

HASH FUNCTIONS AND MESSAGE AUTHENTICATION  
CODES

Dr. Michael Schukat



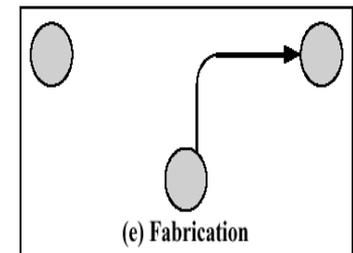
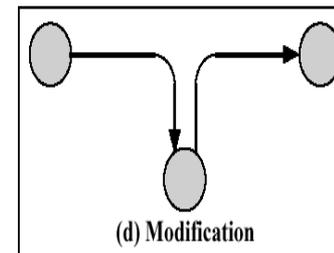
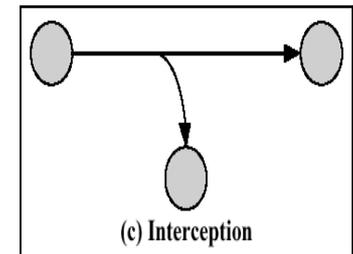
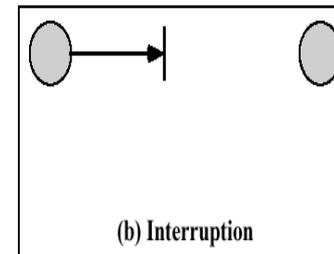
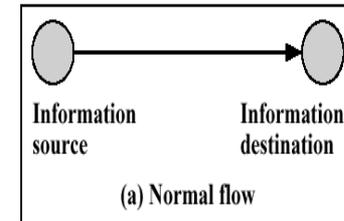
# Lecture Overview

2

- In the previous lectures we have covered block and stream ciphers that provide data confidentiality
- In this slide deck we focus on data integrity, i.e., *“Guarding against improper information modification or destruction, and includes ensuring information non-repudiation and authenticity”*
- Such integrity protection can be provided via
  - ▣ Message authentication codes
  - ▣ Hash functions

# Recap: Types of Security Attacks on Information in Transit

- Integrity checks are particularly important for data in transit
- Here we need to consider the following **active** and **passive** attacks:
  - ▣ **Interception** - of info-traffic flow, attacks confidentiality
  - ▣ **Interruption** - of service, attacks availability
  - ▣ **Modification** - of info, attacks integrity
  - ▣ **Fabrication** - of info, attacks authentication
- In all these scenarios the attacker is a “Man-in-the-Middle” (MitM)



# Recap: Passive Attacks

- Passive attacks are in the nature of eavesdropping or the monitoring of transmissions:
  - ▣ Release of plaintext message content
  - ▣ Traffic analysis of encrypted data communication
    - Allows to analyse patterns of message exchange (sender, receiver, timing) rather than content
- Tools like Wireshark allow for passive attacks

# Recap: Active Attacks

- Active attacks involve the modification or the creation of data in a stream:
  - **Masquerade**
    - **Attacker pretends to be a legitimate sender or receiver of data**
  - **Replay**
    - **Attacker retransmits (encrypted) data which has been previously captured via eavesdropping**
  - **Modification of message content**
    - **Attacker intercepts a message in transit, modifies it and forwards it to the receiver**
  - **Denial of Service (DoS)**
    - **Attacker Inhibits the normal use of communication services**

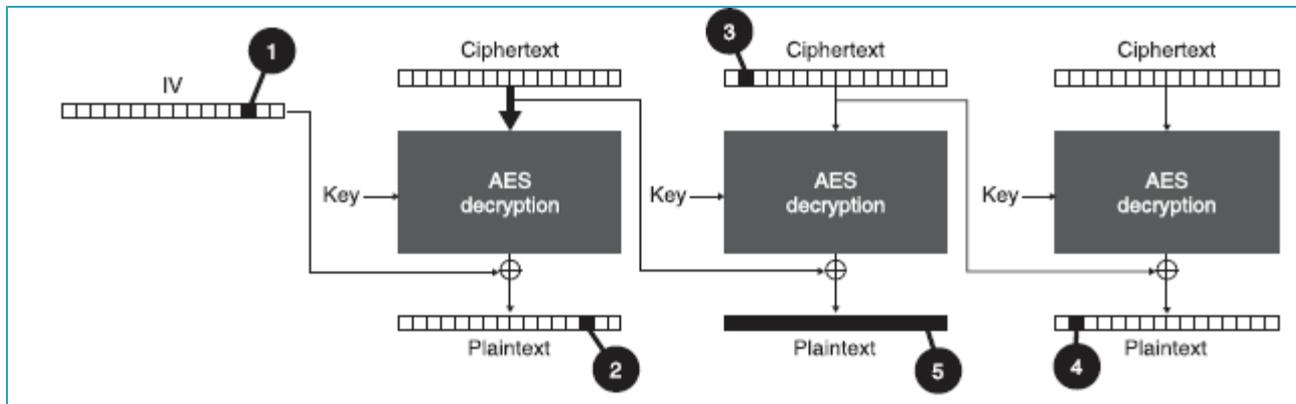
# Attack Scenario

- Your company sends the software patch as email attachment to all the clients
- The patch is encrypted using a secret key, which is pairwise shared with your clients
- However, an attacker can
  - ▣ intercept these emails in transit, changes randomly a few bytes of the encrypted executable and forwards them to their destination, or
  - ▣ forge a similar looking email with some random file attached that claims to be a bug fix
- Your clients replace the executable on their local machines, which of course won't work and bring the entire factory floor to a halt
  - ▣ → financial losses for your clients, huge reputational loss for your company!
- **Therefore, your clients need some mechanism to validate the origin of the email, as well as the integrity of its content**

# Case Study 2: Weakness of Mode

## Block Cipher Modes

- ❑ In CBC, the IV is tagged to an encrypted message as plaintext (thereby allowing the receiver to decrypt the message), a MitM attacker can do changes in transit. Here:
  - ❑ Flipping the  $i^{\text{th}}$  IV-bit (1) flips also the  $i^{\text{th}}$  plaintext bit (2)
  - ❑ Flipping a ciphertext bit (3) will change the entire plaintext block (5), and the corresponding bit of the next plaintext block (4)
- ❑ Other modes show similar weaknesses, i.e. changing one bit in a single block of an encrypted message (in transit) will corrupt the correct decoding of a following blocks
- ❑ **The receiver needs the ability to validate the integrity of the received message (blocks) !**

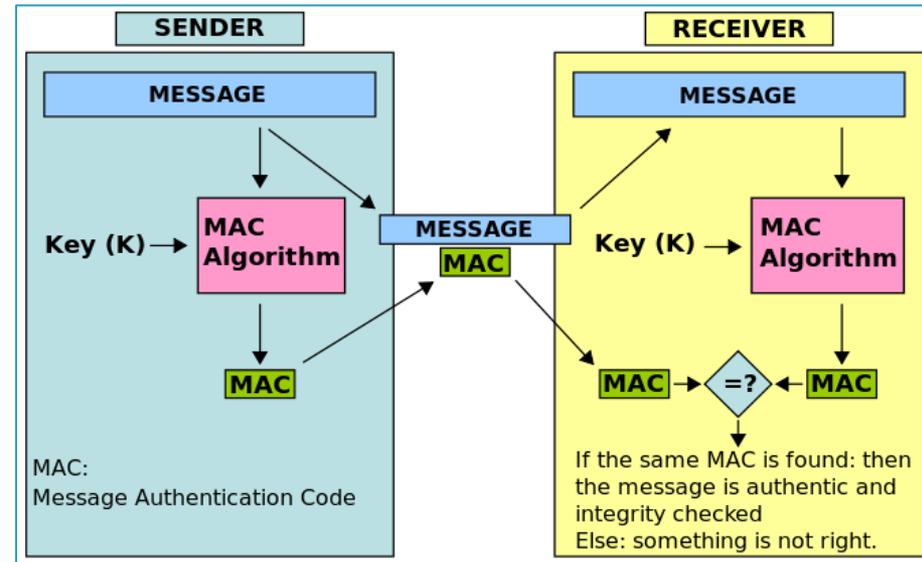


# Message Authentication Code (MAC)

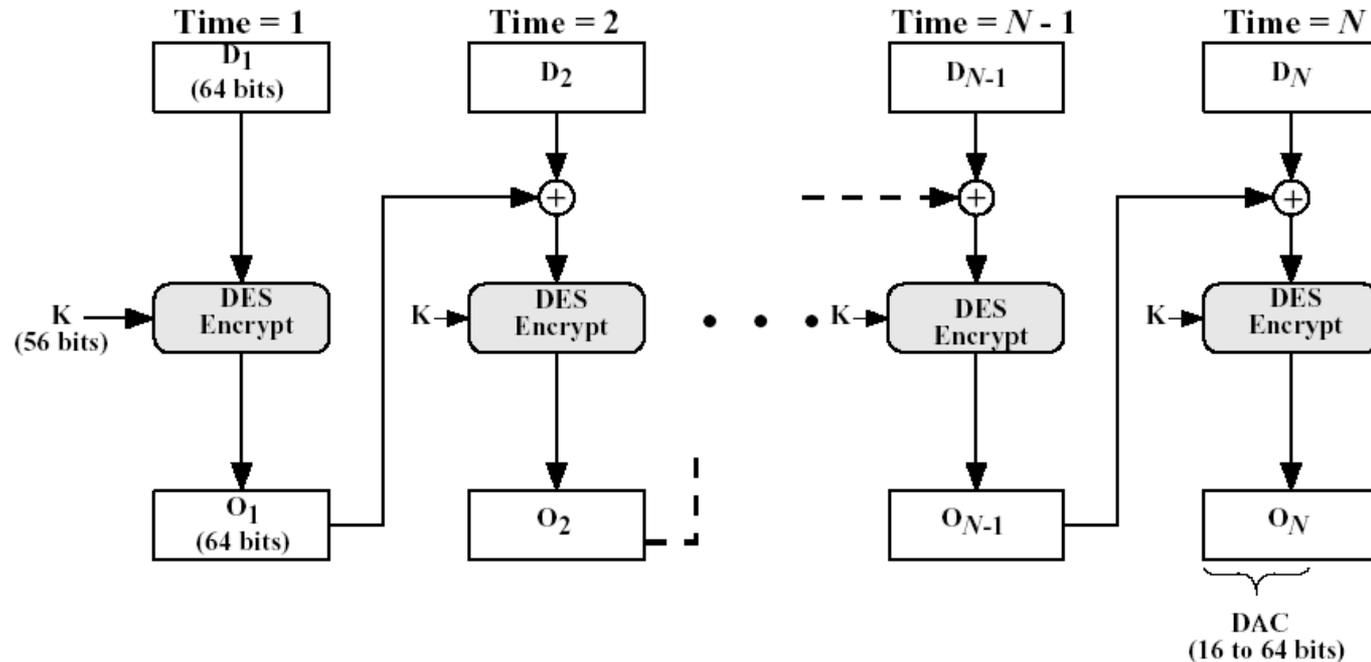
- Message authentication = message integrity [+ source authentication]
- A MAC (also called authentication tag, fingerprint, or cryptographic checksum), is a short piece of information used for authenticating and integrity-checking a message
- A MAC condenses a variable-length message  $M$  using a secret key  $K$  and some algorithm  $C$  to a fixed-sized authenticator:  
 $MAC = C_K(M)$
- After its calculation, the MAC is appended to the message before it is sent
- Note that the message:
  - ▣ can have any length
  - ▣ is not automatically encrypted!

# Typical Use of a MAC (Wikipedia)

- If both MACs are identical, the receiver knows, that
  - ▣ the message was not altered in transit,
  - ▣ the message was sent by the alleged sender, and
  - ▣ if the message includes a sequence number, that the sequence was not altered
- The term **CMAC** is used for a MAC that is calculated using a (block) cipher
- This contrasts to a **HMAC**, where a hash function (later) and a secret key is used



# Typical CMAC Implementation



## □ Generally:

- Any modern block cipher may be used (i.e., it's only DES in the example above)
- Message padding shall apply as seen before
- $MAC = C_K(M)$ , where  $K$  is secret key and  $C$  is a symmetric block cipher (DES above)
- MAC guarantees message integrity AND source authentication
- This construction is also called **Encrypt-then-MAC**

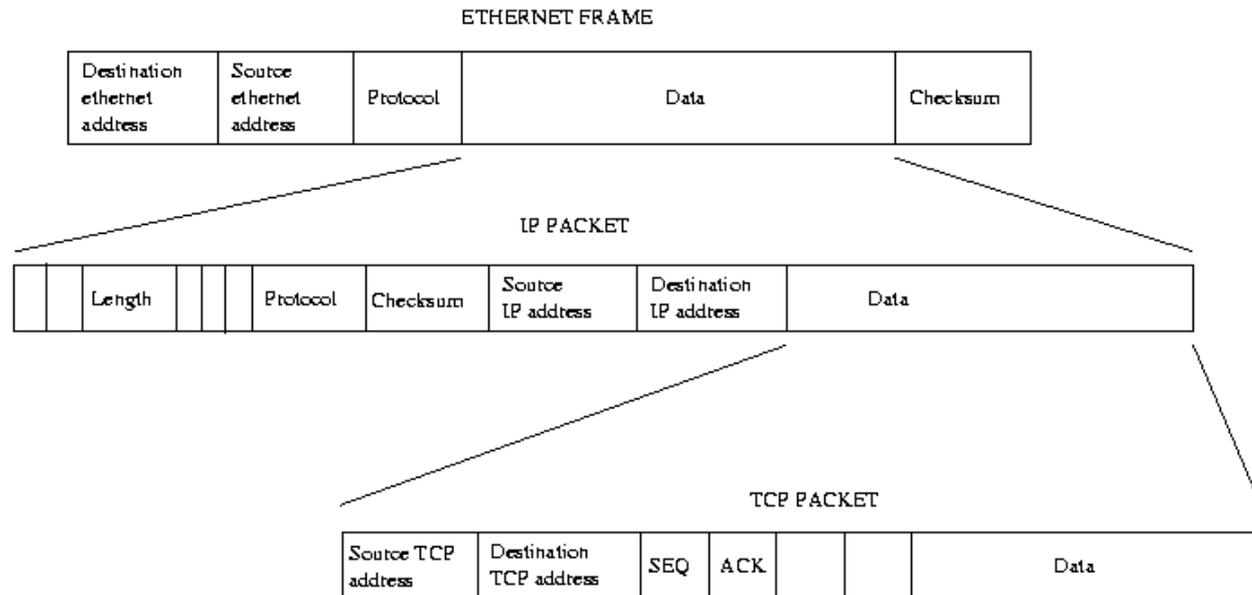
# Message Authentication Benefits

- In summary there are four types of attacks on data in transit, which are addressed by message authentication:
  - ▣ **Masquerade:** insertion of messages into the network from a fraudulent source
  - ▣ **Content modification**
  - ▣ **Sequence modification:** change the order of messages as they arrive
  - ▣ **Timing modification:** delete or repeat messages
- Note that the above may require a unique (i.e. incremented) sequence number in every message
- Therefore, message authentication is concerned with:
  - ▣ Protecting the integrity of a message
  - ▣ Validating identity of originator
  - ▣ Validating sequencing and timeliness
  - ▣ Non-repudiation of origin (dispute resolution)

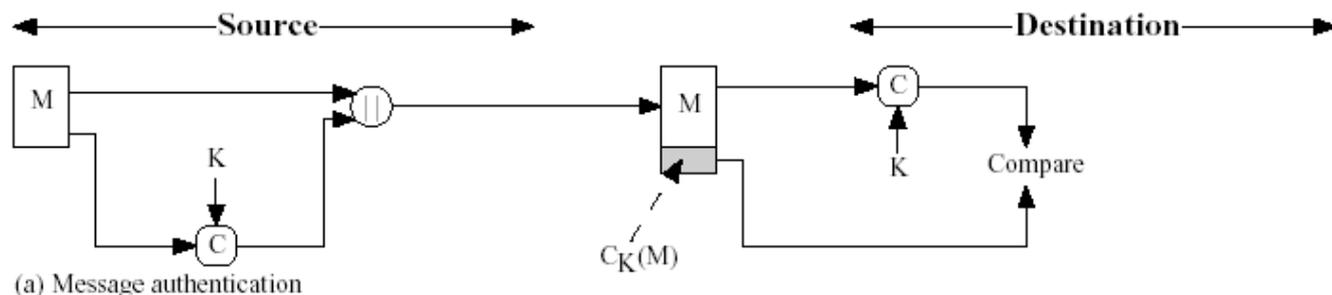
# Example: Authentication of TCP/IP Packets

12

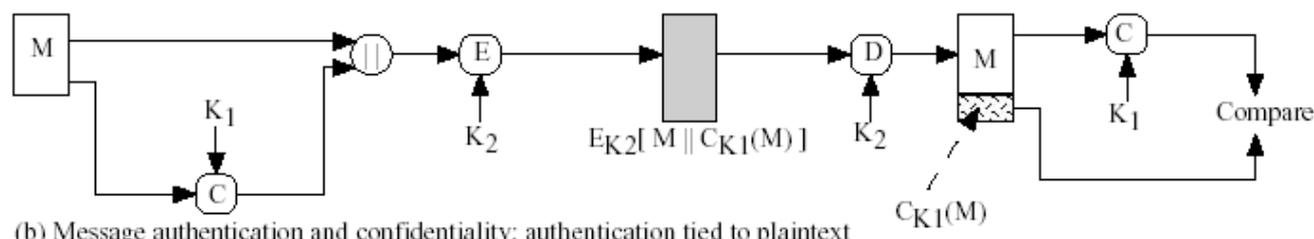
- In TCP/IP data communication, a MAC cannot only cover the payload (i.e., the TCP Data field), but also the TCP header, as well as the non-modifiable fields of the IP header



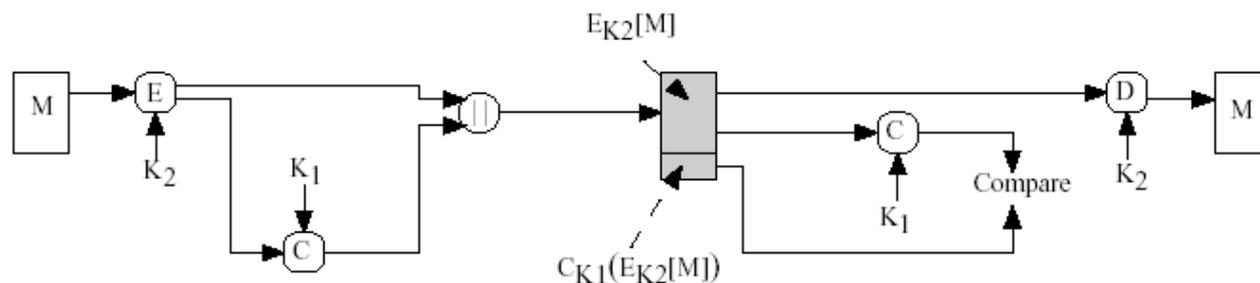
# Basic Use Cases of CMACs



(a) Message authentication



(b) Message authentication and confidentiality; authentication tied to plaintext



(c) Message authentication and confidentiality; authentication tied to ciphertext

- $M$ : Message
- $K$ : Secret key
- $C$ : Block cipher
- $||$ : Concatenation operation

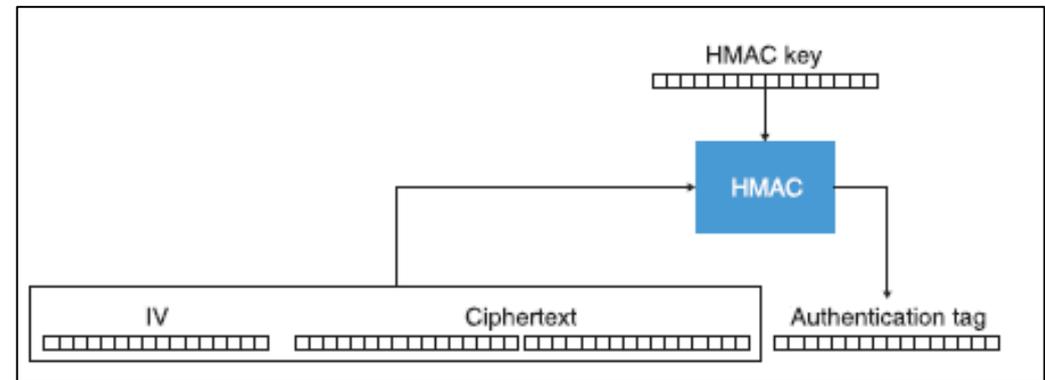
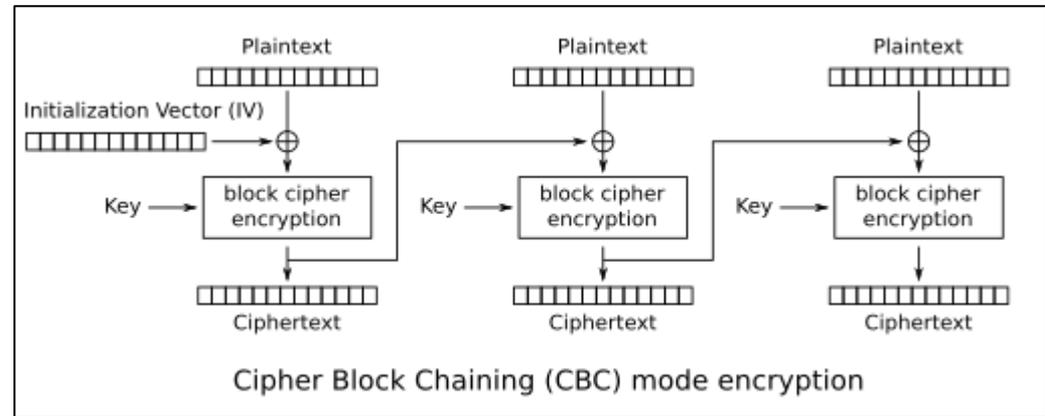
# Case Study CMAC

- Assume you operate a distributed weather station with battery-operated sensors located across Ireland
- You use “public” networks (i.e. Wi-Fi, Internet) to collect data and send it for processing to a central hub in Galway
- Which basic uses of a CMAC as shown in the previous slide would be most appropriate?
  - ▣ In your suggestion consider data privacy concerns and energy budget

# The AES-CBC-HMAC Mode

15

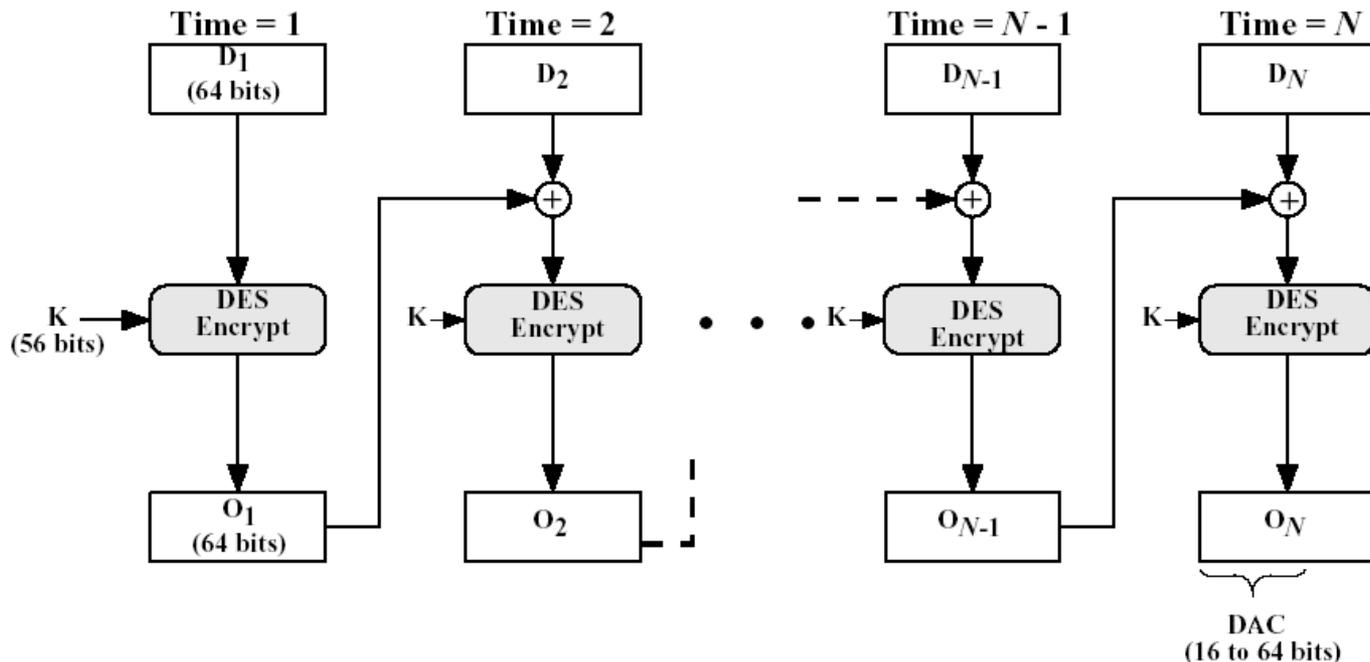
- An example on how to combine authentication with a block cipher mode
- Based on CBC mode (top), but with additional authentication (bottom)
- Here the HMAC takes a single variable length input, i.e. the concatenation of IV + ciphertext
  - ▣ The diagram is misleading as it shows two separate inputs
- How many secret keys would this scheme require?



# Block Cipher Mode of Operation: The Galois / Counter Mode

16

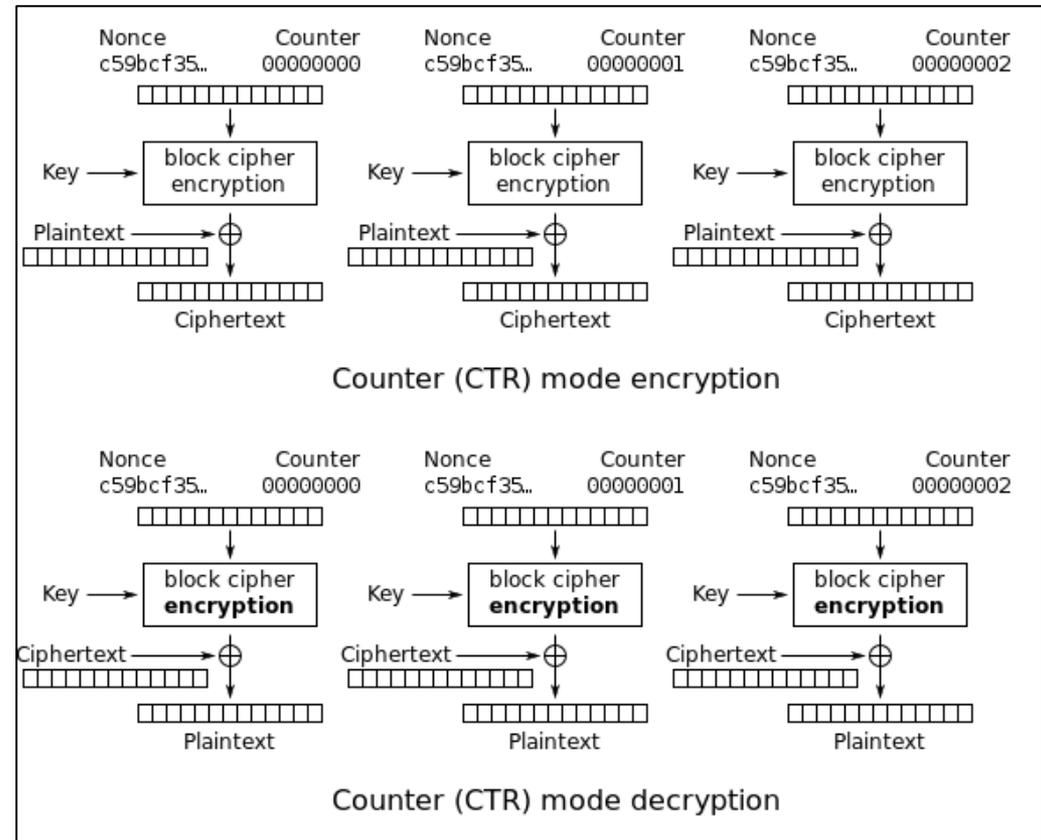
- What are weaknesses of the mode below and the AES-CBC-HMAC Mode (previous slide), i.e.
  - ▣ Can it be parallelised?
  - ▣ Is a 16- to 64-bit DAC sufficient?



# Block Cipher Mode of Operation: The Galois / Counter Mode

17

- Extension of counter mode
- Recall advantages of this mode?



# Block Cipher Mode of Operation: The Galois / Counter Mode

18

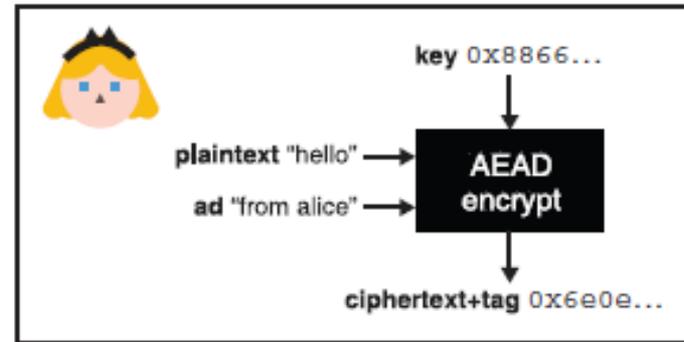
- GCM provides both data authenticity (integrity) and confidentiality
- It belongs to the class of **authenticated encryption with associated data (AEAD)** methods, i.e. it takes as an input
  - ▣ an initialisation vector IV
  - ▣ a **single** secret key  $K$ ,
  - ▣ the plaintext  $P$ , and
  - ▣ some associated data AD
- It encrypts the plaintext (similar to counter mode) using the key to produce ciphertext  $C$ , and computes an authentication tag  $T$  from the ciphertext and the associated data (which remain unencrypted)
- A recipient with knowledge of  $K$ , upon reception of AD,  $C$  and  $T$ , can decrypt the ciphertext to recover the plaintext  $P$  and can check the tag  $T$  to ensure that neither ciphertext nor associated data were tampered with
- GCM uses a block cipher with block size 128 bits (i.e., AES-128), and uses arithmetic in the Galois field  $GF(2^{128})$  to compute the authentication tag
  - ▣ That's modular arithmetic with a modulus of  $2^{128}$

# Features of AEAD

19



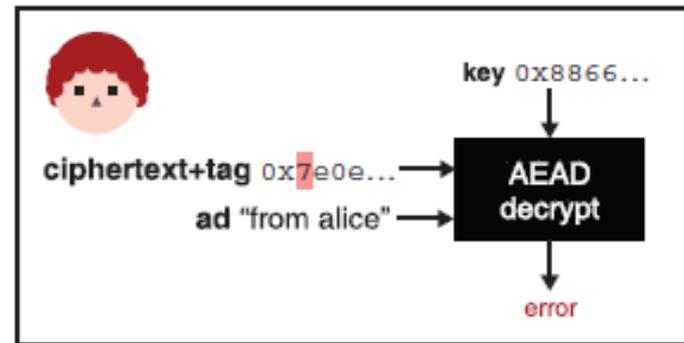
1. Alice and Bob meet in real life to agree on a key.



2. Alice can now use it to encrypt messages with an **AEAD** algorithm and the **symmetric** key. She can also add some optional associated data.



3. The ciphertext and tag are sent to Bob. An observer on the way **intercepts** them and **modifies** the ciphertext.

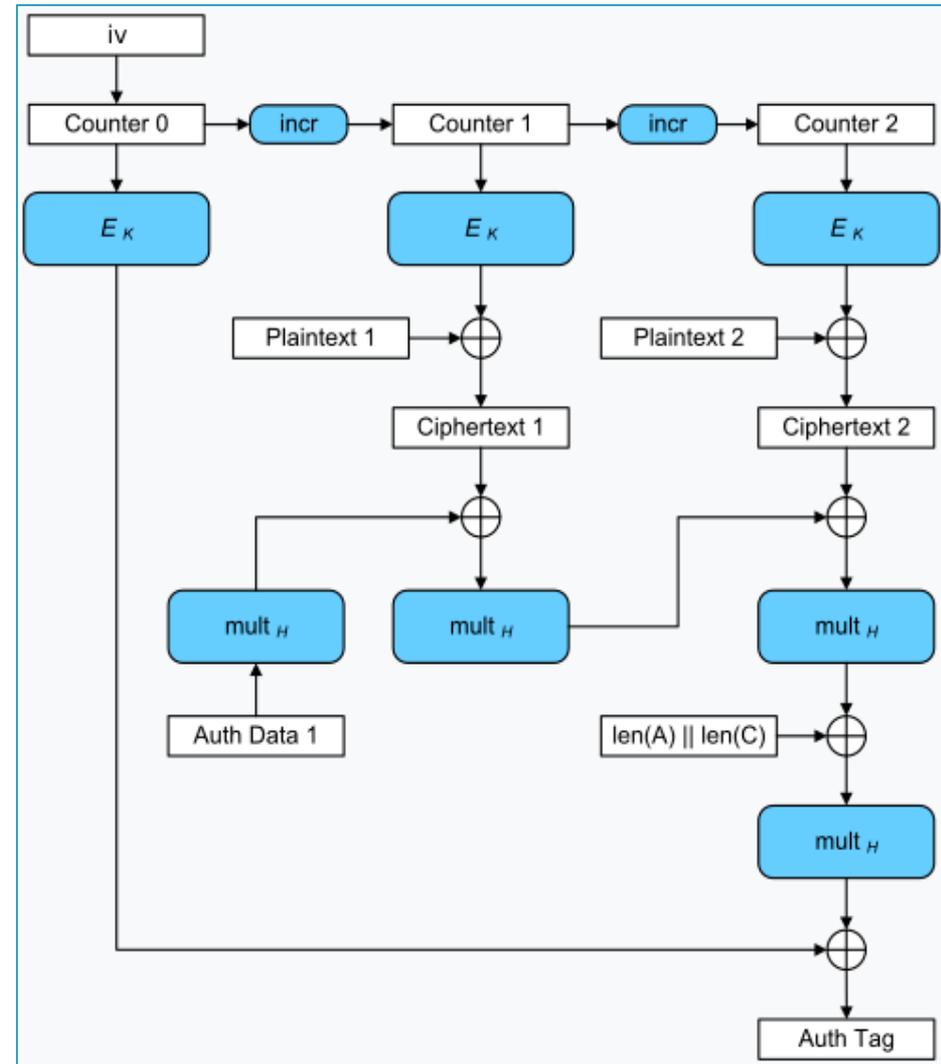


4. Bob uses the AEAD decryption algorithm on the **modified** ciphertext with the same key. The decryption fails.

# Block Cipher Mode of Operation: The Galois / Counter Mode

20

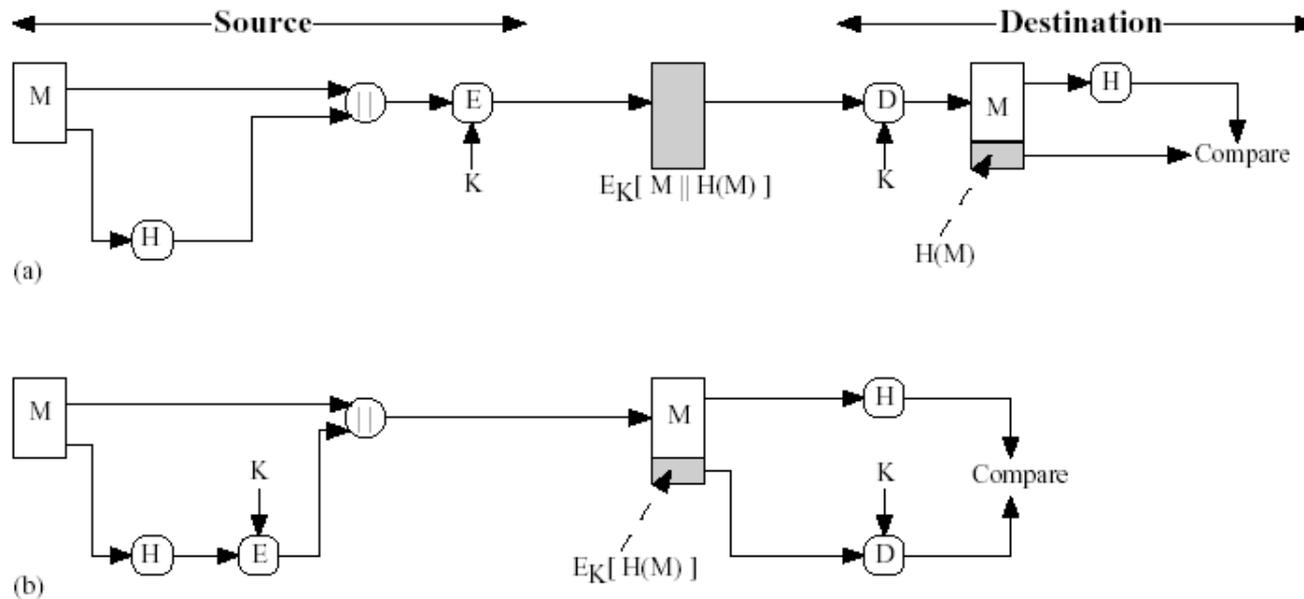
- A 96-bit IV is concatenated with a 32-bit counter (initialised with 0), i.e.  $(IV \ll 32) \parallel C$
- $E_K$  is AES with a 128 – 256 bit key (AES-128, AES-192 or AES-256)
- $\text{mult}_H$  is a hash-function (later) that produces a 128-bit (hash) output
- $\text{Auth\_Data\_1}$  has a variable length (but its hash is 128-bit wide)
- $\text{len}(A)$  and  $\text{len}(C)$  are 64-bit values that are the lengths (in bytes) of  $\text{Auth\_Data\_1}$  and all ciphertext blocks respectively
- $\oplus$  is the bitwise XOR function



# Hash Functions and HMAC

- A hash function produces a fixed size hash code (i.e. hash or fingerprint) based on a variable size input message
  - A hash function
    - does not need a key
    - guarantees the integrity of the message
  - However, since a hash function is public and is not keyed, a hash value may have to be protected (i.e., encrypted)
    - A HMAC (hash-based message authentication code) is a specific type of MAC involving a cryptographic hash function and a secret cryptographic key
    - A HMAC verifies both message integrity and its authenticity
- Modern hash functions calculate 256 - 512-bit hashes

# Basic Uses of HMACs



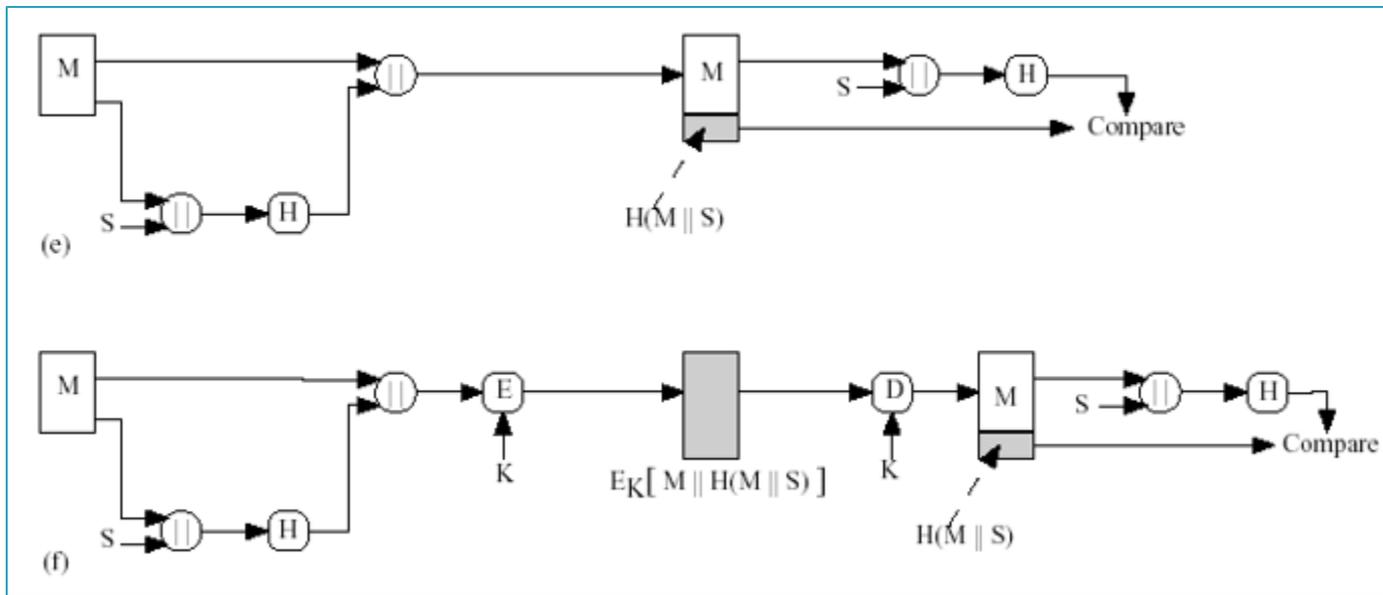
- $M$ : Message
- $H$ : Hash Function
- $E$ : Block Cipher Encryption
- $D$ : Block Cipher Decryption
- $\parallel$ : Concatenation operation

## Note:

- Scenario (a) (and (f)) provide confidentiality and message authentication
- Scenario (b) (and (c)) provide message authentication only

# Basic Uses of HMACs

- In scenarios (e) and (f) a symmetric secret seed  $S$  is used, which is shared between sender and receiver
- $S$  is used to authenticate all messages exchanged between both endpoints
- Scenario (f) also uses a symmetric key  $K$  for confidentiality, which is independent from  $S$



# Case Study HMAC

- Assume you operate a distributed weather station with battery-operated sensors located across Ireland
- You use “public” networks (i.e. Wi-Fi, Internet) to collect data and send it for processing to a central hub in Galway
- Which basic uses of a Hash function as shown in the previous slides would be most **appropriate and efficient?**

# Requirements for a Hash Function $H(x)$

- ❑ **One-way property (also called pre-image resistance):**  
For a given hash function  $H$  and a hash value  $h$  it is infeasible to find  $x$  such that  $H(x) = h$ 
  - ▣ I.e., it is virtually impossible to generate a message given a hash
  - ▣ Such a situation is also called a **hash collision**
- ❑ **Why is the one-way property important?**
  - ▣ See Figure (e): An opponent could intercept  $M \parallel H(M, S)$ , create inputs  $M \parallel X$  (with some random value  $X$ ), until a hash collision is found (i.e.  $S$ )

# Requirements for a Hash Function $H(x)$

- **Weak collision resistance (also called second pre-image resistance):**  
For a given hash function  $H$  and a known input  $x$  it is infeasible to find another input  $y$  with  $y \neq x$  and  $H(x) = H(y)$
  
- **Why is the weak collision resistance important?**
  - See Figure (b): An opponent could
    - calculate  $h(M)$  (as both  $h$  and  $M$  are known)
    - find an alternate message with the same hash code (a hash collision), and
    - send it together with the encrypted (original) hash code to the receiver
  - The receiver would not be able to realise that the original message had been tampered with
    - Think of the previous software patch example

# Requirements for a Hash Function $H(x)$

- **Strong collision resistance (also called collision resistance):**  
It is computational infeasible to find **any** pair of inputs (i.e., messages)  $(x, y)$  with  $H(x) = H(y)$
  
- **Why is the strong collision resistance important?**
  - Again, see Figure (b), but this time the attack vector is different:
    - Rather than intercepting a hashed message in transit, the attacker presents the signing authority a crafted authentic message that has the same hash as a fraudulent message
    - Generating such a crafted message is accommodated by the **Birthday Paradox** discussed earlier

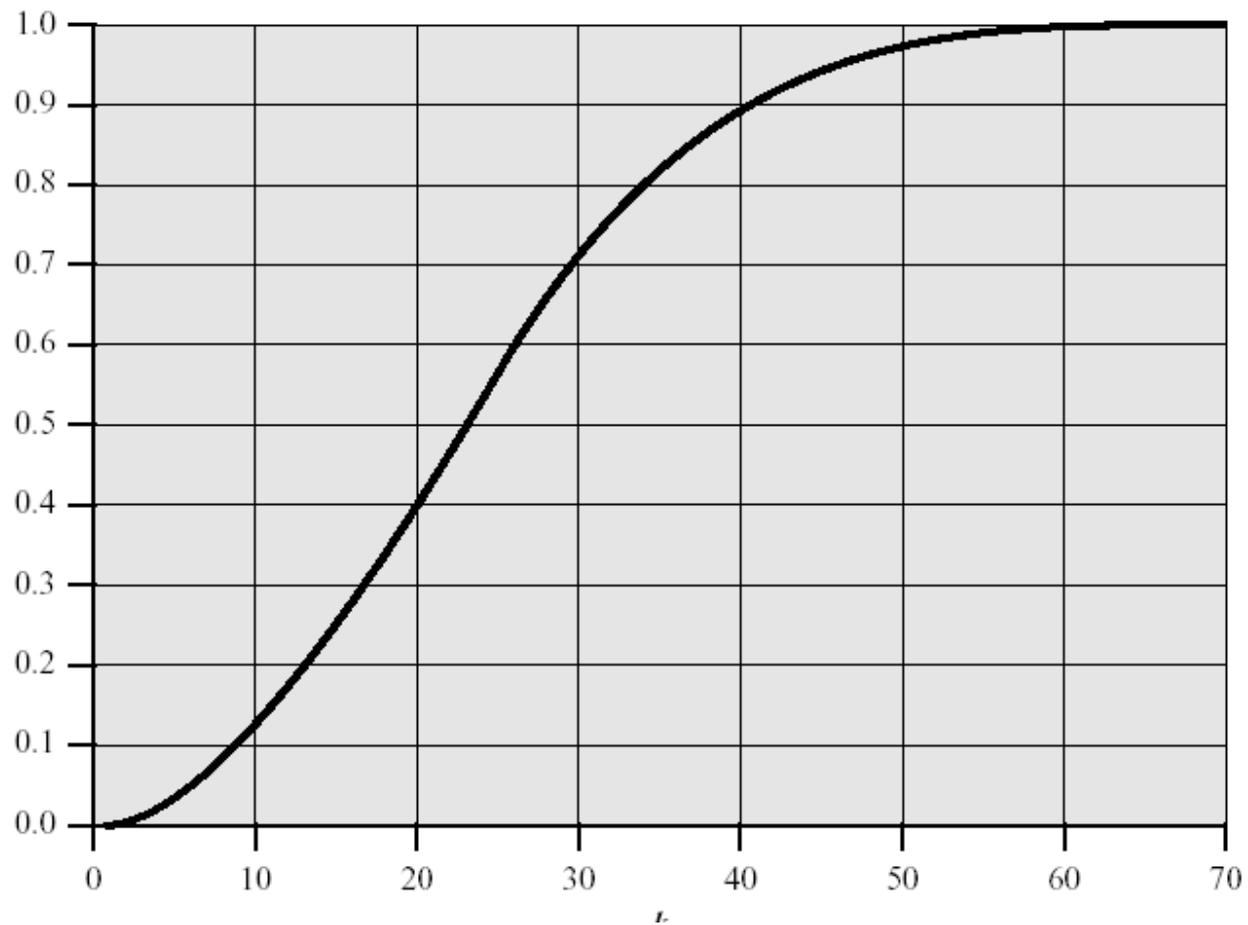
# Birthday Paradox Attack

- Rather than thinking of birthdays, we consider messages and their hashes
- In the BPA the attacker does not intercept a hashed message in transit, but presents the signing authority a crafted authentic message that has the same hash as a fraudulent message (HMAC use case b)
- For a hash value that is  $m$ -bit long, the attacker creates a large number (i.e., in the order of  $2^{0.5m}$ ) of variations of:
  - ▣ correct messages
  - ▣ fraudulent replacement messages
- The birthday paradox will make it more likely to find among both sets a correct message  $M_{\text{nice}}$  that has the same hash as a fraudulent message  $M_{\text{nasty}}$
- $M_{\text{nice}}$  is presented to the signing authority, who
  - ▣ hashes the message
  - ▣ encrypt the hash using the secret key (only known to the signing authority and the receiver)
  - ▣ concatenate message and hash
- Before the message is sent off, the attacker replaces  $M_{\text{nice}}$  with  $M_{\text{nasty}}$
- The receiver gets  $M_{\text{nasty}}$ , but will assume that it was signed (and send) by the signing authority

# Birthday Paradox

- What is the minimum value  $k$  such that the probability is greater than 0.5 that at least 2 people in a group of  $k$  people have the same birthday, assuming that a year has 365 days?
- Intuitively someone would assume that  $k = 365 / 2 = 183$
- **Probability theory shows, that  $k = 23$  is sufficient!**

# Birthday Paradox



# BPA – How to create many Variations of a Message

- The example gives a letter in  $2^{37}$  variations

Dear Anthony,

{This letter is  
I am writing} to introduce {you to  
to you} {Mr.  
--} Alfred {P.  
--}

Barton, the {newly appointed} {new  
chief} {senior} jewellery buyer for {our  
the}

Northern {European  
Europe} {area  
division} . He {will take  
has taken} over {the  
--}

responsibility for {all  
the whole of} our interests in {watches and jewellery  
jewellery and watches}

in the {area  
region} . Please {afford  
give} him {every  
all the} help he {may need  
needs}

to {seek out  
find} the most {modern  
up to date} lines for the {top  
high} end of the

market. He is {empowered  
authorized} to receive on our behalf {samples  
specimens} of the

{latest  
newest} {watch and jewellery  
jewellery and watch} products, {up  
subject} to a {limit  
maximum}

of ten thousand dollars. He will {carry  
hold} a signed copy of this {letter  
document}

as proof of identity. An order with his signature, which is {appended  
attached}

{authorizes  
allows} you to charge the cost to this company at the {above  
head office}

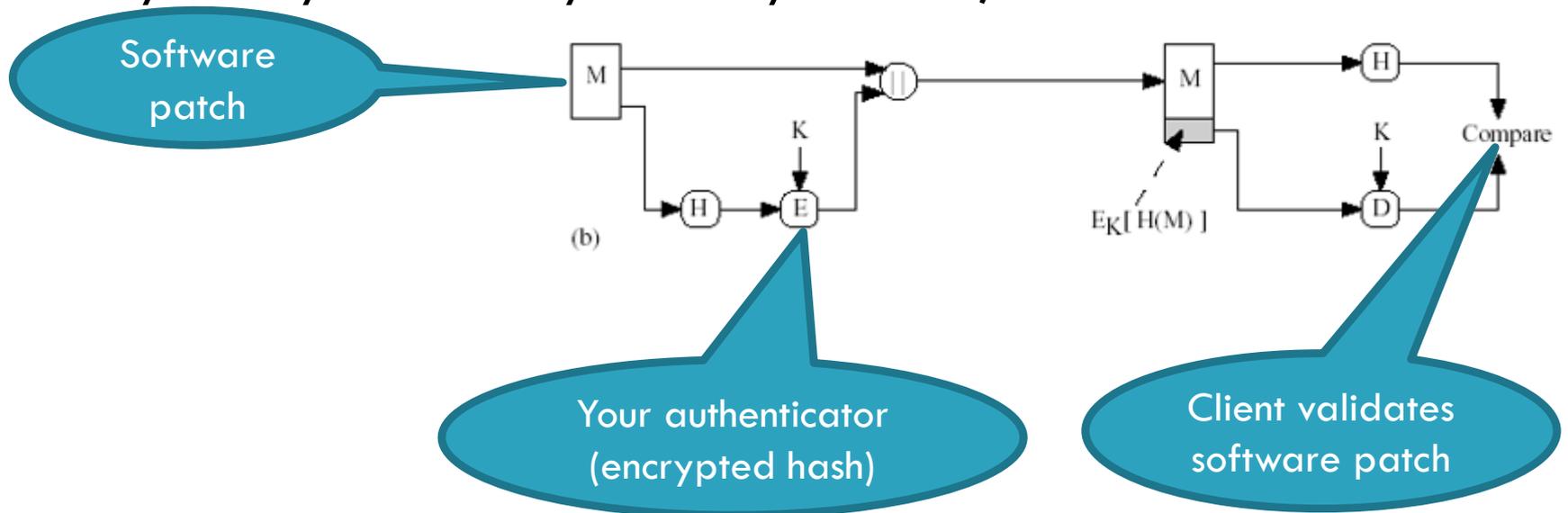
address. We {fully  
--} expect that our {level  
volume} of orders will increase in

the {following  
next} year and {trust  
hope} that the new appointment will {be  
prove}

{advantageous  
an advantage} to both our companies.

# Case Study: Circulating Software using the BPA

- This is a typical insider attack (here conducted by Grumpy George – GG – a disgruntled lead engineer in your team)
- Again, your team develops an urgent software patch, which is hashed
- The 32-bit hash value is encoded using a symmetric key  $K$ , which is shared with your client
- The key is only known to you and your client, but not to GG



# Case Study: Circulating Software via a Birthday Paradox Attack

- GG as the lead engineer creates a large number of binary code versions for
  - ▣ software patches (to be presented to quality team)
  - ▣ malicious software patches (to be circulated)
- How can GG create  $> 2 * 2^{16}$  different source code variations?
  - ▣ GG introduces in both source code files a new constant variable (e.g. long int) that is not otherwise used, e.g.  
...  
`const unsigned long int var = 12; // possible values are 0 ... 264-1`
  - ▣ GG then creates different source codes by systematically incrementing `var`
    - GG is able to create  $2^{64}$  different versions of both programs if needs to be
- GG compiles each of those software versions and calculates their hash
- GG looks for a hash collision, i.e. a software patch and a malicious patch that have the same hash code
- GG present this software patch to quality team, who sign it using key K
- GG replaces the software with the malicious patch before sending it to the client

# Hash Function Execution (Example HAVAL)

- HAVAL creates a 256-bit fingerprint, for example:
  - "The quick brown fox jumps over the lazy **d**og"  
will be translated into the (256 bit) hash  
"b89c551cdfe2e06dbd4cea2be1bc7d557416c58ebb4d07cb  
c94e49f710c55be4"
  - "The quick brown fox jumps over the lazy **c**og"  
will be translated into the hash  
"60983bb8c8f49ad3bea29899b78cd741f4c96e911bbc272e  
5550a4f195a4077e"
- **I.e. very similar inputs result in totally different outputs, there is no correlation between a hash and its original input**

# A naive Hash Function based on XOR

- Consider the XOR function  $\oplus$ :
- The input is broken into  $m$  blocks
- For the resulting hash value  $C$ , each bit  $C_i$  is calculated via

$$C_i = b_{i1} \oplus b_{i2} \oplus b_{i3} \oplus \dots \oplus b_{im}$$

Where

- $m$  = the number of  $n$ -bit blocks and
- $b_{ij}$  is the  $i^{\text{th}}$  bit of the  $j^{\text{th}}$  block

***EX-OR Gate Truth Table***

<i><b>A</b></i>	<i><b>B</b></i>	<i><b>A <math>\oplus</math> B</b></i>
<i><b>0</b></i>	<i><b>0</b></i>	<i><b>0</b></i>
<i><b>0</b></i>	<i><b>1</b></i>	<i><b>1</b></i>
<i><b>1</b></i>	<i><b>0</b></i>	<i><b>1</b></i>
<i><b>1</b></i>	<i><b>1</b></i>	<i><b>0</b></i>

# A naive Hash Function based on XOR

	<b>Bit 1</b>	<b>Bit 2</b>	<b>...</b>	<b>Bit n</b>
<b>Block 1</b>	$b_{11}$	$b_{21}$		$b_{n1}$
<b>Block 2</b>	$b_{12}$	$b_{22}$		$b_{n2}$
<b>...</b>				
<b>Block m</b>	$b_{1m}$	$b_{2m}$		$b_{nm}$
<b>Hash code</b>	$C_1$	$C_2$		$C_n$

# A naive Hash Function based on XOR

- ◆ Consider the ASCII-encoded input “ABC” and a hash function H that calculates an 8-bit hash h:
  - $\text{ASCII}(A) = 65_{10} = 01000001_2$
  - $\text{ASCII}(B) = 66_{10} = 01000010_2$
  - $\text{ASCII}(C) = 67_{10} = 01000011_2$

- ◆ 

	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1
A	0	1	0	0	0	0	0	1
B	0	1	0	0	0	0	1	0
C	0	1	0	0	0	0	1	1
<b>h</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>

$$H(\text{“ABC”}) = h = 64_{10} = \text{“@”}$$

# A naive Hash Function based on XOR

- ◆ Does this algorithm fulfil the requirements of a hash function:
  - One-way property?
  - Weak collision resistance?

	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1
A	0	1	0	0	0	0	0	1
B	0	1	0	0	0	0	1	0
C	0	1	0	0	0	0	1	1
<b>h</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>

$$H(\text{"ABC"}) = 64_{10} = \text{"@"}$$

# Example: 8-bit Hash Function based on XOR

- ◆ Fulfils requirements of hash function?

- One-way property? Certainly not!

- Weak collision resistance?  $H(\text{"ABC"}) = H(\text{"@@@"}) = H(\text{"@@@@@@"}) =$

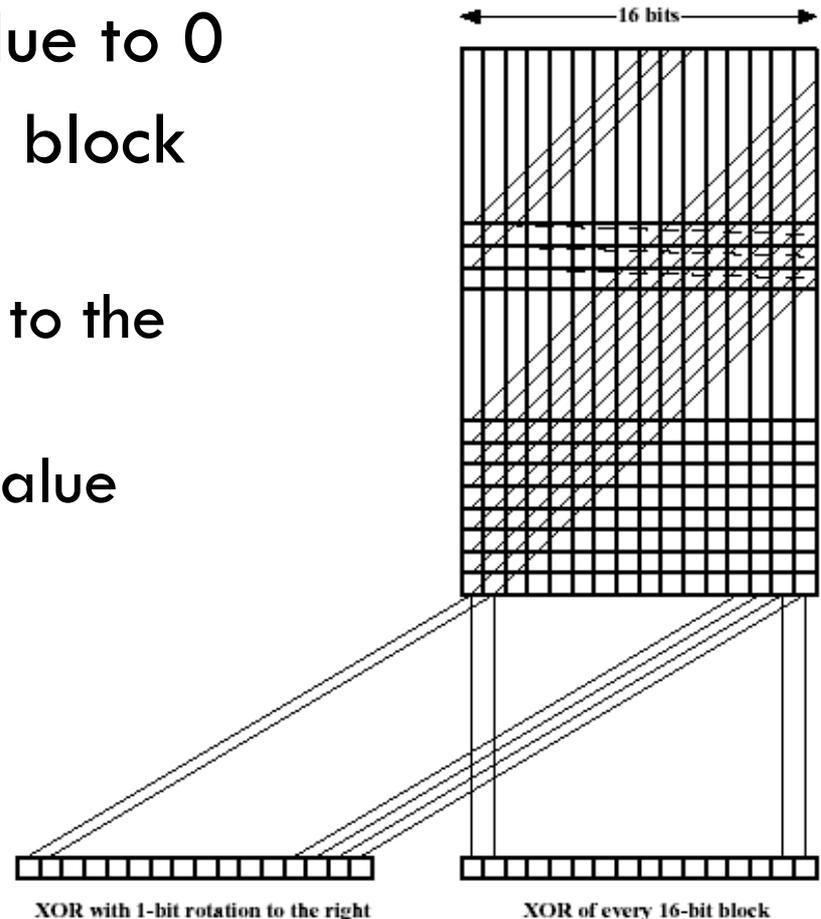
...

	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1
"@"	0	1	0	0	0	0	0	0
"@"	0	1	0	0	0	0	0	0
"@"	0	1	0	0	0	0	0	0
<b>h</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>

$$H(\text{"@@@"}) = 64_{10} = \text{"@"}$$

# A naive Hash Function based on rotating XOR

- Initially set the n-bit hash value to 0
- Process each successive n-bit block a follows:
  - ▣ Rotate the current hash value to the left by one bit
  - ▣ XOR the block into the hash value



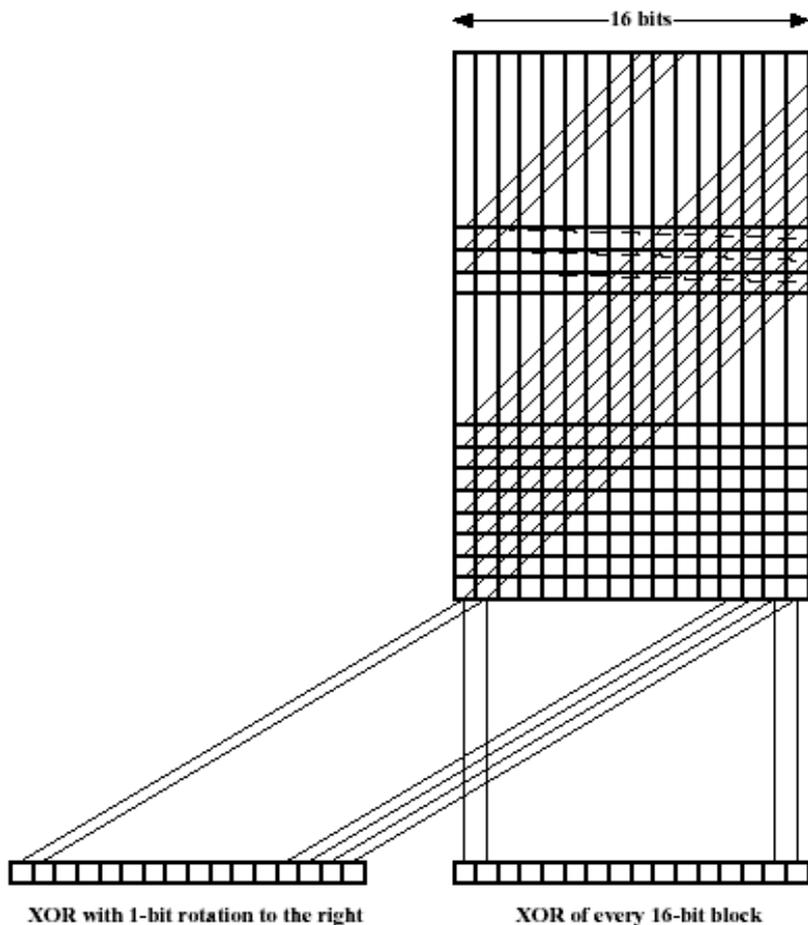
# Example: Simple Hash Function based on Rotating XOR

- Consider “ABCD”
- “AB” = 01000010 01000011<sub>2</sub>
- “CD” = 01000100 01000101<sub>2</sub>
- “CD” left-rotated = 10001000 10001010<sub>2</sub>

	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
	0	1	0	0	0	0	1	1	0	1	0	0	0	0	1	1
	1	0	0	0	1	0	0	0	1	0	0	0	1	0	1	0
h	1	1	0	0	1	0	1	1	1	1	0	0	1	0	0	1

$$h = \text{CBC9}_{16}$$

# Example: Simple Hash Function based on Rotating XOR

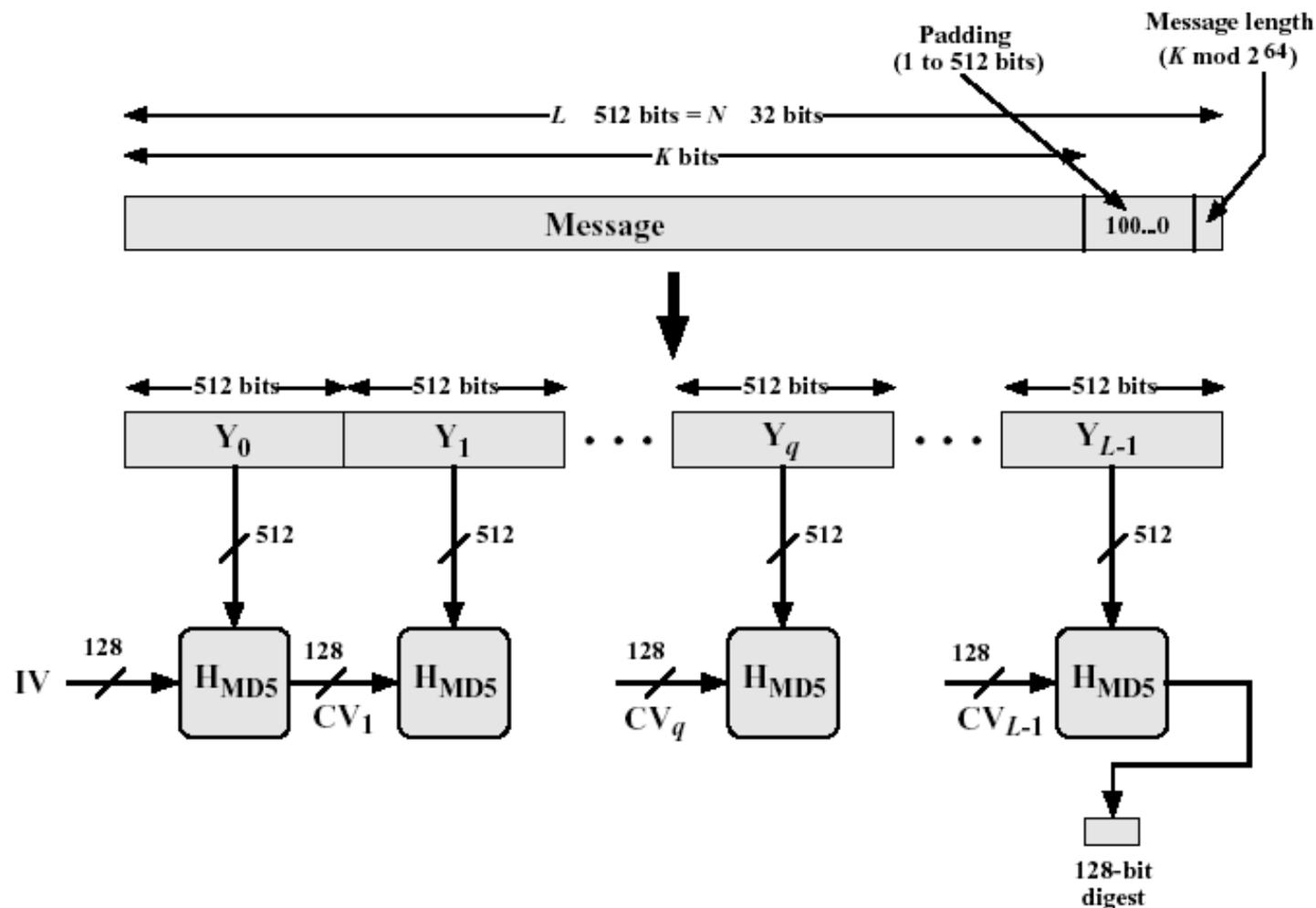


- ❑ Assume a password must be at least 2 ASCII-encoded characters long
- ❑ Fulfills requirements of hash Function?
  - ▣ One-way property?
  - ▣ Weak collision resistance?

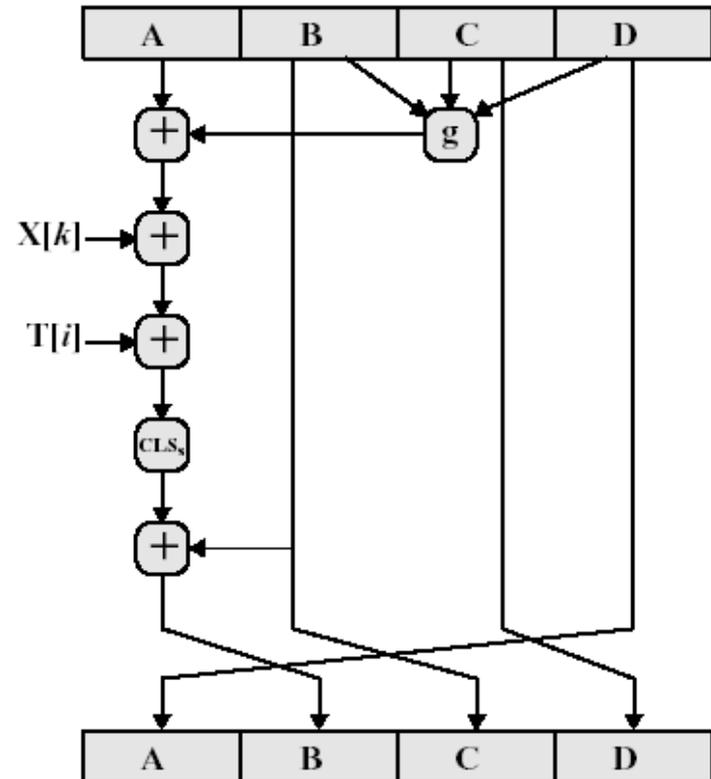
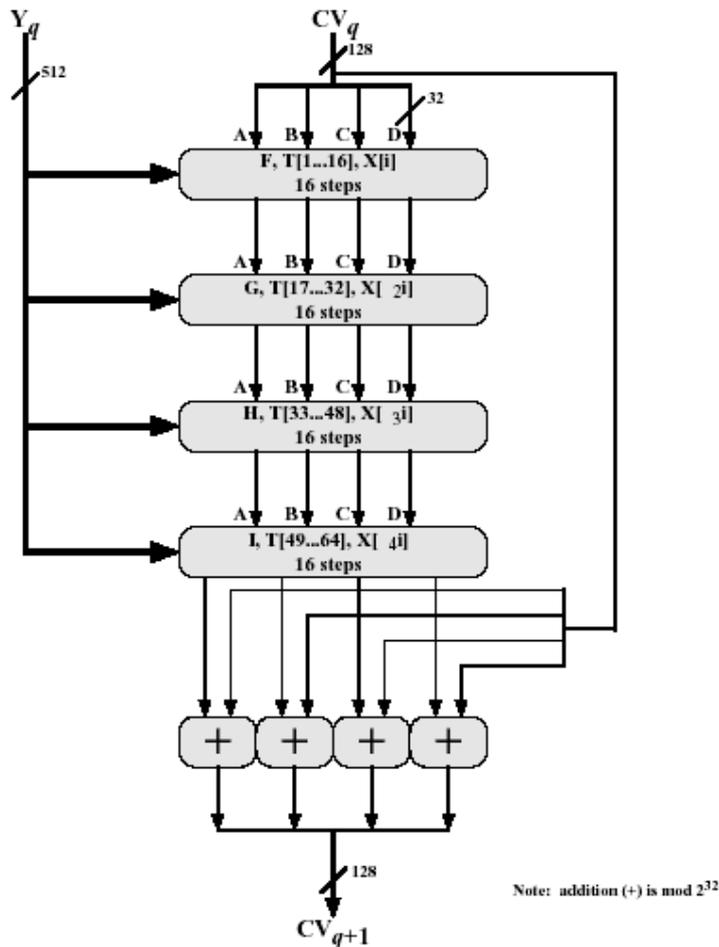
# Examples for Hash Algorithms

- In order to meet the aforementioned requirements, a hash algorithm must
  - ▣ be non-trivial
  - ▣ calculate long hash values
- Popular hash functions include:
  - ▣ MD5:
    - Produces a 128-bit hash value
    - Specified as Internet standards (RFC1321)
    - Still has some popularity, but unsafe for years (broken via collision attacks)
  - ▣ SHA (Secure Hash Algorithm) - X:
    - Family of hash functions, designed by NIST & NSA
    - SHA-3 (released 2015) produces 224-, 256-, 384- and 512-bits hash values
    - Internet standard
  - ▣ RIPEMD-160:
    - Creates a 160-bit hash value
    - Developed in Europe
- See <https://defuse.ca/checksums.htm>

# FYI: MD5-An Overview



# FYI: MD5-Processing of a Single 512 Bit Block (left) and Elementary MD5 Operation



# FYI: MD5-Table T

T [1] = D76AA478	T [17] = F61E2562	T [33] = FFFA3942	T [49] = F4292244
T [2] = E8C7B756	T [18] = C040B340	T [34] = 8771F681	T [50] = 432AFF97
T [3] = 242070DB	T [19] = 265E5A51	T [35] = 699D6122	T [51] = AB9423A7
T [4] = C1BDCEEE	T [20] = E9B6C7AA	T [36] = FDE5380C	T [52] = FC93A039
T [5] = F57C0FAF	T [21] = D62F105D	T [37] = A4BEEA44	T [53] = 655B59C3
T [6] = 4787C62A	T [22] = 02441453	T [38] = 4BDECFA9	T [54] = 8F0CCC92
T [7] = A8304613	T [23] = D8A1E681	T [39] = F6BB4B60	T [55] = FFEFF47D
T [8] = FD469501	T [24] = E7D3FBC8	T [40] = BEBFBC70	T [56] = 85845DD1
T [9] = 698098D8	T [25] = 21E1CDE6	T [41] = 289B7EC6	T [57] = 6FA87E4F
T [10] = 8B44F7AF	T [26] = C33707D6	T [42] = EAA127FA	T [58] = FE2CE6E0
T [11] = FFFF5BB1	T [27] = F4D50D87	T [43] = D4EF3085	T [59] = A3014314
T [12] = 895CD7BE	T [28] = 455A14ED	T [44] = 04881D05	T [60] = 4E0811A1
T [13] = 6B901122	T [29] = A9E3E905	T [45] = D9D4D039	T [61] = F7537E82
T [14] = FD987193	T [30] = FCEFA3F8	T [46] = E6DB99E5	T [62] = BD3AF235
T [15] = A679438E	T [31] = 676F02D9	T [47] = 1FA27CF8	T [63] = 2AD7D2BB
T [16] = 49B40821	T [32] = 8D2A4C8A	T [48] = C4AC5665	T [64] = EB86D391

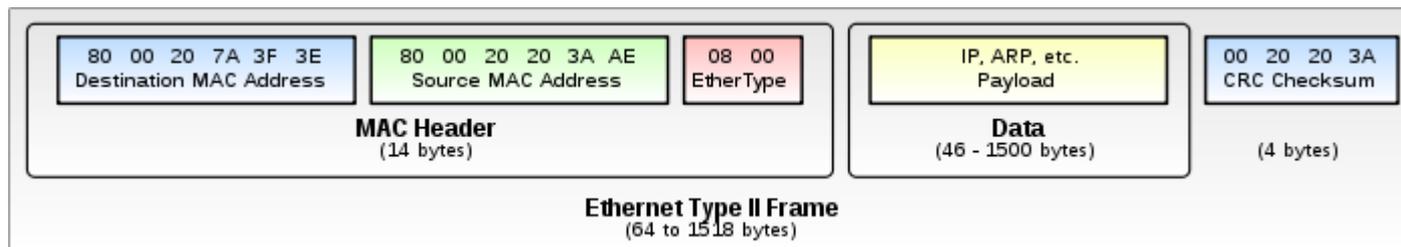
# FYI: MD5-Primitive Functions and their Truth Tables

Round	Primitive function $g$	$g(b, c, d)$
1	$F(b, c, d)$	$(b \text{ AND } c) \text{ OR } (\text{NOT } b \text{ AND } d)$
2	$G(b, c, d)$	$(b \text{ AND } d) \text{ OR } (c \text{ AND NOT } d)$
3	$H(b, c, d)$	$B \text{ EXOR } c \text{ EXOR } d$
4	$I(a, b, c)$	$C \text{ EXOR } (b \text{ or NOT } d)$

b	c	d	F	G	H	I
0	0	0	0	0	0	1
0	0	1	1	0	1	0
0	1	0	0	1	1	0
0	1	1	1	0	0	1
1	0	0	0	0	1	1
1	0	1	0	1	0	1
1	1	0	1	1	0	0
1	1	1	1	1	1	0

# Non-Cryptographic Hash Functions aka Checksums

- ❑ Checksums are designed to detect bit errors of files or data streams, e.g.
  - ❑ Hard disk storage errors
  - ❑ Data transmission errors
- ❑ CRC (Cyclic Redundancy Code) is a well know example
- ❑ Such checksums are too short and vulnerable to brute force attacks, and **are not suitable for cryptographic purposes**



CT437

COMPUTER SECURITY AND FORENSIC COMPUTING

HASH CRACKING AND RAINBOW TABLES

Dr. Michael Schukat



# Lecture Overview

2

- Methods to reverse-engineer hashed passwords
- Rainbow tables
- A recap on SQL injection attacks (based on CT417 content), i.e.
  - ▣ SQL
  - ▣ HTTP get / post Methods and PHP
  - ▣ SQL injection attacks
  - ▣ SQL injection attack mitigation strategiescan be found at the end of this slide deck

# Lecture Motivation

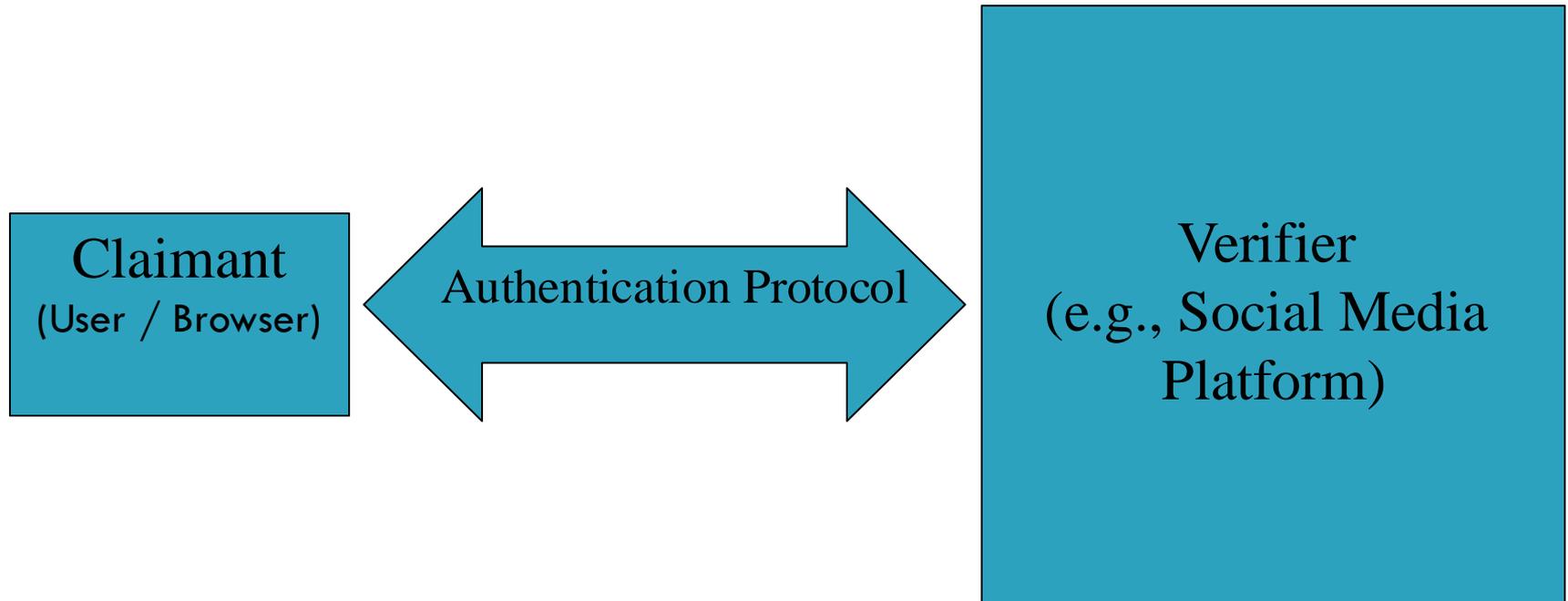
3

- One-way property, weak and strong collision resistance are fundamental properties of a hash function
- These come also into play when we consider common password storage methods ...
- ... and approaches to undermine such methods
- Such approaches are summarised in this slide deck

# What is a Password?

- A memorised secret used to confirm the identity of a user
  - ▣ Typically, an arbitrary string of characters including letters, digits, or other symbols
  - ▣ A purely numeric secret is called a personal identification number (PIN)
- The secret is memorised by a party called the **claimant** while the party verifying the identity of the claimant is called the **verifier**
- Claimant and verifier communicate via an **authentication protocol**

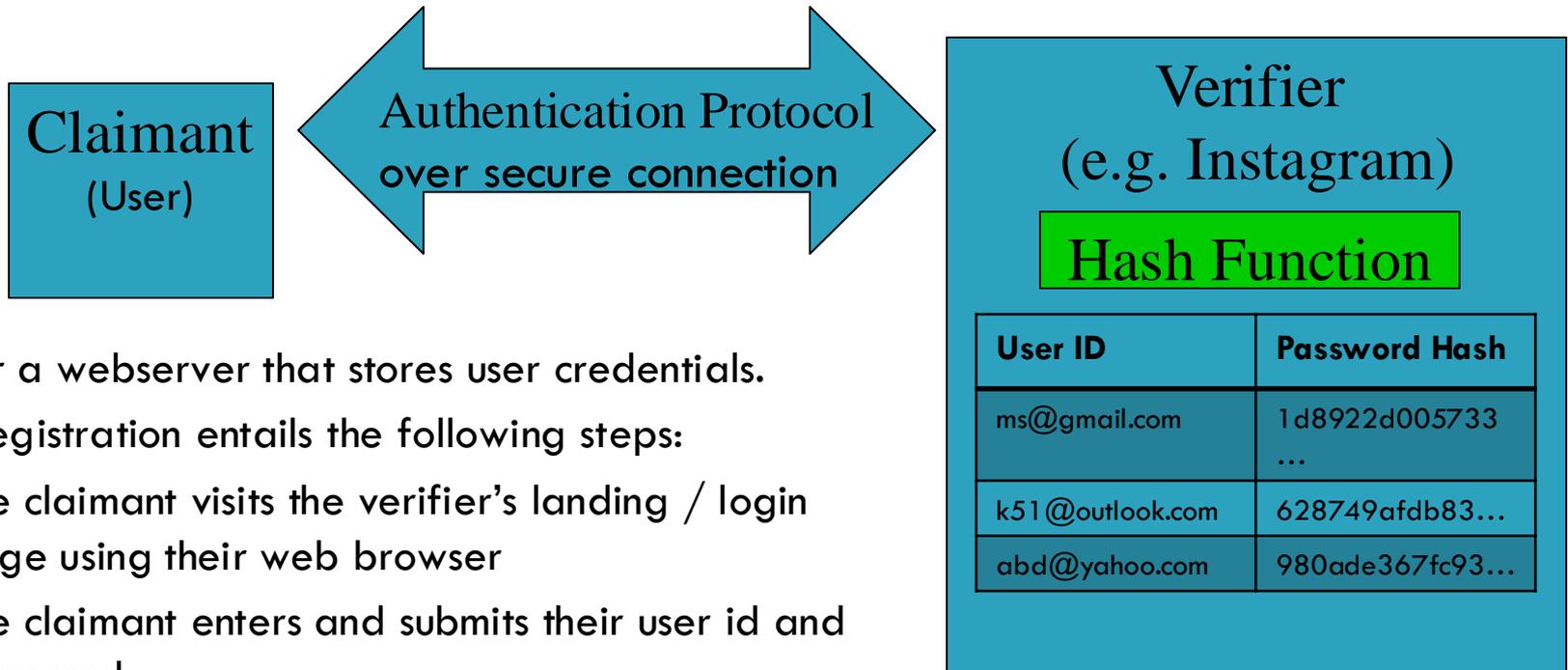
# Claimant and Verifier



# Storing User Passwords

- User passwords at rest (e.g., in database tables) are hashed instead of being stored in plaintext
- Idea:
  - ▣ “KenSentMe!” → “7b24afc8bc80e548d66c4e7ff72171c5”
    - Note: This token is in hex format, it is 128 bit long (32 x 4 bits)
  - ▣ An attacker cannot algorithmically reverse-engineer a hash function to recover the original password
    - Recall hash function properties
  - ▣ The verifier does not have a plaintext copy of the password either

# Why is this Form of Password Hash Management problematic?

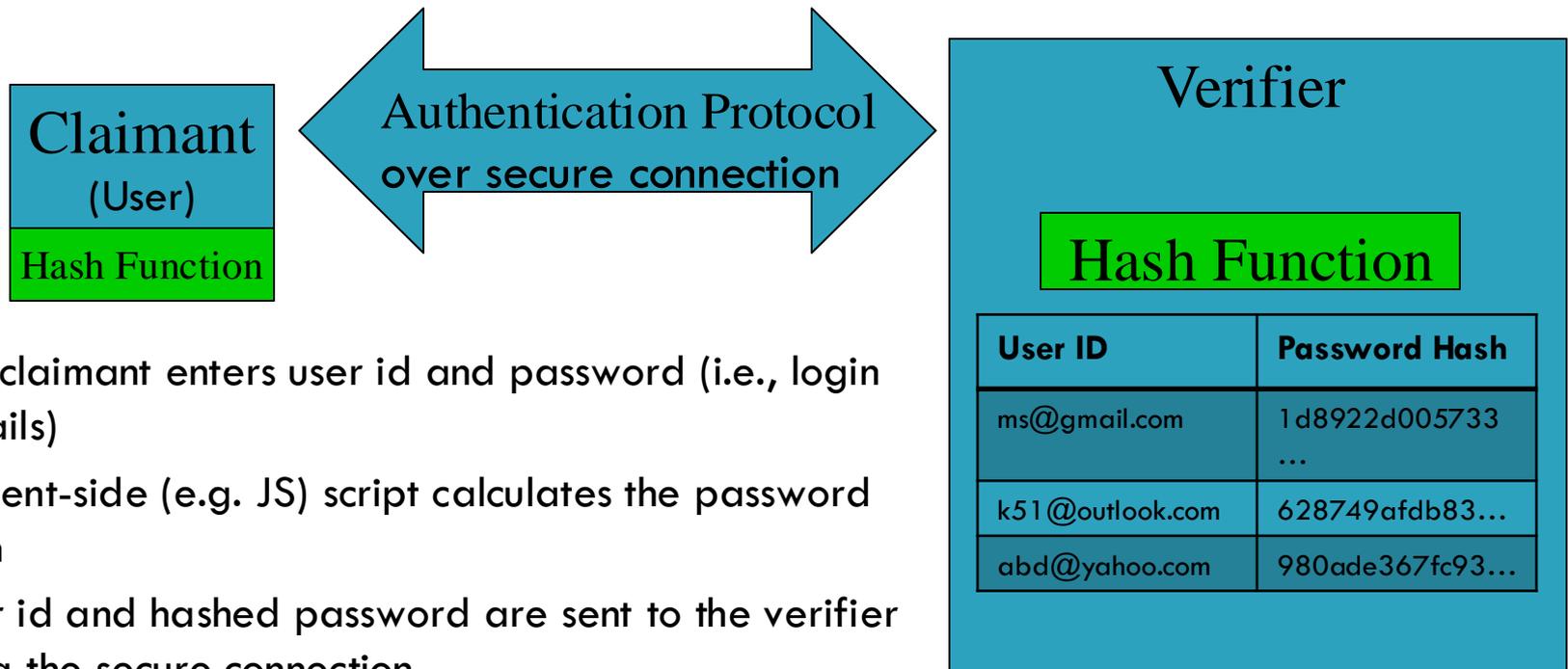


Consider a webserver that stores user credentials.

A user registration entails the following steps:

1. The claimant visits the verifier's landing / login page using their web browser
2. The claimant enters and submits their user id and password
3. Both are sent to the verifier over the secure connection
4. The verifier calculates the hash, and stores it together with the user name in the DB table

# Server-Side Password Storage



1. The claimant enters user id and password (i.e., login details)
2. A client-side (e.g. JS) script calculates the password hash
3. User id and hashed password are sent to the verifier using the secure connection
4. The verifier checks if the transmitted user id and hashed password against the stored values in the table
5. The verifier notifies the claimant via the authentication protocol if the authentication was successful

# Dictionary-Based Brute-Force Search

- ❑ Assume an attacker retrieves an entire DB table containing user IDs and hashed passwords
- ❑ Hash functions are one-way functions, so hash values cannot be transformed back to the original input
- ❑ However, assuming that a user picks a common word or phrase, or a known password as their own password, a simple dictionary search can be used to systematically identify a match for a given hash value
  - ▣ Here the underlying hash function must be known
- ❑ Such dictionaries are based on large word, phrase or password collections
- ❑ 😊 :
  - ▣ Straight forward process
  - ▣ Large dictionaries are readily available (next slide)
- ❑ ☹️ :
  - ▣ Significant computational effort to find match
  - ▣ No guaranteed result

# CrackStation's Password Cracking Dictionary

□ <https://crackstation.net/crackstation-wordlist-password-cracking-dictionary.htm>

## CrackStation's Password Cracking Dictionary

---

I am releasing CrackStation's main password cracking dictionary (1,493,677,782 words, 15GB) for download.

### What's in the list?

---

The list contains every wordlist, dictionary, and password database leak that I could find on the internet (and I spent a LOT of time looking). It also contains every word in the Wikipedia databases (pages-articles, retrieved 2010, all languages) as well as lots of books from [Project Gutenberg](#). It also includes the passwords from some low-profile database breaches that were being sold in the underground years ago.

The format of the list is a standard text file sorted in non-case-sensitive alphabetical order. Lines are separated with a newline "\n" character.

You can test the list without downloading it by giving SHA256 hashes to the [free hash cracker](#). Here's a [tool for computing hashes easily](#). Here are the results of cracking [LinkedIn's](#) and [eHarmony's](#) password hash leaks with the list.

The list is responsible for cracking about 30% of all hashes given to CrackStation's free hash cracker, but that figure should be taken with a grain of salt because some people try hashes of really weak passwords just to test the service, and others try to crack their hashes with other online hash crackers before finding CrackStation. Using the list, we were able to crack 49.98% of one customer's set of 373,000 human password hashes to motivate their move to a better salting scheme.

### Download

---

**Note:** To download the torrents, you will need a torrent client like Transmission (for Linux and Mac), or uTorrent for Windows.

### Torrent (Fast)

GZIP-compressed (level 9). 4.2 GiB compressed. 15 GiB uncompressed.

### HTTP Mirror (Slow)

### Checksums (crackstation.txt.gz)

MD5: 4748a72706ff934a17662446862ca4f8  
SHA1: efa3f5ecbfba03df523418a70871ec59757b6d3f  
SHA256: a6dc17d27d0a34f57c989741acdd485b8aee45a6e9796daf8c9435370dc61612

# Example

- Assume a hash code and the underlying hash function are known
- The dictionary contains  $10^{10}$  entries
- A single laptop / PC can compute  $10^5$  hash values per second
- It takes  $10^5$  seconds ( $\sim 29$  hours) to search the entire dictionary for a match
- This process can be vastly improved by using pre-processed lookup tables

# Lookup Table-Based Attacks

- For a given hash function and dictionary
  - ▣ Calculate the hash values for all dictionary entries
  - ▣ Insert both values to a table (i.e. one line per entry)
  - ▣ Sort table (e.g. in ascending order of hash values)
    - Also called **lookup table**
  - ▣ Store the table
- Example table (assuming 44-bit hash values):

Hash value	Password
0x00000000354	gangster
0x00000001003	Bluemoon
0x00000001032	Z0om!
...	...

# Lookup Table-Based Attacks

- A matching password for a given hash value can be recovered by systematically searching the look-up table via a binary search
- 😊 :
  - ▣ Such a table can be generated offline
  - ▣ The search process itself is fast ( $\sim \log_2(\# \text{ of entries})$ ) using binary search
    - A table containing  $1.8 \times 10^{19}$  entry would require just 64 guesses to find (or not) the correct password for a given hash value
- 😞 :
  - ▣ Huge table, with no guaranteed result
  - ▣ Different table required for every hash function

# Lookup Table-Based Attacks: Example

- Assume a hash function that generates 16-byte (128 bit) hash values
- We calculate a lookup table for all possible 6-character long passwords composed of 64 possible characters A-Z, a-z, 0-9, "." and "/"
- A table would consist of  $64^6$  (= 68,719,476,736) entries, with every entry consisting of a 6-byte password and a 16 bytes hash
- **Total size of table ~ 1.4 Terabyte**
- However, there are online services available that host pre-computed look-up tables for password attacks (see next slide)

# Crackstation's free Password Hash Cracker

□ <https://crackstation.net/>

Free Password Hash Cracker

Enter up to 20 non-salted hashes, one per line:

d9295ddb9fd599a8c8849d14d0186ea0b6d998a4e70335bd8b712831b74fa8

I'm not a robot   
reCAPTCHA  
Privacy - Terms

Crack Hashes

**Supports:** LM, NTLM, md2, md4, md5, md5(md5\_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1 sha1\_bin), QubesV3.1BackupDefaults

Hash	Type	Result
d9295ddb9fd599a8c8849d14d0186ea0b6d998a4e70335bd8b712831b74fa8	sha256	Craughwe11

**Color Codes:** Green Exact match, Yellow Partial match, Red Not found.

## [Download CrackStation's Wordlist](#)

### How CrackStation Works

CrackStation uses massive pre-computed lookup tables to crack password hashes. These tables store a mapping between the hash of a password, and the correct password for that hash. The hash values are indexed so that it is possible to quickly search the database for a given hash. If the hash is present in the database, the password can be recovered in a fraction of a second. This only works for "unsalted" hashes. For information on password hashing systems that are not vulnerable to pre-computed lookup tables, see our [hashing security page](#).

Crackstation's lookup tables were created by extracting every word from the Wikipedia databases and adding with every password list we could find. We also applied intelligent word mangling (brute force hybrid) to our wordlists to make them much more effective. For MD5 and SHA1 hashes, we have a 190GB, 15-billion-entry lookup table, and for other hashes, we have a 19GB 1.5-billion-entry lookup table.

You can download CrackStation's dictionaries [here](#), and the lookup table implementation (PHP and C) is available [here](#).

# In-Class Activity: Password Recovery

- 5 minutes only, work alone or in a group
- What to do:
  - ▣ Pick a password and calculate its MD5 or SHA1 hash using <https://defuse.ca/checksums.htm>
  - ▣ Copy and paste the hash value into <https://crackstation.net/> to see if it is can be recovered
  - ▣ Repeat the above and keep a list of all passwords
    - that **can** be cracked
    - that **cannot** be cracked

# Rainbow Tables

- Look-up tables are huge and take up a lot of hard disk space
- Rainbow tables in contrast provide an efficient way to represent large numbers of hash values
- They require more processing time and less storage to find a match compared to a simple lookup table
- Rainbow tables are a practical example of a **space–time trade-off**
- They are based on pre-computed hash chains

# Pre-Computed Hash Chains

- Such chains contain long sequences of password candidates (green strings below) and hash values (black strings below)
- They are based on using a hash function “→” and a reduction function “→”, e.g.,  
aaaaaa → 173bdfede2ee3ab3 → jdklkuo → 9fdde3a0027fbb36 → ... → k3rtol
  - In this example we only consider passwords (green) that are 6 characters long, which are converted into 64-bit hash values
  - Each chain starts with a different password
  - Each chain has a fixed length, e.g. 100,000 passwords and their hashes
  - Here “→” converts the 64-bit hash value into an arbitrary 6-byte long string again, i.e. it's not an inverted hash function!
- We only store the first and the last value (starting point and end point), i.e. “aaaaaa” and “k3rtol”

# Example for a simple Reduction Function

19

```
private static String reductionFunction(long val) {                               // Hash value is just a long integer
    String car, out;                                                            // The method returns an alphanumeric string
    int i;
    char dat;

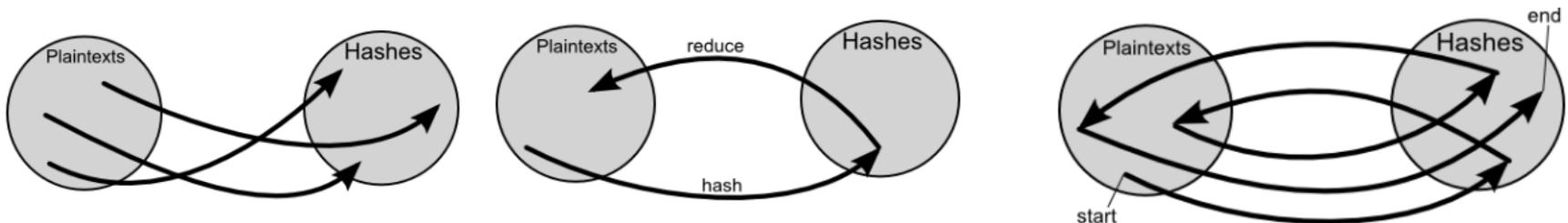
    car = new String("0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz!#");
    out = new String("");

    for (i = 0; i < 8; i++) {
        dat = (char) (val % 63);
        val = val / 63;
        out = out + car.charAt(dat);
    }

    return out;
}
```

# Coverage of Hash Chains

- The reduction function determines the range (i.e., length and composition) of plaintext (i.e., password) candidates that are covered
- Example:
  - Consider the password “Domino5”
  - In order to have this word stored in a chain, the reduction function must create outputs that are
    - At least 7 characters long
    - Contain small and capital letters, as well as numbers
  - Also, hash chains may not be able to cover all possible character combinations



# Pseudo-Code to create a single Chain

- This example creates a chain with the start value “abcdefg” that covers 10,001 plaintext words
- Note that the last value of this chain is a hash value (i.e. ciphertext)
- We don't know for certain what type of words the reduction function returns, possible only words of length 7 that consist of small letters only

```
String plaintext, first, ciphertext;

plaintext = first = "abcdefg";

for ( int i=0; i<10000; i++ ) {
    ciphertext = hash_it (plaintext);
    plaintext = reduce_it (ciphertext);
}

System.out.printf ("%s:%s\n", first, ciphertext);
```

# Chain Lookup

Assume we have a table with just 2 chains (with start and end values), i.e.

`aaaaaa` → 173bdfede2ee3ab3 → ... → `8995tg` → 9fdde3a0027fbb36 → ... → `k3rtol`  
`hfk39f` → 856385934954950 → ... → `delphi` → 759858fde66e8aa8 → ... → `prp56e`

... and a hash value “759858fde66e8aa8” we’d like to crack

Starting with this hash value we apply consecutively “→” and “→”, until we

- hit a known end value (e.g., `k3rtol`), or
- have repeated “→” and “→” x times (with x being the length of the chain)

If we hit a known end value, e.g. “`prp56e`”, we repeat the transformation, beginning with the start value of the chain, i.e., “`hfk39f`”, until we hit “759858fde66e8aa8” again

The input that led to the hash value (i.e., “`delphi`”) is the solution

# Chain Lookup Pseudocode

1. Input: Hash value  $H$
2. Reduce  $H$  into another plaintext  $P$
3. Look for the plaintext  $P$  in the list of final plaintexts (i.e. end values), if it is there, break out of the loop and goto step 6.
4. If it isn't there, calculate the hash  $H$  of the plaintext  $P$
5. Goto step 2., unless you've done the maximum amount of iterations
6. If  $P$  matches one of the final plaintexts, you've got a matching chain; in this case walk through the chain in question again starting with the corresponding start value, until you find the text that translates into  $H$

# Chain Collisions

- Consider the following scenario:

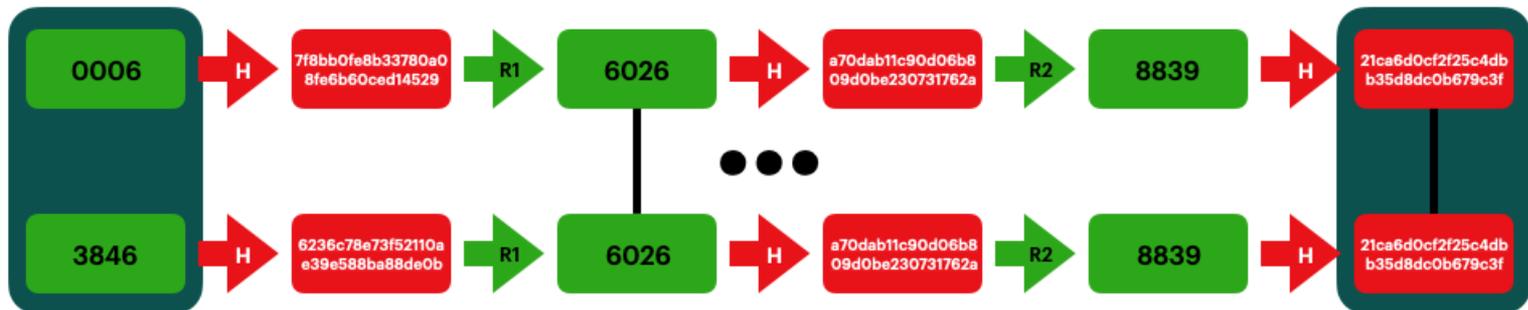
aaaaaa → ... → 173bdfede2ee3ab3 → delphi → 759858fde66e8aa8 → ... → prp56e  
hfk39f → ... → 856385934954950 → delphi → 759858fde66e8aa8 → ... → prp56e

- These 2 chains could merge, because

- ▣ the reduction function translates two different hashes into the same password (as reduction functions are imperfect), or
- ▣ the hash function translates two different passwords into the same hash (which should not happen → see hash function requirements)

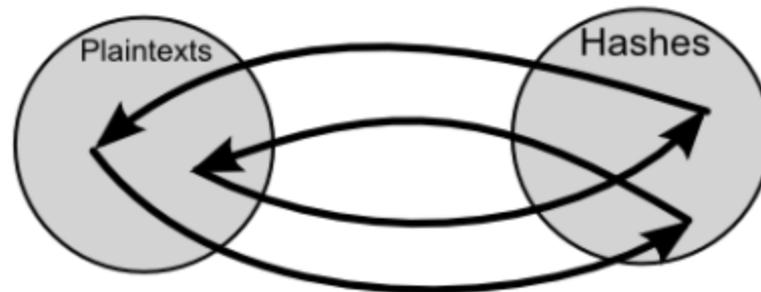
- Because of these collisions or chain loops (next slide) hash chains will not cover as many passwords as theoretically possible despite having paid the same computational cost to generate

- ▣ Previous chains are not stored in their entirety; therefore, it is impossible to detect this efficiently



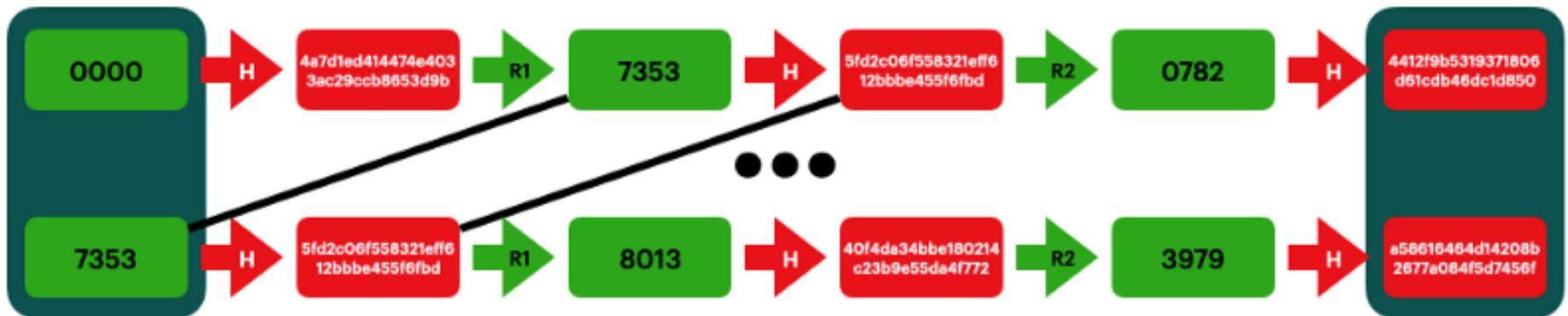
# Chain Loops

- Here you find repetitions of hashes in a single chain
- The result of imperfect reduction functions that map two different hashes into the same plaintext



# Rainbow Tables

- Rainbow tables effectively solve the problem of collisions with ordinary hash chains by replacing the single reduction function  $R$  with a sequence of related reduction functions  $R_1$  through  $R_k$  (one reduction function per chain element)
- In this way, for two chains to collide and merge they must hit the same value on the same iteration, which is rather unlikely



# Example for a Reduction Function for a Rainbow Table

27

```
private static String reductionFunction(long val, int round) { // Note that for the first function call "round" has to be 0,
    String car, out; // and has to be incremented by one with every subsequent call.
    int i; // I.e. "round" created variations of the reduction function.
    char dat;

    car = new String("0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz!#");
    out = new String("");

    for (i = 0; i < 8; i++) {
        val -= round;
        dat = (char) (val % 63);
        val = val / 63;
        out = out + car.charAt(dat);
    }

    return out;
}
```

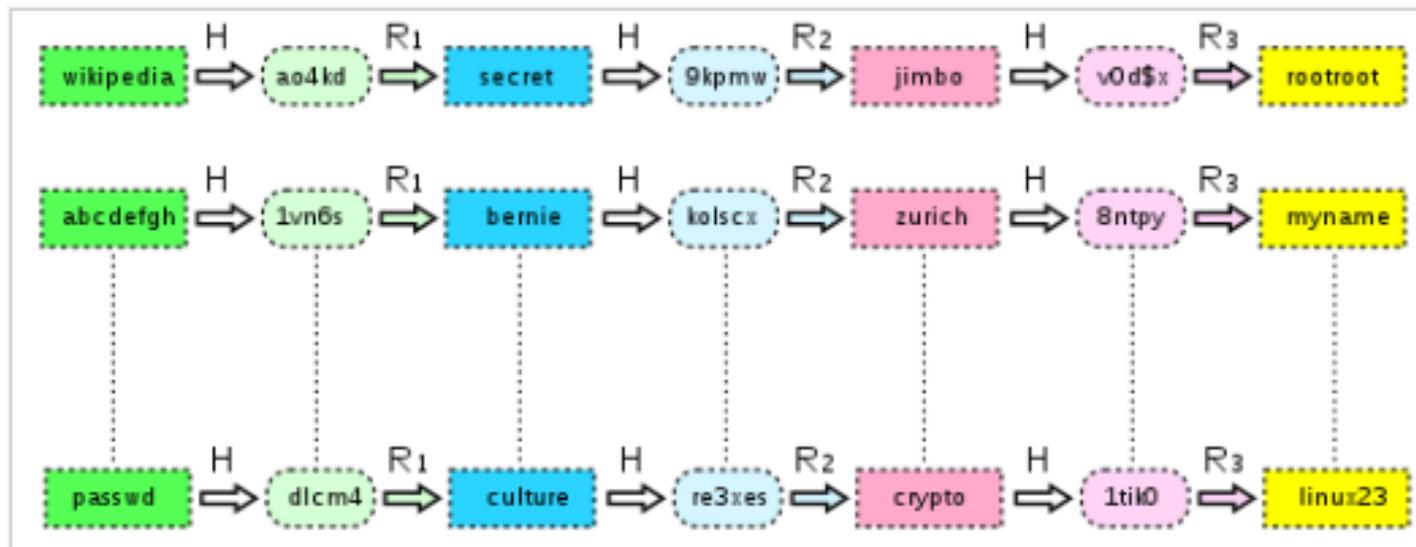
# Coverage of Reduction Functions

- Rather than calculating a random string a reduction function may calculate an integer index value to identify an entry (word) in a large (password) dictionary
- Example:
  - $H(\text{lalo}) = 368437\text{FDA}$
  - $R(368437\text{FDA}) = 6 \rightarrow \text{dict}[6] = \text{robot1 23}$
  - $H(\text{robot1 23}) = \text{DDA0087e73}$
  - ...
- This is similar to a lookup table, but requires far less space, as hashes are not stored
- However, it may be difficult to design a hash function that covers all dictionary indices

#	dict entry
0	Dog5
1	Simple
2	fEED2
3	lalo
4	mEn
5	hat
6	robot123
7	rose
...	

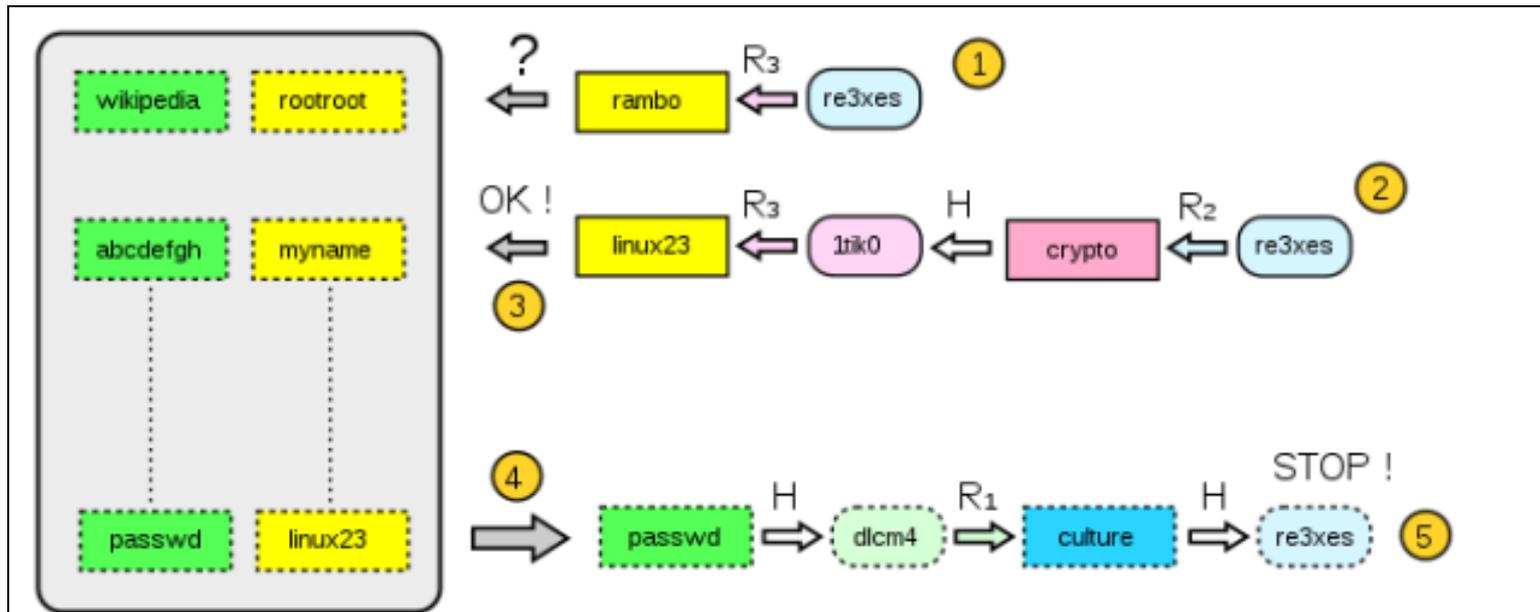
# Searching a Rainbow Table (Wikipedia)

- Let's assume a Rainbow table of length 3 with 3 different reduction functions  $R_1$ ,  $R_2$  and  $R_3$
- Again, we just store start (green) and end (yellow) value of each chain



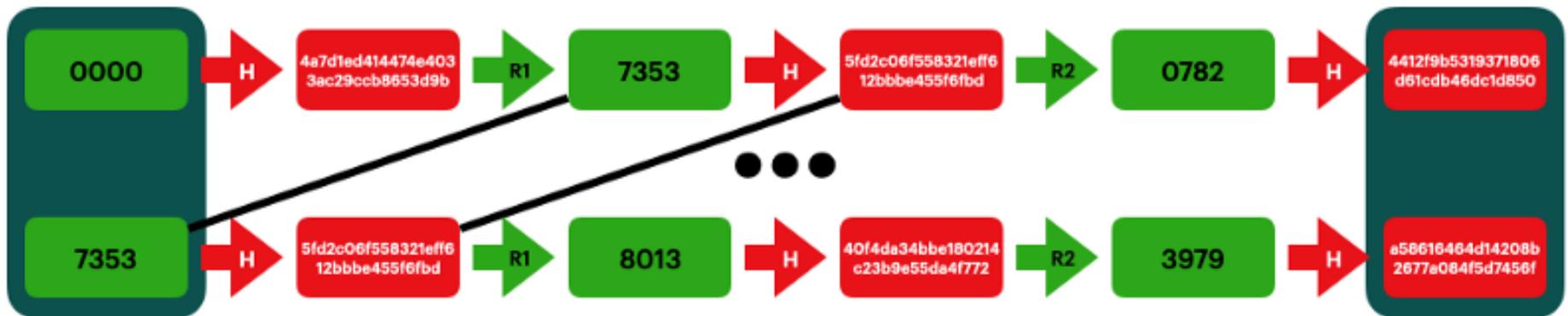
# Searching a Rainbow Table (Wikipedia)

- Consider you have the rainbow table below and the password hash “re3xes”
  - ▣ Calculate  $R_3(\text{“re3xes”})$  and check if the result matches any of the chain ends (yellow boxes)
  - ▣ Calculate  $R_2(H(R_3(\text{“re3xes”})))$  and check if the result matches any of the chain ends
  - ▣ ..
  - ▣ Repeat this process until the algorithm reaches  $R_1$ , or a match is found
  - ▣ If a match is found, traverse through the chain in question as seen before, to find the solution



# Perfect and non-perfect Rainbow Tables

- ◆ In a **perfect rainbow table** any word does not appear in more than one chain
- ◆ **Non-perfect rainbow tables (as shown below)** have redundant entries
  - They are easier to compute, but less memory-efficient because of these repetitions (which are not collisions!)



# Defense against Rainbow Tables

## □ Idea:

- ▣ Increase the (required minimum) length of a password
- ▣ By doing so there are many more potential passwords to be considered by a rainbow table ...
  - ... up to a point where such tables are simply no more economical to generate
- ▣ Increasing the password length can be either done by the
  - password owner (e.g., on the client side), or
  - algorithmically (e.g., on the client or server side)

# Defence against Rainbow Tables

## Client-side defence:

- A user requirement to choose long passwords that contain different types of characters,  
e.g. consider passwords that contain “A...Z”, “a...z”, “1-8”:
  - ▣ 6 characters long passwords result in  $6^{60} = 46,656,000,000$  combinations
  - ▣ 10 characters long passwords result in  $10^{60} = 604,661,760,000,000,000$  combinations

## Server- (and potentially client-) side defence:

### 1. Password salting

- ▣ A unique and random, but known string (“salt”) per user that is appended to each password before its hash is calculated
- ▣ The salt is stored in the user database

User ID	Salt	Password Hash	Password (not part of table)
ms@gmail.com	12367	1d8922d005733...	12367KenSentme!
k51@outlook.com	56f87	628749afdb83...	56f87Fluffybear
abd@yahoo.com	465d0	980ade367fc93...	46d05Limerick

# Defense against Rainbow Tables

## 2. Password peppering

- ▣ Similar to Salting, but a unique *secret* string is concatenated to all passwords before they are hashed

## 4. Multiple iterations

- ▣ A password is hashed multiple (e.g., 1 000) times before stored in the database

## 5. Combination approach

- ▣ Different techniques are combined to create a complex hash algorithm, e.g.,
- ▣  $\text{NewHash}(\text{password}) = \text{hash}(\text{hash}(\text{password}) \parallel \text{salt})$

# SQL Attacks

Some revision material covering

- SQL
- HTTP get / post Methods and PHP
- SQL injection attacks
- SQL injection attack mitigation strategies

# What are SQL Injections?

36

- ❑ SQL injection is a code injection technique, used to attack data-driven applications, in which malicious SQL statements are inserted for execution
- ❑ A way of exploiting user input and SQL Statements to compromise the database and/or retrieve sensitive data
- ❑ Such attacks are closely linked to various web technologies, i.e. HTTP and PHP

# HTTP get / post Methods and PHP

37

- ❑ PHP is a general-purpose server-side scripting language especially suited to web development
- ❑ PHP originally stood for Personal Home Page, but it now stands for the recursive initialism PHP: Hypertext Pre-processor
- ❑ The HTTP GET method sends the encoded user information appended to the page request
- ❑ The page and the encoded information are separated by the ? Character
- ❑ Example: <http://www.test.com/index.htm?name1=value1&name2=value2>
- ❑ PHP provides \$\_GET associative array to access all the sent information using GET method, e.g.

foo.php:

```
<?php
...
$var1 = $_GET['first_name'];
...
```

```
<form method="GET" action="foo.php">
First Name: <input type="text" name="first_name" /> <br />
Last Name: <input type="text" name="last_name" /> <br />
<input type="submit" name="action" value="Submit" />
</form>
```

# HTTP get / post Methods and PHP

38

- The POST method transfers information via HTTP headers
- The information is encoded as described in case of GET method and put into a header called QUERY\_STRING
- The POST method does not have any restriction on data size and type to be sent
- The data sent by POST method goes through HTTP header (rather than the page request)
- PHP provides \$\_POST associative array to access all the sent information using POST method

```
foo.php:  
<?php  
...  
$var1 = $_POST['first_name'];  
...
```

```
<form method="POST" action="foo.php">  
  
First Name: <input type="text" name="first_name" /> <br />  
Last Name: <input type="text" name="last_name" /> <br />  
  
<input type="submit" name="action" value="Submit" />  
  
</form>
```

# SQL Syntax Review

39

- Basic select query:

```
SELECT <columns> FROM <table> WHERE  
<condition>
```

- Example:

```
SELECT * FROM user WHERE id = 1 AND pass =  
'bla'
```

- Note:

- ▣ Literal strings are delimited with single quotes
- ▣ Numeric literals aren't delimited

# SQL Syntax Review

40

- Some databases allow semicolons to separate multiple statements:

```
DELETE FROM user WHERE id = 1; INSERT INTO  
user (id, pass) VALUES (1, 'secure');
```

- For most SQL variants, the sequence `--` means the rest of the line should be treated as a comment

# SQL Code Injection Example

41

```
1  <!--
2  Login code
3  -->
4  <?php
5  require_once('connection.php');
6
7  $email = $password = $pwd = '';
8
9  $email = $_POST['username'];
10 $pwd = $_POST['password'];
11
12 $password = MD5($pwd);
13
14 $sql = "SELECT * FROM tblclinician WHERE Email='$email' AND Password='$password'";
15 $result = mysqli_query($conn, $sql);
16
17 if(mysqli_num_rows($result) > 0)
18 {
19     ...
20     header("Location: searchpat1.php");
21 }
22 else
23 {
24     header("Location: loginfailed.php");
25 }
26 ?>
```

# SQL Code Injection Example

42



```
$email = $_POST['username'];  
$pwd = $_POST['password'];  
  
$password = MD5($pwd);  
  
$sql = "SELECT * FROM tblclinician WHERE Email='$email' AND Password='$password';"  
$result = mysqli_query($conn, $sql);
```

Table tblclinician:

Email	Hashed Password
ms@mail.ie	af47f8d1ac4
...	...

# SQL Code Injection Example

43

```
‘; DROP TABLE tblclinician; --
```



**Member Login**

Username :

Password :

```
$sql = "SELECT * FROM tblclinician WHERE Email='"; DROP  
TABLE tblclinician; --' AND Password='"
```

- Note: The SQL DROP TABLE statement deletes an existing table in a database
- While an attacker does not know the tables' names, the attacker can do a **blind attack**
- More generally, If DB details are not known to the attacker, **blind SQL injections** are used

# Other Code Injections if DB structure is known

44

- ❑ `SELECT * FROM tblclinician WHERE Email =''; INSERT INTO tblclinician (Email>Password) VALUES ('hacker',1 23);--' AND `Password`='`
- ❑ `SELECT * FROM `login` WHERE Email =''; UPDATE tblclinician SET Password = 1284ffa WHERE Email = ms@mail.ie ;--' AND `Password`='`
- ❑ The first injection creates a new user (hacker) including password hash
- ❑ The second injection replaces a user's password hash

# Types of SQL Injection Attacks

45

- ❑ **Blind SQL Injection**
  - ❑ Enter an attack on one vulnerable page but it may not display results
  - ❑ A second page would then be used to view the attack results
- ❑ **Conditional Response**
  - ❑ Test input conditions to see if an error is returned or not
  - ❑ Depending on the response, the attacker can determine yes or no information
- ❑ **First Order Attack**
  - ❑ Runs right away
- ❑ **Second Order Attack**
  - ❑ Injects data which is then later executed by another activity (job, etc.)
- ❑ **Lateral Injection**
  - ❑ Attacker can manipulate values using implicit functions

# What is at Risk?

46

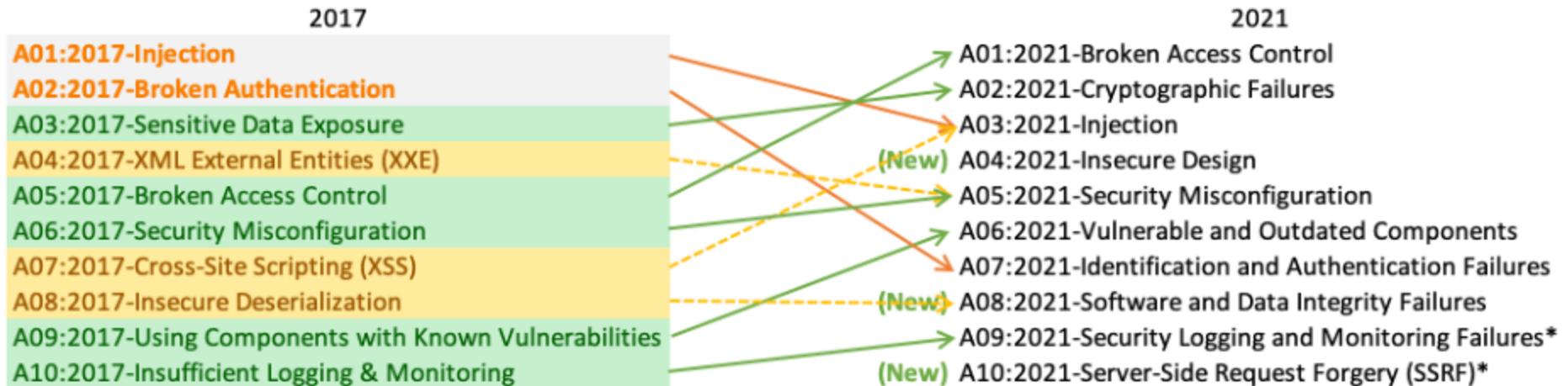
- Any web application that accepts user input
  - ▣ Both public and internal facing sites
  - ▣ Public facing sites will likely receive more attacks than internal facing sites
- For the last couple of years (i.e. since 2013), (SQL) Injection is one of the frontrunners on the OWASP top ten list
  - ▣ A well understood attack, but still not fully grasped by the developer community



# OWASP Top 10

47

- The Open Web Application Security Project (OWASP) is a non-profit foundation dedicated to improving the security of software



\* From the Survey

# Some historical Notes

48

- Guess Inc. is an American clothing brand and retailer
- Guess.com was open to a SQL injection attack
- In 2002 Jeremiah Jacks discovered the hole and was able to pull down 200,000 names, credit card numbers and expiration dates in the site's customer database
- The episode prompted a year-long investigation by the US Federal Trade Commission



# Some historical Notes

49

- In 2003 JJ used an SQL injection to retrieve 500,000 credit card numbers from PetCo
- In 2014 Russian hackers used a Botnet to recover a vast collection of stolen data, including 1.2 billion unique username/password pairs, by compromising over 420,000 websites using SQL injection techniques



# What can SQL Injections do?

50

- Retrieve sensitive information, including
  - ▣ Usernames/ **Passwords**
  - ▣ Credit Card information
  - ▣ Social Security / PPS numbers
- Manipulate data, e.g.
  - ▣ Delete records
  - ▣ Truncate tables
  - ▣ Insert records
- Manipulate database objects, e.g.
  - ▣ Drop tables
  - ▣ Drop databases

# What can SQL Injections do?

51

- Retrieve System Information
  - ▣ Identify software and version information
  - ▣ Determine server hardware
  - ▣ Get a list of databases
  - ▣ Get a list of tables
  - ▣ Get a list of column names within tables
- Manipulate User Accounts
  - ▣ Create new sysadmin accounts
  - ▣ Insert admin level accounts into the web-app
  - ▣ Delete existing accounts

CT437

COMPUTER SECURITY AND FORENSIC COMPUTING

PUBLIC KEY CRYPTOGRAPHY

Dr. Michael Schukat



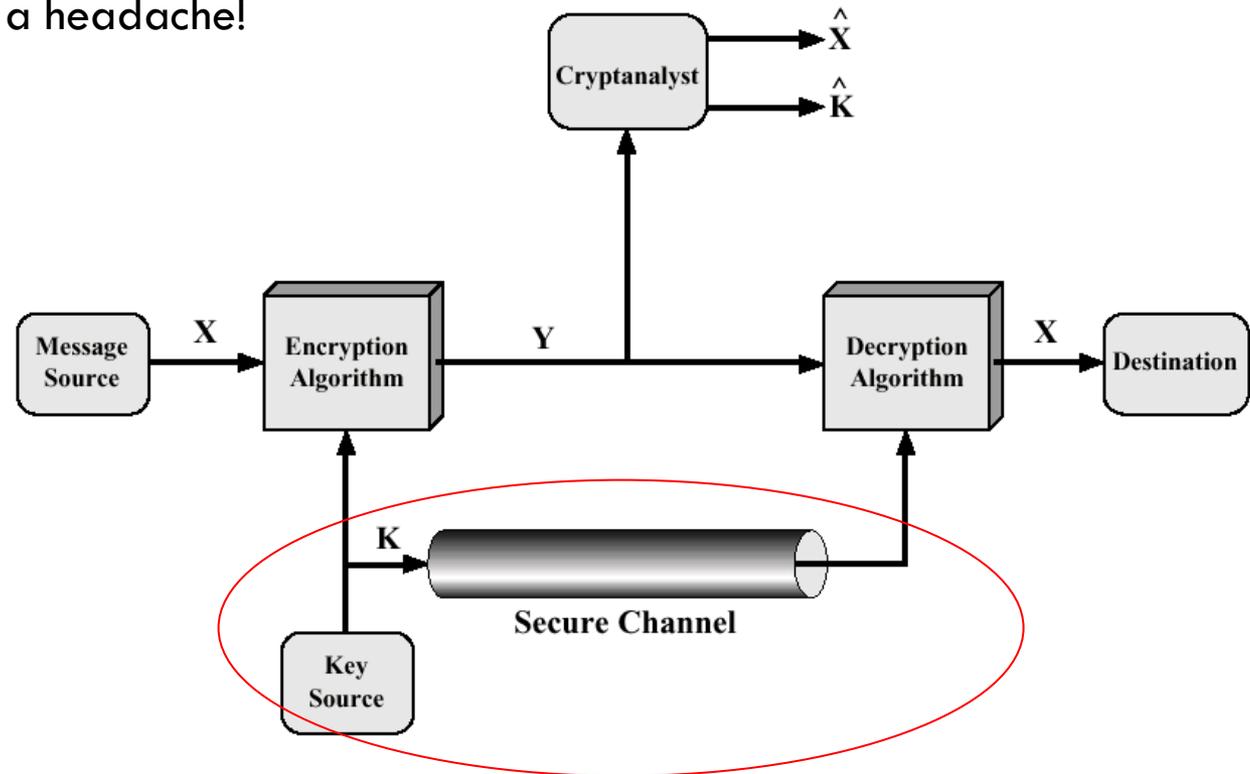
# Lecture Content

2

- Public key cryptography versus private key cryptography
- Public key cryptography applications
- Diffie-Hellman Key exchange
  - ▣ Man-in-the-Middle (MitM) attacks
- RSA encryption
- Optimisation techniques for public key encryption
- ECC encryption
- The Double-Ratchet algorithm

# Model of Conventional Cryptosystem

Symmetric block ciphers are cryptographically strong,  
but key distribution can be a headache!



$$Y = E_K(X), X = E_K^{-1}(Y)$$

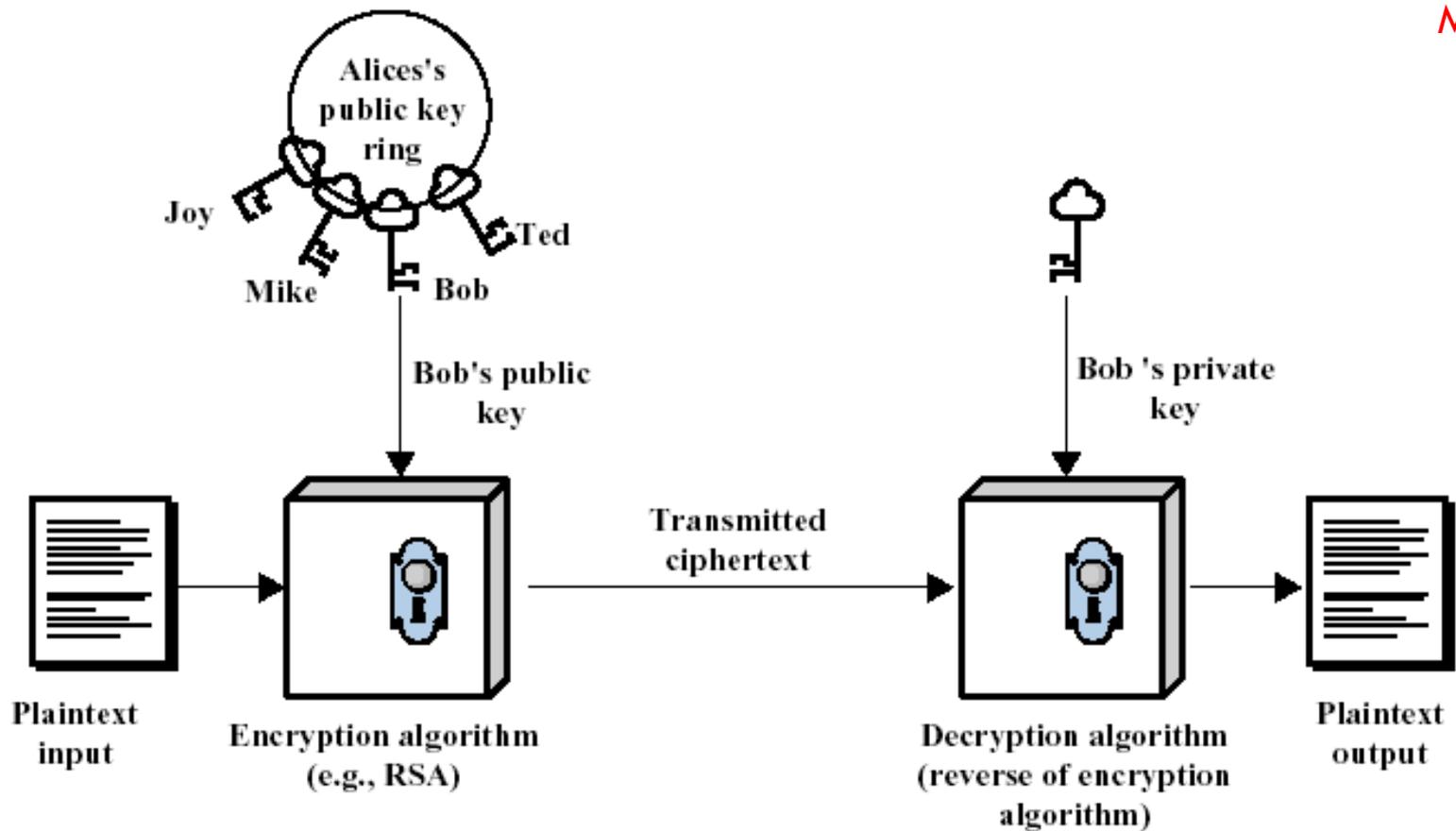
# Features and Limitations of Private-Key Cryptography

- Traditional symmetric/single key cryptography uses one key, shared by both sender and receiver
  - ▣ If this key is disclosed, communications are compromised
  
- The key is also symmetric, both parties are equal
  - ▣ This is problematic too, as it does not protect the sender from a situation, where:
    - the receiver forges a message using that key
    - and claims that it was sent by the sender
    - Think about an electronic contract that is exchanged between two business partners that use a shared key
    - One party can forge a contract and claim it was sent by the other side
    - Message authentication (HMAC or CMAC) doesn't solve the problem!

# Features of Public-Key Cryptography

- **Public-key/two-key/asymmetric cryptography** involves the use of two keys:
  - a **public-key**, which the owner shares with any peer; it is used to:
    - **Encrypt messages** send from the peer to the owner
    - Verify the integrity and origin of messages send from the owner to a peer (**signature validation**)
  - a **private-key**, known only to the recipient/owner, used to:
    - **Decrypt messages** that were encoded using their public key
    - Digitally sign data send to a peer (**signature creation**)
- The keys are **asymmetric**, because they are not equal
- Those who encrypt a message or verify a signature (using the receiver's public key) cannot decrypt the message or forge a signature
- It is computationally very hard (and infeasible) for an attacker to rebuild an owner's private key by analysing their public key
- This is achieved through the application of number- theoretic concepts

# Public-Key Encryption

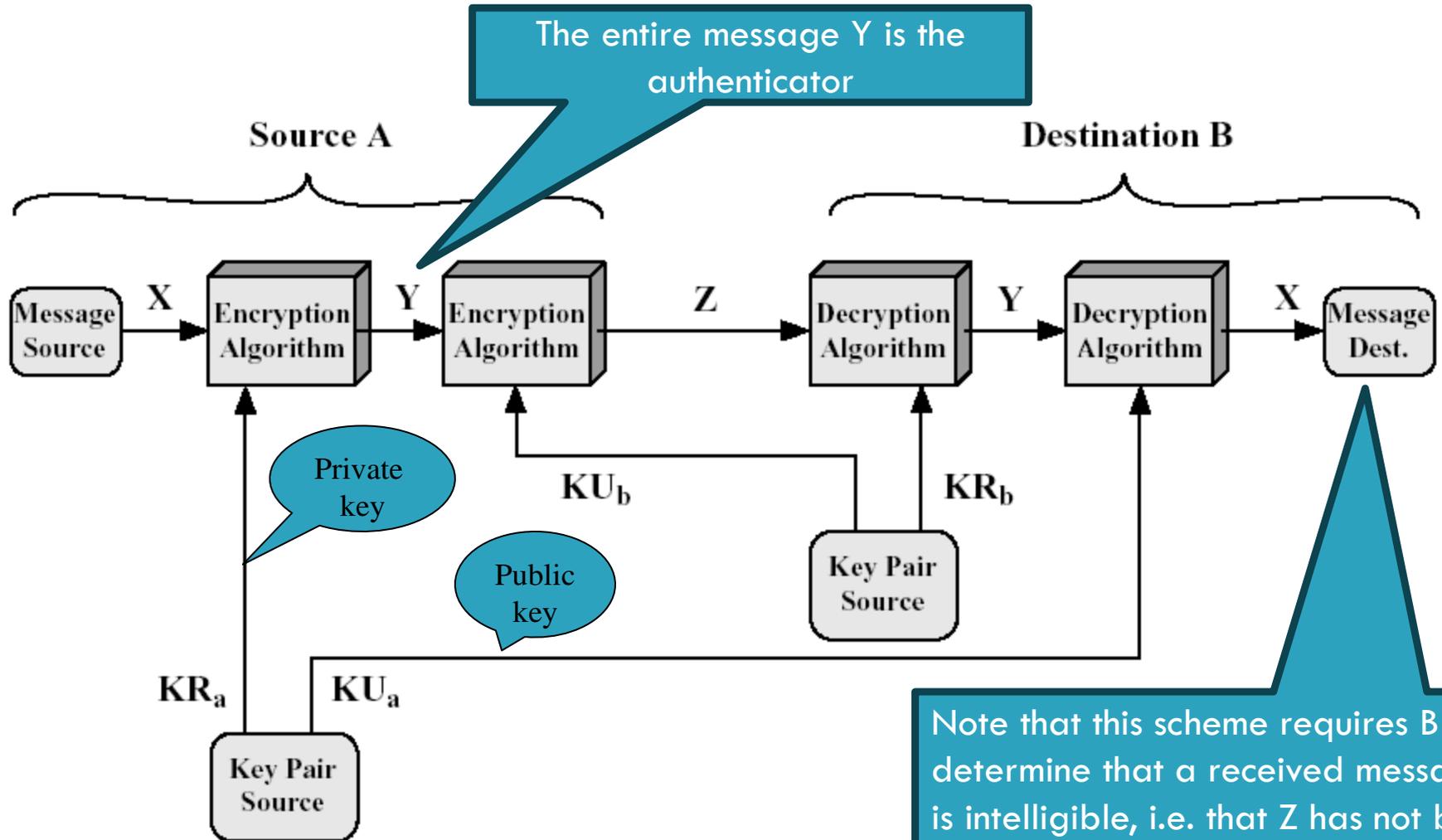


M

# Applications of Public-Key Cryptosystems

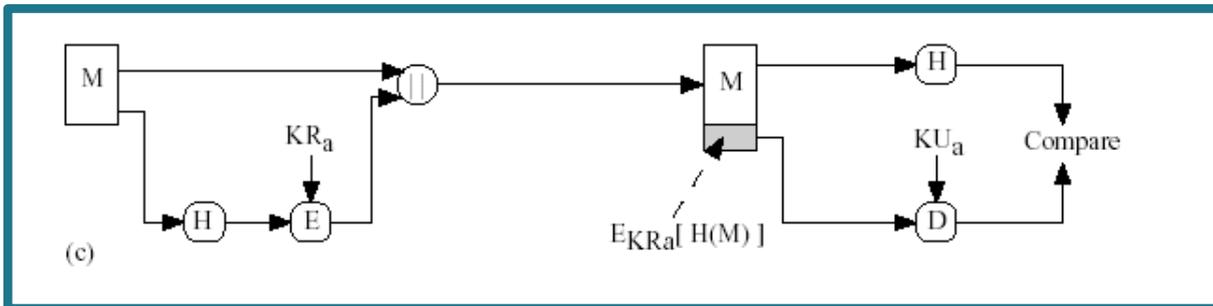
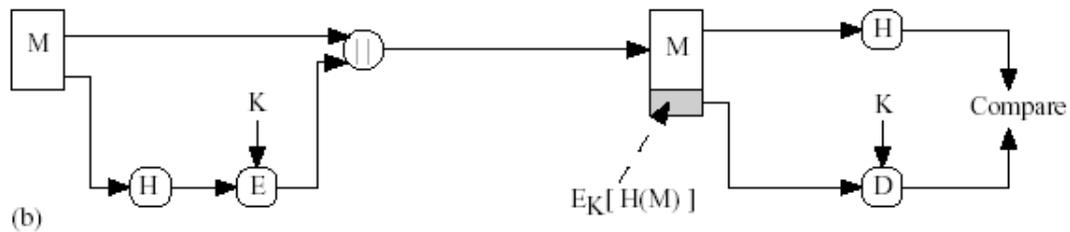
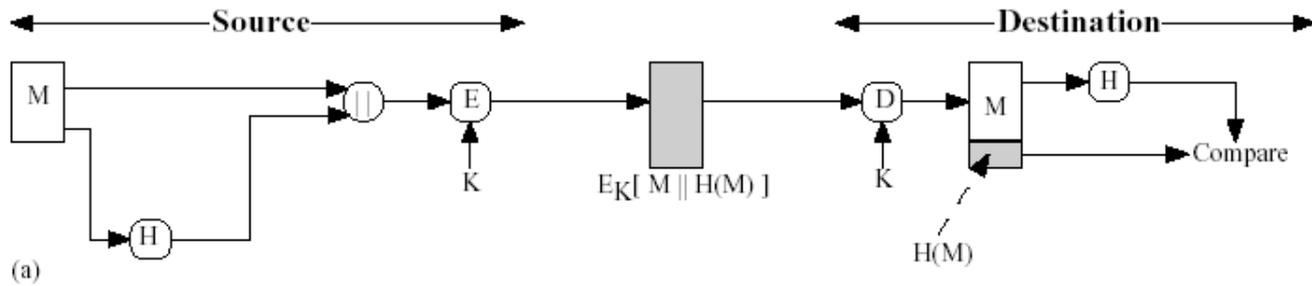
- **Data encryption/decryption:**  
The sender encrypts the message with the recipient's public key and the receiver decodes the message using their private key
  - Recall symmetric encryption where only **one** key is used
- **Digital signature/authentication:**  
The sender “signs” a message with their private key. Signing is achieved by encrypting the message or its MAC using their private key (next slide)
  - Recall private key encryption where sender and receiver just share one key
- **Key exchange:**  
Two sides negotiate a **symmetric** session key
  - Private key encryption is much faster than public key encryption
  - This key may also be used for conventional message authentication
- Note that in order to avoid confusion we use from now on the terms:
  - Symmetric key for private key encryption (block ciphers and stream ciphers)
  - Public and private keys for public key encryption

# Public-Key Cryptosystems: Secrecy and Authentication



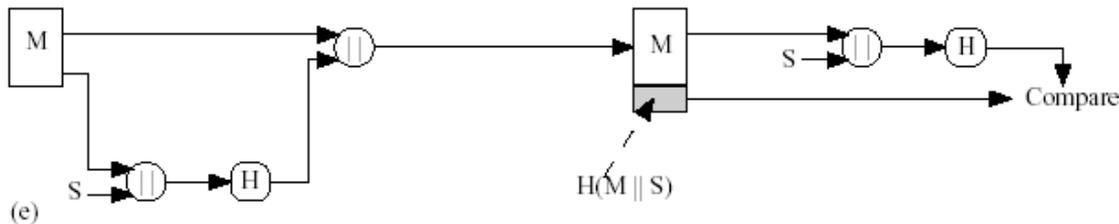
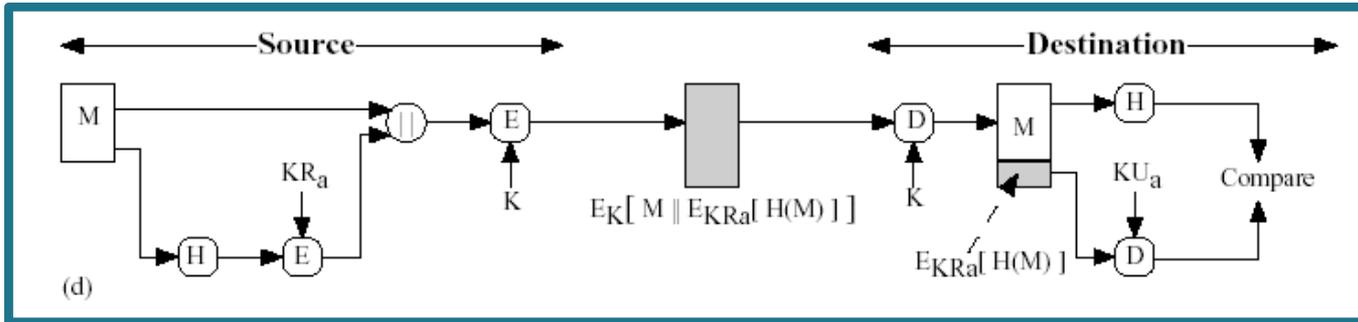
Note that this scheme requires B to determine that a received message is intelligible, i.e. that  $Z$  has not been manipulated by a MitM in transit

# Recap: Basic Uses of Hash Functions (H) in Combination with asymmetric Encryption (c)

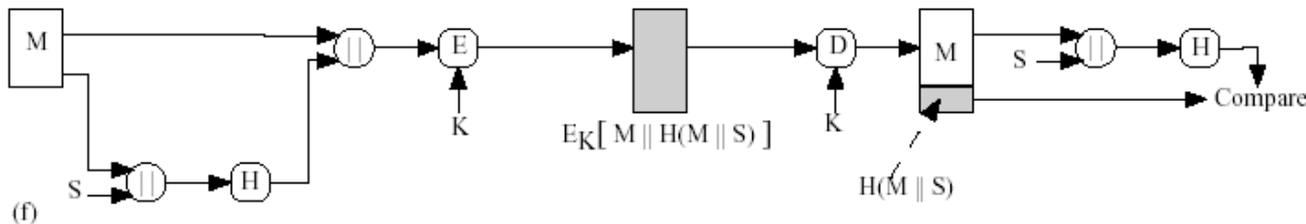


$KR_a$  = Sender's private key  
 $KU_a$  = Sender's public key

# Recap: Basic Uses of Hash Functions (H) in Combination with asymmetric Encryption (d)



$KR_a$  = Sender's private key  
 $KU_a$  = Sender's public key



# Public-Key Cryptosystems

11

- There are different cryptosystems, including (from simplest to most complex):
  - ▣ **Diffie Hellman key exchange**
  - ▣ **RSA**
  - ▣ **DSS**
  - ▣ **Elliptic Curve Cryptography**

Algorithm	Encryption/Decryption	Digital Signature	Key Exchange
RSA	Yes	Yes	Yes
Diffie-Hellman	No	No	Yes
DSS	No	Yes	No

# Modular Arithmetic

12

- Modular arithmetic is a system of arithmetic for integers, where numbers wrap around when reaching a certain value  $n$ , called the modulus
  - ▣ Recall modulus operator “%” in C and other languages, i.e. “division with rest” with rest being the modulus
  - ▣ Example:  $75 / 6 = 12$  remainder  $3 \rightarrow 75 \% 6 = 3$
- Numbers  $\{0, 1, \dots, n - 1\}$  are called “multiplicative group of integers modulo  $n$ ”, or simply  $Z_n$ , for some  $n > 0$
- Within  $Z_n$ , addition and multiplication is well defined!

# Example: Multiplication in $Z_9$

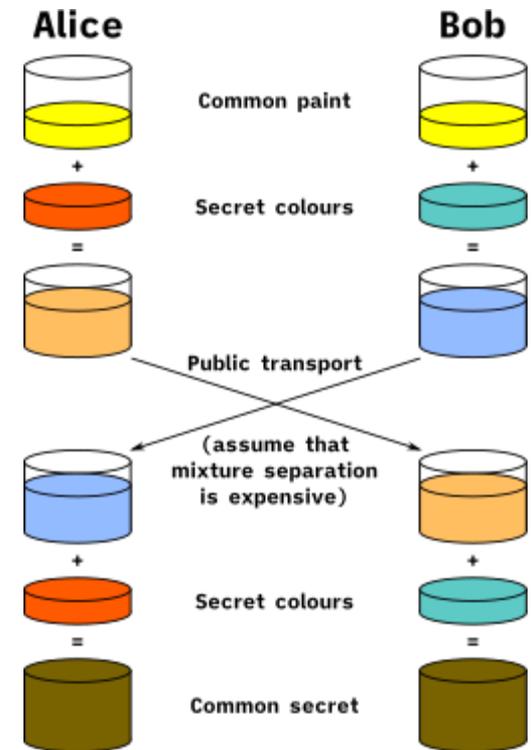
**Mx3**

<b>*</b>	<b>0</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>
<b>0</b>	0	0	0	0	0	0	0	0	0
<b>1</b>	0	1	2	3	4	5	6	7	8
<b>2</b>	0	2	4	6	8	1	3	5	7
<b>3</b>	0	3	6	0	3	6	0	3	6
<b>4</b>	0	4	8	3	7	2	6	1	5
<b>5</b>	0	5	1	6	2	7	3	8	4
<b>6</b>	0	6	3	0	6	3	0	6	3
<b>7</b>	0	7	5	3	1	8	6	4	2
<b>8</b>	0	8	7	6	5	4	3	2	1

# Illustration of Concept behind Diffie-Hellman Key Exchange (Wikipedia)

14

- Alice and Bob want to share a secret colour using public transport
  - ▣ i.e. an adversary (i.e. Mallory, not shown) can get samples of any colour that is exchanged between both
- Alice and Bob agree on a common “public” paint color (yellow in the example)
- Each of them add a secret colour and send their mix to the other party
  - ▣ Mallory can intercept both, but cannot separate the mixtures
- Alice and Bob receive the other’s mixture and add their secret colour
- Both colours are identical
- → This color is their common secret



# Diffie-Hellman Key Exchange

- Diffie-Hellman provides a mechanism for a secure key exchange between two endpoints
  - ▣ The negotiated key is subsequently used as a symmetric key (or as a seed for a key) for data encryption and message authentication (as seen before)
- The algorithm uses the multiplicative group of integers modulo  $q$ 
  - ▣  $q$  has typically a length of 1024 or 2048 bits
- It is based on the difficulty of computing discrete logarithms over such groups, e.g.

$$6^3 \bmod 17 = 216 \bmod 17 = 12 \quad (\text{easy})$$

$$12 = 6^y \bmod 17? \quad (\text{difficult})$$

- ▣ Recall  $6^3 = 6 \times 6 \times 6$ , so we need just the multiplication

- The core equation for the key exchange is

$$K = (A)^B \bmod q$$

# Diffie-Hellman: Global Public Elements

- Alice and Bob select:
  - ▣ A prime number  $q$  which determines  $Z_q$
  - ▣ A positive integer  $a$ , with  $1 < a < q$  and  $a$  is a **primitive root** of  $q$ 
    - Note that  $a$  is also called the generator
- **Definition:**  $a$  is a primitive root of  $q$ , if numbers  $a \bmod q, a^2 \bmod q, \dots, a^{(q-1)} \bmod q$  are distinct integer values between 1 and  $(q-1)$  (i.e. in  $Z_q$ ) in some permutation
- **Example:**  $a = 3$  is a primitive root of  $Z_5$  (i.e.  $q = 5$ ),  $a = 4$  is not:

$3^1 = 3 = 0 * 5 + 3$	$4^1 = 4 = 0 * 5 + 4$
$3^2 = 9 = 1 * 5 + 4$	$4^2 = 16 = 3 * 5 + 1$
$3^3 = 27 = 5 * 5 + 2$	$4^3 = 64 = 12 * 5 + 4$
$3^4 = 81 = 16 * 5 + 1$	$4^4 = 256 = 51 * 5 + 1$

# Primitive Roots of $\mathbb{Z}_n$ with $15 < n < 32$

17

$n$	primitive roots modulo $n$
16	
17	3, 5, 6, 7, 10, 11, 12, 14
18	5, 11
19	2, 3, 10, 13, 14, 15
20	
21	
22	7, 13, 17, 19
23	5, 7, 10, 11, 14, 15, 17, 19, 20, 21
24	
25	2, 3, 8, 12, 13, 17, 22, 23
26	7, 11, 15, 19
27	2, 5, 11, 14, 20, 23
28	
29	2, 3, 8, 10, 11, 14, 15, 18, 19, 21, 26, 27
30	
31	3, 11, 12, 13, 17, 21, 22, 24

# Generation of Secret-Key: Part 1

- Alice and Bob share publicly a prime number  $q$  and a primitive root  $a$
- Alice (User A):
  - ▣ Select secret number  $X_A$  with  $0 < X_A < q$
  - ▣ Calculate public value  $Y_A = a^{X_A} \bmod q$  (← difficult to reverse)
  - ▣  $Y_A$  is sent to Bob (user B)
- Bob (User B):
  - ▣ Select secret number  $X_B$  with  $0 < X_B < q$
  - ▣ Calculate public value  $Y_B = a^{X_B} \bmod q$  (← difficult to reverse)
  - ▣  $Y_B$  is send to Alice

# Generation of Secret-Key: Part 2

- Alice:

- Alice owns  $X_A$  and receives  $Y_B$

- She generates the secret key:  $K = (Y_B)^{X_A} \bmod q$

- Bob:

- Bob owns  $X_B$  and receives  $Y_A$

- Bob generates the secret key:  $K = (Y_A)^{X_B} \bmod q$

- **Both keys are identical!**

# Generation of Secret-Key: Part 2

$$\begin{aligned} K &= (YB)^{XA} \pmod q \\ &= (a^{XB} \pmod q)^{XA} \pmod q \\ &= (a^{XB})^{XA} \pmod q \\ &= a^{XB \cdot XA} \pmod q \\ &= a^{XA \cdot XB} \pmod q \\ &= (a^{XA})^{XB} \pmod q \\ &= (a^{XA} \pmod q)^{XB} \pmod q \\ &= (YA)^{XB} \pmod q \end{aligned}$$

# Example for Diffie-Hellman

- Alice and Bob agree on public values  $q$  and  $a$ , and determine their respective secrets  $X_A$  and  $X_B$  :
- Let  $q = 5$  and  $a = 3$
- Alice picks  $X_A = 2$ , therefore  $Y_A = a^{X_A} \bmod 5 = 4$
- Bob picks  $X_B = 3$ , therefore  $Y_B = a^{X_B} \bmod 5 = 2$
- Alice sends  $Y_A = 4$  to Bob
- Bob sends  $Y_B = 2$  to Alice
- Alice calculates:  $K = (Y_B)^{X_A} \bmod q = 2^2 \bmod 5 = 4$
- Bob calculates:  $K = (Y_A)^{X_B} \bmod q = 4^3 \bmod 5 = 4$

# Ephemeral versus Static Diffie-Hellman Keys

- The generated DH keys can be either
  - ▣ static (to be reused)
  - ▣ ephemeral (only used once, e.g., for one session only)
- Ephemeral keys
  - ▣ provide forward secrecy, but no endpoint authenticity
    - Forward secrecy: If the current key is recovered by an adversary, it only affects the current session, but no past or future sessions
- Static keys
  - ▣ do not provide forward secrecy
  - ▣ do provide (implicit) endpoint authenticity
  - ▣ do not protect against replay-attacks

# Example DH Parameters

23

□ Standardised, see <https://www.ietf.org/rfc/rfc3526.txt>

□ Example 2048-bit MODP Group

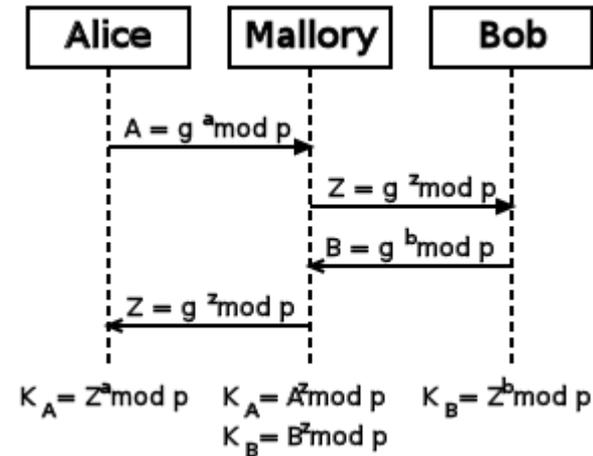
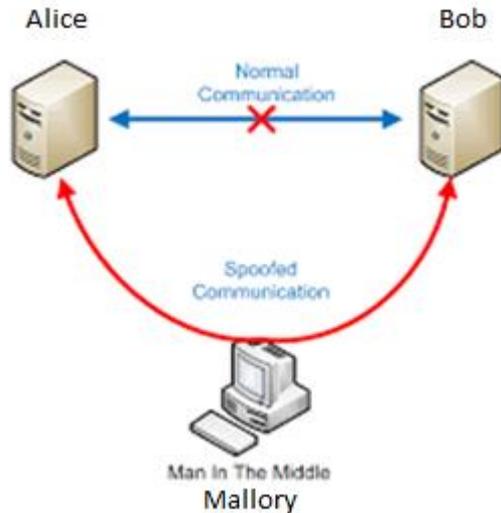
□ == rounded

□  $q = 2^{2048} - 2^{1984} - 1 + 2^{64} * \{ [2^{1918} \pi] + 124476 \}$

□  $q =$  FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1  
29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD  
EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245  
E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED  
EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE45B3D  
C2007CB8 A163BF05 98DA4836 1C55D39A 69163FA8 FD24CF5F  
83655D23 DCA3AD96 1C62F356 208552BB 9ED52907 7096966D  
670C354E 4ABC9804 F1746C08 CA18217C 32905E46 2E36CE3B  
E39E772C 180E8603 9B2783A2 EC07A28F B5C55DF0 6F4C52C9  
DE2BCBF6 95581718 3995497C EA956AE5 15D22618 98FA0510  
15728E5A 8AACAA68 FFFFFFFF FFFFFFFF

□  $a = 2$

# DH and Man-in-the-Middle (MitM) Attacks



- Mallory is a MitM attacker with the ability to intercept, and fabricate messages
  - ▣ Not to confuse with a Meet-in-the-Middle attack (→ double-DES and triple-DES)
- Both Alice and Bob are unaware of Mallory's existence, as there is no mutual authentication and an unprotected communication link
- Alice and Bob exchange their shared values (A and B in the example), but these are intercepted by Mallory
- Mallory completes both key exchanges sending her own shared value Z to both Alice and Bob
- By doing so, Mallory establishes two individual (secure) connections with Alice and Bob
- Alice and Bob have no idea that they became victims of a MitM attack!

# In-Class Activity: Diffie-Hellman MitM Attack

- Let  $q = 5$  and  $a = 3$ ;
- $X_{\text{Alice}} = 2$ , therefore  $Y_{\text{Alice}} = a^{X_{\text{Alice}}} \bmod 5 = 4$
- $X_{\text{Bob}} = 3$ , therefore  $Y_{\text{Bob}} = a^{X_{\text{Bob}}} \bmod 5 = 2$
- $X_{\text{Malory}} = 1$ , therefore  $Y_{\text{Malory}} = a^{X_{\text{Malory}}} \bmod 5 = 3$
- What session keys between
  - ▣ Alice and Malory
  - ▣ Malory and Bobare generated?
- Note: User A's key  $K = (Y_B)^{X_A} \bmod q$
- Note: User B's key  $K = (Y_A)^{X_B} \bmod q$

# Solution

26

- Alice sends “4” to Bob, but this message is intercepted by Malory
- Bob sends “2” to Alice, but this message is intercepted by Malory
- Malory sends “3” to both parties, claiming to be either Bob or Alice
- Alice receives “3” and calculates K as follow:  $K = 3^2 \bmod 5 = 4$ 
  - ▣ Malory calculates  $4^1 \bmod 5 = 4$
- Bob receives “3” and calculates K as follow:  $K = 3^3 \bmod 5 = 2$ 
  - ▣ Malory calculates  $2^1 \bmod 5 = 2$
- Alice and Bob think they just mutually agreed on a shared secret key
- From this point onwards Malory as a MitM can read, manipulate and fabricate messages between Alice and Bob

# The RSA Algorithm

- Published by Rivest, Shamir and Adleman in 1977, but first discovered by Clifford Cocks (British mathematician and cryptographer) in 1973
- The RSA scheme works similar to a block cipher, where a plaintext  $M$  and a ciphertext  $C$  are integers between 0 and  $n - 1$ , i.e. elements of  $Z_n$
- $M$  can be a plaintext message (block), a hash value, or a private key picked by the sender to be shared with the message recipient
  - E.g., “ABC” = “01000001 01000010 01000011” =  $4276803_{10}$
- Principle:
$$C = M^e \pmod n$$
$$M = C^d \pmod n = M^{ed} \pmod n$$
- Public key  $KU = \{e, n\}$
- Private key  $KR = \{d, n\}$
- With  $n$  sufficiently large it is infeasible to determine  $d$  given  $e$  and  $n$

# Key Generation for the RSA Algorithm

Euler's totient  
function Phi

## Key Generation

Select  $p, q$

$p$  and  $q$  both prime

Calculate  $n = p \times q$

Calculate  $\phi(n) = (p - 1)(q - 1)$

Greatest  
common divisor

Select integer  $e$

$\text{gcd}(\phi(n), e) = 1; 1 < e < \phi(n)$

Calculate  $d$

$d = e^{-1} \text{ mod } \phi(n)$

Public key

$KU = \{e, n\}$

Private key

$KR = \{d, n\}$

See next slide

# Example

- Let  $p = 7$ ,  $q = 11$  and  $n = pq = 77$
- $\phi(77) = (p - 1)(q - 1) = 6 \times 10 = 60$
- Factorisation of  $60 = 1 * 2 * 5 * 2 * 3$

Therefore, the divisors of 60 are: 2, 3, 5

- List of all integers  $x$ ,  $1 < x < 60$ , with  $\text{GCD}(60, x) = 1$ :  
7, 11, 13, 17, 19, 23, 29, 31, 37, 47, 49, 53, 59

□ Note that these integers either

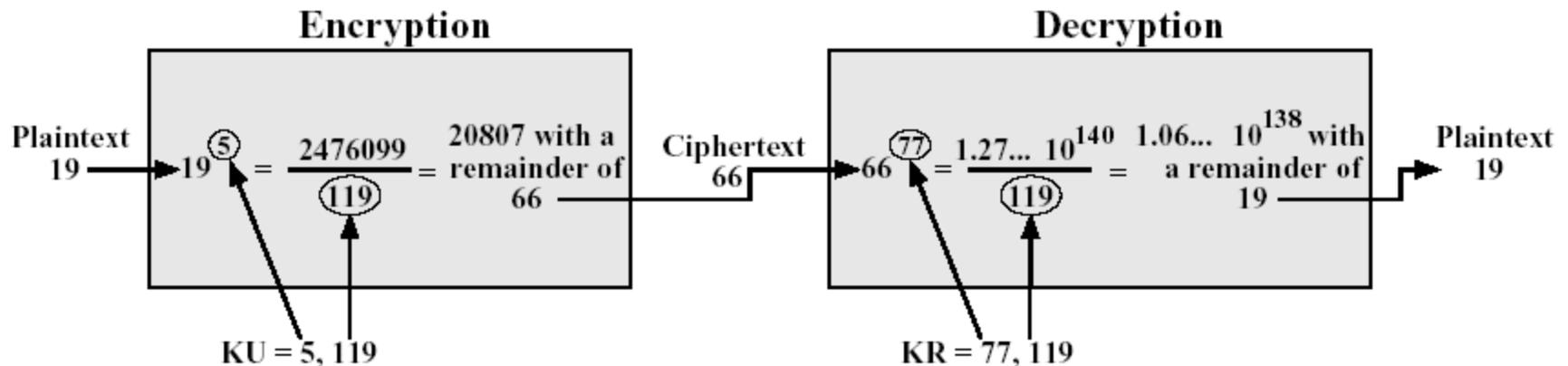
- are prime numbers (that cannot share a common divisor with 60), or
- do not share a common divisor with 60 (i.e., 7 and 49)

# Example (continued)

- Let  $e = 7$
- Choose  $d$  with  $ed = 1 \pmod{\phi(pq)} \Leftrightarrow$   
 $7d = 1 \pmod{60} \Leftrightarrow 7d \pmod{60} = 1$ 

$7*1 \pmod{60} = 7$	$7*2 \pmod{60} = 14$	$7*3 \pmod{60} = 21$
$7*4 \pmod{60} = 28$	$7*5 \pmod{60} = 35$	$7*6 \pmod{60} = 42$
$7*7 \pmod{60} = 49$	$7*8 \pmod{60} = 56$	$7*9 \pmod{60} = 3$
$7*10 \pmod{60} = 10$	$7*11 \pmod{60} = 17$	$7*12 \pmod{60} = 24$
...	$7*43 \pmod{60} = 1$	
- Therefore  $d = 43$
- Therefore  $KU = (7, 77)$  and  $KR = (43, 77)$
- Note there are better / more efficient algorithms (i.e. the Extended Euclidean Algorithm) to calculate  $d$

# Example for an Encryption/Decryption



## ❑ Obvious drawbacks:

- ❑ Very large numbers are to be computed
  - Ordinary integer or floating-point variables don't work
  - Instead, large number libraries need to be used
- ❑ This makes RSA encryption / decryption is very slow!

# Computational Aspects of Public Key Cryptography

- Assume you have to evaluate the expression  $C = 503^{23} \bmod 899$  as part of the encoding process
  - Note that the modulus is small enough to fit into an integer variable
- $503^{23} = 1.367929313795408423250439710106 \times 10^{62}$  cannot be properly represented using an ordinary integer or floating-point variable!
- In order to solve this problem, the exponentiation must be broken down into smaller steps, e.g.
  - $$503^{23} \bmod 899 = ((503^6 \bmod 899) \times (503^6 \bmod 899) \times (503^6 \bmod 899) \times (503^5 \bmod 899)) \bmod 899$$
  - $$503^6 \bmod 899 = ((503^3 \bmod 899) \times (503^3 \bmod 899)) \bmod 899$$
  - $$503^5 \bmod 899 = ((503^3 \bmod 899) \times (503^2 \bmod 899)) \bmod 899$$
  - $$503^3 \bmod 899 = ((503^2 \bmod 899) \times 503) \bmod 899$$

# Computational Aspects of Public Key Cryptography

- ... or even iteratively:

$$503^{23} \bmod 899 = \\ ((((((503^2 \bmod 899) \times 503) \bmod 899) \times 503) \bmod 899) \times \dots \times 503) \bmod 899$$

- This expression consists of 22 nested multiplications and 22 nested modulus operations and can be easily calculated by using a loop
- However, once a single number squared is too large to fit into a 32-bit or 64-bit (unsigned) integer variable, a big number library must be used

# The Security of RSA

- There are various angles to attack the RSA algorithm:
  - ▣ Brute force: Trying all possible private keys (not a great idea!)
  - ▣ Mathematical attacks: Factor  $n$  (which is the product of two primes); see some very old data below:

Number of Decimal Digits	Approximate Number of Bits	Data Achieved	MIPS-years	Algorithm
100	332	April 1991	7	quadratic sieve
110	365	April 1992	75	quadratic sieve
120	398	June 1993	830	quadratic sieve
129	428	April 1994	5000	quadratic sieve
130	431	April 1996	500	generalized number field sieve

- ▣ See also (for some more recent data)  
[https://en.wikipedia.org/wiki/RSA\\_numbers#RSA-704](https://en.wikipedia.org/wiki/RSA_numbers#RSA-704)
- ▣ Timing attacks: Based on analysis of the run time of an decryption algorithm

# Breaking RSA

37

- Consider the key pair  $(e, n)$  and  $(d, n)$  or simply  $(e, n)$  and  $d$ 
  - ▣  $n = p * q$ , with  $p$  and  $q$  being large (secret!) primes
- Factorising  $n$  is unfeasible for very large  $n$
- However, let's assume  $n$  can be factored into  $p$  and  $q$
- The adversary can now do the following calculations:
  - ▣  $\phi(n) = (p - 1) * (q - 1)$
  - ▣ Identify  $d$ , so that  $e * d = 1 \pmod{\phi(n)}$ 
    - $e$  is known, use the aforementioned Extended Euclidean Algorithm

# Step 1: Factorise N

38

```
// This is a very lightweight integer factoring algorithm, not very efficient or
// sophisticated.
// Assume n is the product of two primes p1 and p2
void factorise(int n) {
    int i;
    for (p1 = 2; i <= sqrt(n); i++) {
        if (n % p1 == 0)
            printf("n = %d; p1 = %d; p2 = %d\n", n, p1, n / p1);
        break;
    }
}
// Note that the integer values above would be replaced with large number
// representations, i.e., BIGNUM in OpenSSL
```

# Step 2: Determine e

39

// We know p and q (n was successfully factorised), d is in the public key KR= d, n  
// This is again a very lightweight algorithm, not very efficient or sophisticated.

```
int breakRSA(int p, int q, int d) {  
    int prod, found = 0, start = 1, df = -1;  
    int phi = (p - 1) * (q - 1);  
    while ((!found) && (start < phi)) { // exit if needed  
        prod = d * start;  
        if (prod % phi == 1) found = 1;  
        else start++;  
    }  
    if (found) df = start;  
    return (df);  
}
```

// Note that the integer values above would be replaced with large number  
// representations, i.e., `BIGNUM` in `OpenSSL`

# How to choose p and q

40

- When choosing p and q, the following should be considered:
  1.  $p \neq q$ , as  $p = q = \sqrt{n}$
  2. Neither p or q must not be “small”, as factorising could produce a result in a reasonable amount of time (see previous slide “Step 1: Factorise N”)
  3. p must not be similar in size to q, because of *Fermat's method of factoring a composite number N*:
    - N can be represented as the difference of two squares:
      - $p * q = N \Leftrightarrow a^2 - b^2 \Leftrightarrow (a - b) (a + b) [= p * q]$
    - $N = a^2 - b^2$  can be rewritten as:  $b^2 = a^2 - N$
    - To find a solution, iterate through a (starting with  $\text{round}(\sqrt{N})$ ), until  $a^2 - N$  is a square number (i.e.  $b^2$ )

# Fermat's Factoring Algorithm

41

```
// This function assumes N can be factorised. It returns N's factors
// p and q, using "pass by reference" pointers, so that both values
// are returned.
```

```
void fermatFactor(int N, int *p, int *q) {
    int a = ceiling(sqrt(N)); // start value for a
    int b2 = a * a - N; // see last slide
    while (sqrt(b2) * sqrt(b2) <> b2) { // is b2 a square?
        a = a + 1; // No, so increment a ...
        b2 = a * a - N; // ... and update b2
    }
    *p = a - sqrt(b2);
    *q = a + sqrt(b2);
}
```

If  $p (= a - b)$  and  $q (= a + b)$  are similar in size, it takes only a small number of iterations over  $a$  to find a solution

# Example

42

1.  $n = 33$  (based on secret values  $p = 3$  and  $q = 11$ )
2. First iteration:  $a = 6$  (i.e.,  $\text{ceiling}(\text{sqrt}(33))$ ):
  1.  $b^2 = 6 * 6 - 33 = 3$
  2.  $b^2$  is not a square number
  3.  $a = a + 1$
3. Second iteration:  $a = 7$ :
  1.  $b^2 = 7 * 7 - 33 = 16$
  2.  $b^2$  is a square number
4. Calculate  $p$  and  $q$ :
  1.  $p = 7 - \text{sqrt}(16) = \underline{3}$
  2.  $q = 7 + \text{sqrt}(16) = \underline{11}$

# Breaking RSA in Practise

43

<https://arstechnica.com/information-technology/2022/03/researcher-uses-600-year-old-algorithm-to-crack-crypto-keys-found-in-the-wild/>



# CVE-2022-26320

44

NIST

≡ NVD MENU

[Information Technology Laboratory](#)

NATIONAL VULNERABILITY DATABASE

NIST NATIONAL VULNERABILITY DATABASE NVD

VULNERABILITIES

## CVE-2022-26320 Detail

### Description

The Rambus SafeZone Basic Crypto Module before 10.4.0, as used in certain Fujifilm (formerly Fuji Xerox) devices before 2022-03-01, Canon imagePROGRAF and imageRUNNER devices through 2022-03-14, and potentially many other devices, generates RSA keys that can be broken with Fermat's factorization method. This allows efficient calculation of private RSA keys from the public key of a TLS certificate.

### Severity

CVSS Version 3.x

CVSS Version 2.0

CVSS 3.x Severity and Metrics:



NIST: NVD

Base Score: **9.1 CRITICAL**

Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N

### QUICK INFO

**CVE Dictionary Entry:**

CVE-2022-26320

**NVD Published Date:**

03/14/2022

**NVD Last Modified:**

03/23/2022

**Source:**

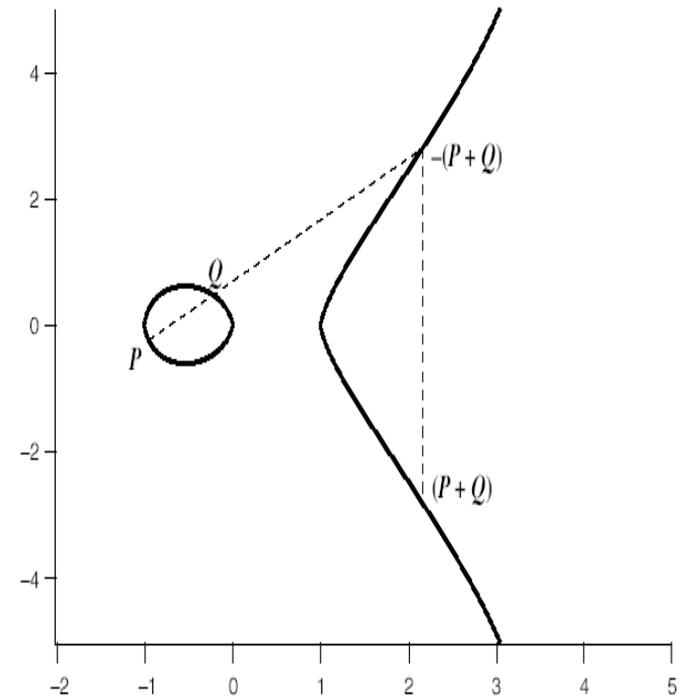
MITRE

# Elliptic Curve Cryptography (ECC)

- Traditional methods exploit the properties of arithmetic using large finite groups  $Z_n$  with  $n$  having a typical size of 1024 bits, i.e. 309 decimal digits
- The security depends on the difficulty of factorising large numbers or calculating discrete logarithms
- Using large numbers makes such algorithms computationally expensive
- In ECC,  $Z_n$  is replaced by points of an elliptic curve, making the discrete log calculation problem different and much harder compared to the discrete log in ordinary groups

# Elliptic Curve Groups

- Elliptic curves are based on simplified cubic equations, e.g.  
 $y^2 = x^3 + ax + b$   
where  $a$  and  $b$  are real numbers
- The curve shown here is defined by the equation  
 $y^2 = x^3 - x$  (i.e.,  $a = -1$  and  $b = 0$ )
- To plot such a curve, we need to compute  
 $y = \text{sqrt}(x^3 + ax + b)$
- Since the shape of the curve depends on  $a$  and  $b$ , ECs can be described as  $E(a, b)$ 
  - The above curve can be written as  $E(-1, 0)$
- In order to operate on elliptic curves, we need to introduce an operation that is equivalent to the addition as well as a “0” element

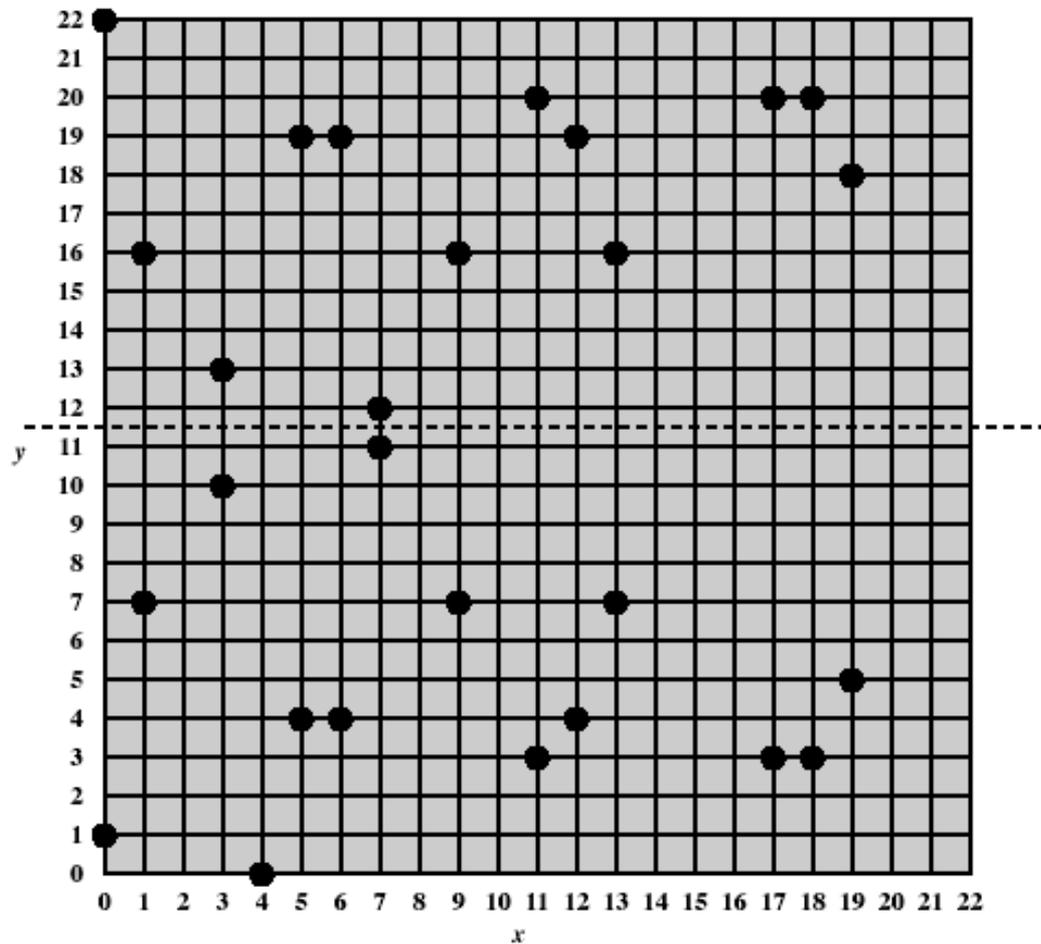


# Elliptic Curves over a Finite Field

- In order to have values  $(x, y)$  within  $\mathbb{Z}_p$ , the modulus operation is used again:  
$$y^2 \bmod p = (x^3 + ax + b) \bmod p$$
- $p$  is either a prime number or  $p = 2^m$
- We only consider pairs  $(x, y)$ , where both  $x$  and  $y$  are integer values
- Example: Table of all integer solutions for  $E_{23}(1,1)$

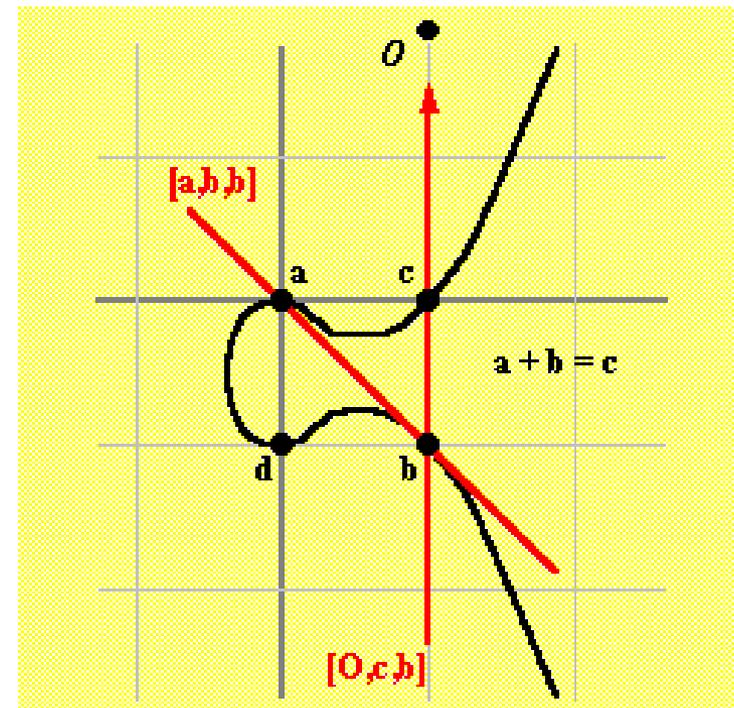
(0, 1)	(6, 4)	(12, 19)
(0, 22)	(6, 19)	(13, 7)
(1, 7)	(7, 11)	(13, 16)
(1, 16)	(7, 12)	(17, 3)
(3, 10)	(9, 7)	(17, 20)
(3, 13)	(9, 16)	(18, 3)
(4, 0)	(11, 3)	(18, 20)
(5, 4)	(11, 20)	(19, 5)
(5, 19)	(12, 4)	(19, 18)

# The Elliptic Curve $E_{23}(1,1)$



# Adding Points on an Elliptic Curve

- ECC requires the equivalent of an addition on  $E_p(A,B)$  of two points  $a$  and  $b$
- This is done (geometrically) as follows:
  - Draw a straight line through  $a$  and  $b$  to find the third intersecting point  $w$ ,
  - then draw a vertical line through  $w$  to find the intersecting point  $c$  (that's the sum)
- Every line intersects the curve three times (tangents are counted twice), e.g., the line through  $a$  and  $b$  intersects a "third" point  $b$ . We name this line  $[a,b,b]$
- $O$  is called the origin, or point at infinity
- We can say
$$a + b = c$$
$$a + a = b$$
$$a + d = b + c = O$$
$$a + O = a$$



# ECC over a Finite Field: Addition

- There's  $p$  as defined before
- Addition of two field elements  $S = (x_S, y_S)$  and  $Q = (x_Q, y_Q)$  with  $S \neq -Q$ :
  - $S + Q = R = (x_R, y_R)$
  - $x_R = (L^2 - x_S - x_Q) \bmod p$
  - $y_R = (L(x_S - x_Q) - y_S) \bmod p$
  - $L$  is either
    - $((y_Q - y_S) / (x_Q - x_S)) \bmod p$ , if  $S \neq Q$ , or
    - $((3x_S^2 + a) / (2y_S)) \bmod p$ , if  $S = Q$

# ECC over a Finite Field: Addition and Multiplication

- The addition of two elliptic points  $P$  and  $Q$  consists of a number of integer operations (mod  $q$ ):
  - ▣ 5 or 6 subtractions
  - ▣ 1 or 4 multiplications
  - ▣ 1 division
- A multiplication  $(P * Q)$  is done via consecutive additions
- A scalar multiplication  $(x * Q)$  with some scalar  $x$  is the operation of successively adding a point  $Q$  along an elliptic curve to itself  $x$  times (i.e.  $Q + Q + Q + \dots + Q$ )

# ECC Diffie-Hellman

- Similar to conventional Diffie-Hellman, but operates of finite EC field:
  - ▣ Users A & B select a suitable curve  $E_p(a, b)$
  - ▣ Users select base point (equivalent to primitive root)  
 $G = (x_1, y_1)$
  - ▣ User A & B select private keys  $n_a$  and  $n_b$
  - ▣ Users A & B compute public keys PA and PB
  - ▣ Shared keys are exchanged
  - ▣ Secret key K is computed

# ECC Diffie-Hellman Example

- Use  $E_{211}(0, -4)$  that is equivalent to  $y^2 \bmod 211 = (x^3 - 4) \bmod 211$
- Choose  $G = (2, 2)$
- User A chooses  $n_a = 121$ , so A's public key PA is:  
 $121 * G = 121 * (2, 2) = (115, 48)$
- User B chooses  $n_b = 203$ , so B's public key PB is:  
 $203 * G = 203 * (2, 2) = (130, 203)$
- The shared secret key K is  $121 * (130, 203) = 203 * (115, 48) = (169, 69)$
- Note:
  - ECC-DH (or ECDH for short) can be compromised via a MitM!
  - We still use a BIGNUM integer representation, but the range of values is significantly smaller, and operations can be executed much quicker (see next slide)

# Comparable Key Sizes for Equivalent Security

<b>Symmetric scheme (key size in bits)</b>	<b>ECC-based scheme (size of <math>p</math> in bits)</b>	<b>RSA (modulus size in bits)</b>
56	112	512
80	160	1024
112	224	2048
128	256	3072
192	384	7680
256	512	15360

# FYI: Curve25519

58

- Curve25519 is an elliptic curve offering 128 bits of security (with 256 bits key size) and designed for use with the elliptic curve Diffie–Hellman (ECDH) key agreement scheme
- It is one of the fastest ECC curves and is not covered by any known patents
- It was first released by the cryptologist Daniel J. Bernstein in 2005
- In 2013, interest began to increase considerably when it was discovered that the NSA had potentially implemented a backdoor into the most common EC encryption method
  - ▣ i.e. the P-256 curve based Dual\_EC\_DRBG algorithm
- Today it is the de facto alternative to P-256
- Its reference implementation is public domain software

# The Double-Ratchet Algorithm<sup>[1]</sup>

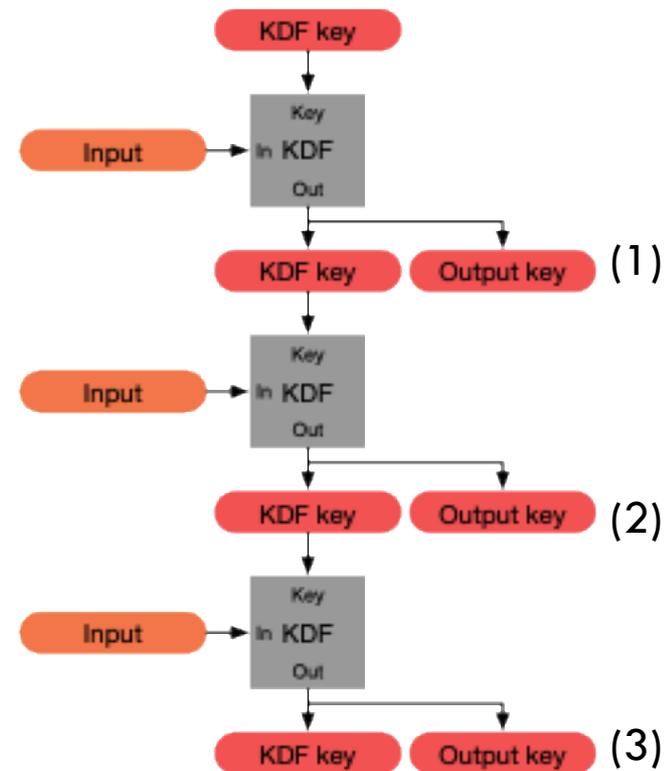
59

- ❑ The **Double Ratchet algorithm** is a cryptographic protocol used by two parties to exchange encrypted messages
  - ❑ Messages are encrypted using (fast) symmetric key algorithms (e.g., AES)
  - ❑ Every message that is exchanged in either direction is encrypted using a different private key
- ❑ The algorithm is implemented in the Signal protocol, which in turn is used in secure messaging apps such as the Signal app and WhatsApp
- ❑ It ensures forward secrecy and post-compromise security, making conversations secure even if previous keys are compromised
- ❑ (Perfect) forward secrecy and post-compromise security are properties of secure communication protocols
  - ❑ **Forward security** ensures the confidentiality of past sessions even if long-term keys are compromised
  - ❑ **Post-compromise security** ensures the security of future communications even after an initial compromise

# Key Derivation Function (KDF) and KDF Chains

60

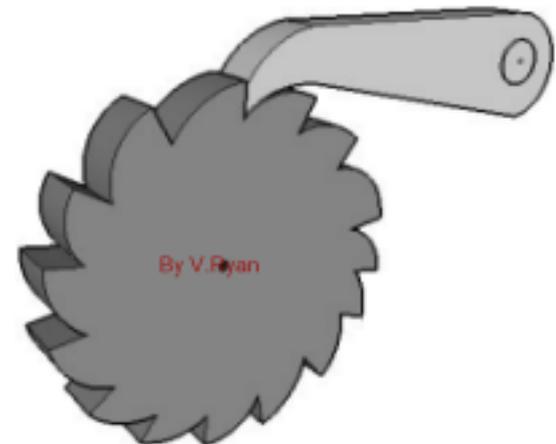
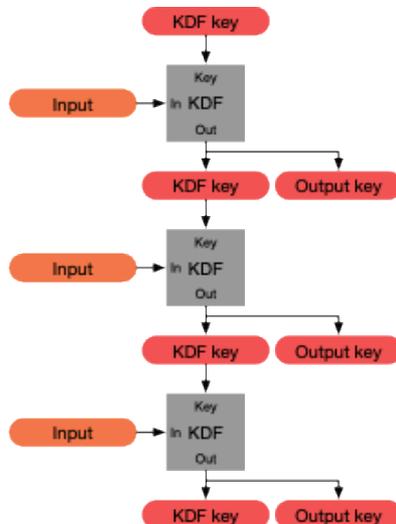
- A **KDF** is a cryptographic function that
  - ▣ is used to create a new secret key for each message
  - ▣ takes a secret **(KDF key)** and some **(Input)** data, and returns an output
  - ▣ looks like a “one-way” function (i.e., a hash function)
- In a **KDF chain** some of the KDF output is used as an **(Output key)** and some is used to make a new **(KDF key)**
- If two endpoints agree on the same initial **(KDF key)** and the same **(Input)**, they create the same sequence of output keys, and can exchange messages securely
- A KDF chain guarantees forward security, but not automatically post-compromise security
  - ▣ Consider output key (2) being recovered by an attacker:
  - ▣ The attacker cannot calculate key (1)
  - ▣ The attacker is only prevented from calculating Output key (3), if Input is a secret shared by both endpoints



# KDF Chains

61

- A KDF chain is like a ratchet, which only goes in one direction
  - ▣ each step provides a different output (KDF key | | Output key)
- Both Alice and Bob have both a “send” and “receive” ratchet each
- Alice’s “send” and Bob’s “receive” ratchet are initialised using the same initial KDF and the same *Input* key (and visa versa)
- Every time a message is to be sent by either side, it is encrypted first using a new encryption key (*Output* key) that is generated by invoking the KDF (i.e., the “sender” ratchet)
- Similarly, every time the receiver receives a new message it calculates the (same) key for message decryption by invoking the KDF (i.e., the “receiver” ratchet)

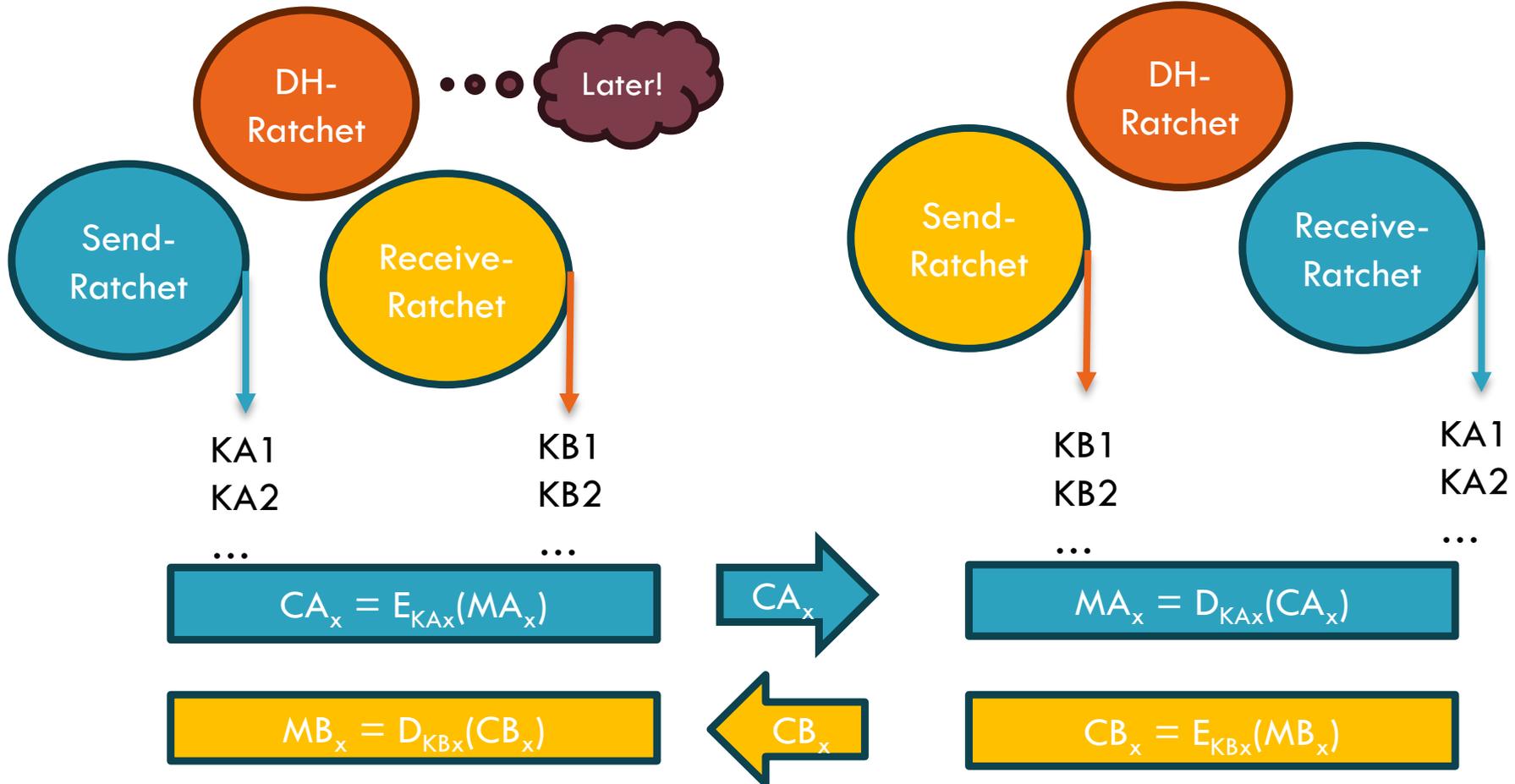


# Sender and Receiver Ratchet

62

Alice

Bob



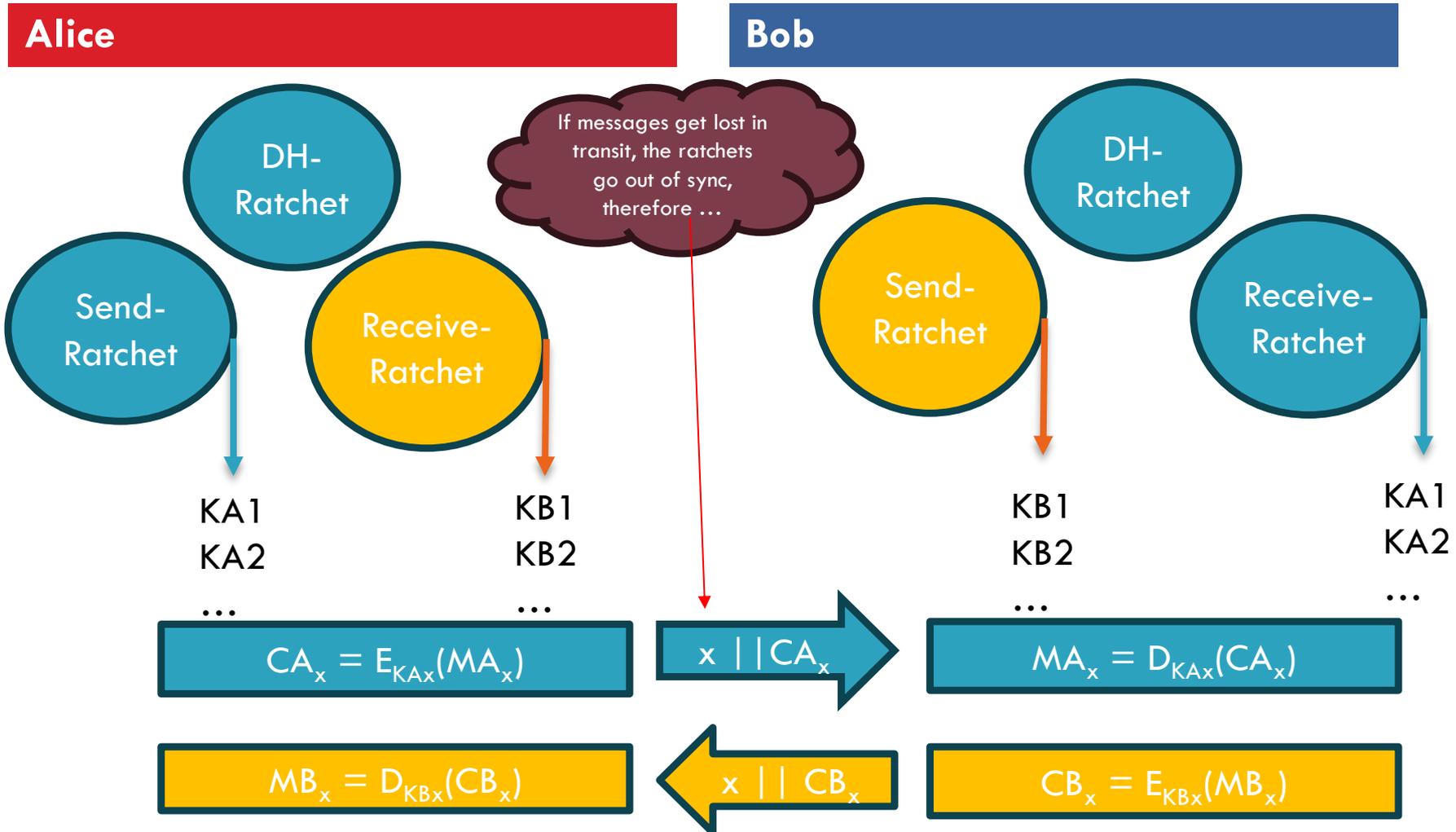
# Explanations

63

- $K\{A | B\}_x$  is a secret key used by A or B for encoding and decoding a message (e.g.,  $KA_5$  or  $KB_7$ )
  - ▣ x is simply an incremented index value (i.e., 1, 2, 3,...)
- $M\{A | B\}_x$  are (indexed) plaintext messages generated by A or B (e.g.,  $MA_5$  or  $MB_7$ )
- $C\{A | B\}_x$  is the corresponding ciphertext
  - ▣ E.g.,  $MA_3 \leftrightarrow CA_3$
- E() and D() are corresponding encryption and decryption functions that use a key  $KA_x$  (e.g.,  $D_{KA_5}(CA_5)$ )

# Synchronising Sender and Receiver Ratchets to compensate for lost Messages

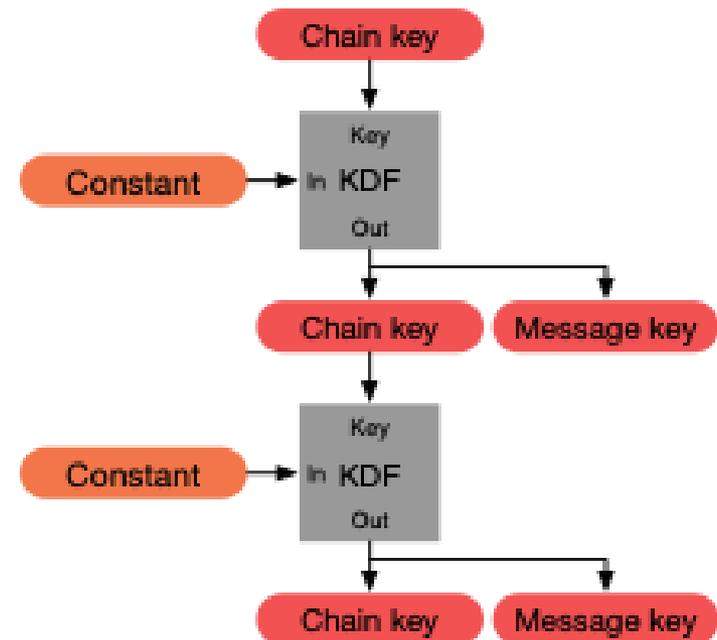
64



# Symmetric Key Ratchet

65

- “Send” and “receive” ratchets are also called the **symmetric-key ratchets**
- Since every message sent is encrypted with a unique *Message key* (see diagram), the receiver may have to buffer generated (decryption) keys to deal with packets received out-of-order
- Here KDF keys are called **(Chain keys)**
- *The sequence of generated chain keys is called a **sending chain / receiving chain***
- Here KDF chains use a (secret) **(Constant)** as a 2<sup>nd</sup> input to provide post-compromise security



# The Diffie-Hellman Ratchet

66

- As Alice and Bob exchange messages, they also exchange new Diffie-Hellman public keys to generate shared secret keys
- These secret keys become the *input* to another KDF chain, the **root chain**
  - ▣ This is called the **Diffie-Hellman ratchet**
- The output keys from the root chain provide for new KDF chain keys for the sending and receiving ratchet
- The complete construct is called a Double Ratchet, consisting of the symmetric key ratchets and the DH ratchet, which require KDF keys for three chains:
  - ▣ a **sending chain and a receiving chain** (linked to the “send” and “receive” ratchets)
    - With Alice’s sending chain matches Bob’s receiving chain, and vice versa
  - ▣ a **root chain** (linked to the DH-ratchet)

# The Diffie-Hellman Ratchet

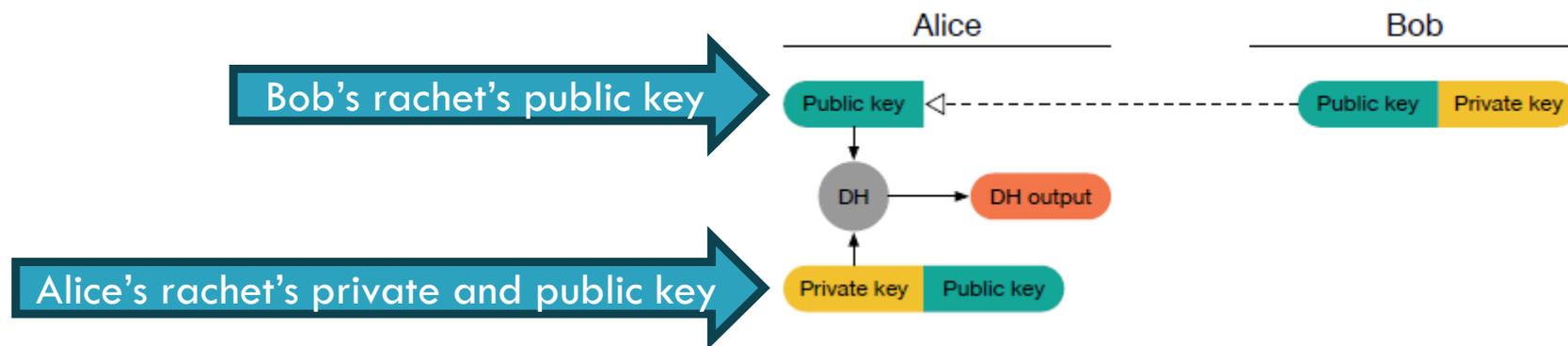
67

- To implement the DH ratchet, each party generates a DH key pair (a Diffie-Hellman public key and private key) which becomes their current ratchet key pair
- Every message from either party begins with a header which contains the sender's current DH-ratchet public key
- When a new ratchet public key is received from the other party, a DH ratchet step is performed which replaces the local party's current ratchet key pair with a new key pair
- This results in a “ping-pong” behavior as the parties take turns replacing ratchet key pairs

# Stepping through the DH-Ratchet: Step 1

68

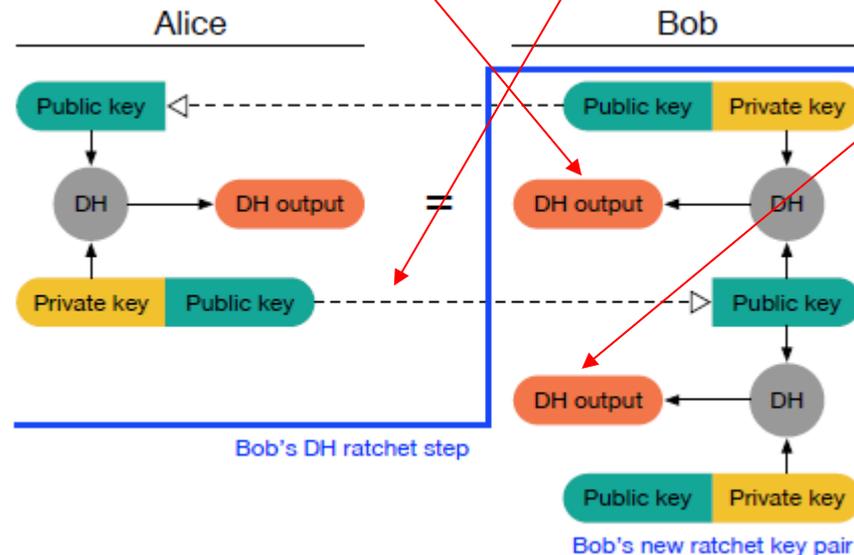
- Alice receives Bob's ratchet's public key
  - ▣ Alice's ratchet's public key isn't yet known to Bob
- As part of the initialisation Alice performs a DH calculation using her ratchet's **(Private key)** and Bob's ratchet's **(Public key)**



# Stepping through the DH-Ratchet: Step 2

69

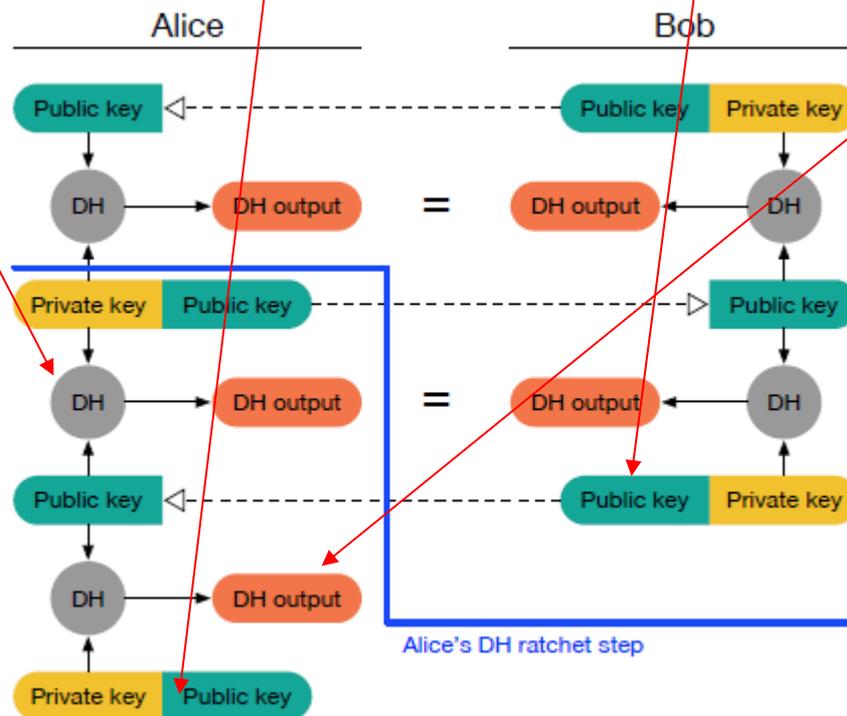
- Alice's initial messages advertise her ratchet's public key
- Once Bob receives one of these messages, he performs a DH ratchet step (consisting of two (DH) steps, i.e., Diffie-Hellman key exchange calculations):
  - ▣ He calculates the **DH output** between Alice's ratchet's public key and his ratchet's **(Private key)**, which equals Alice's initial (DH output)
  - ▣ Bob then calculates a **new ratchet key pair** and calculates a new **DH output**:



# Stepping through the DH-Ratchet: Step 3

70

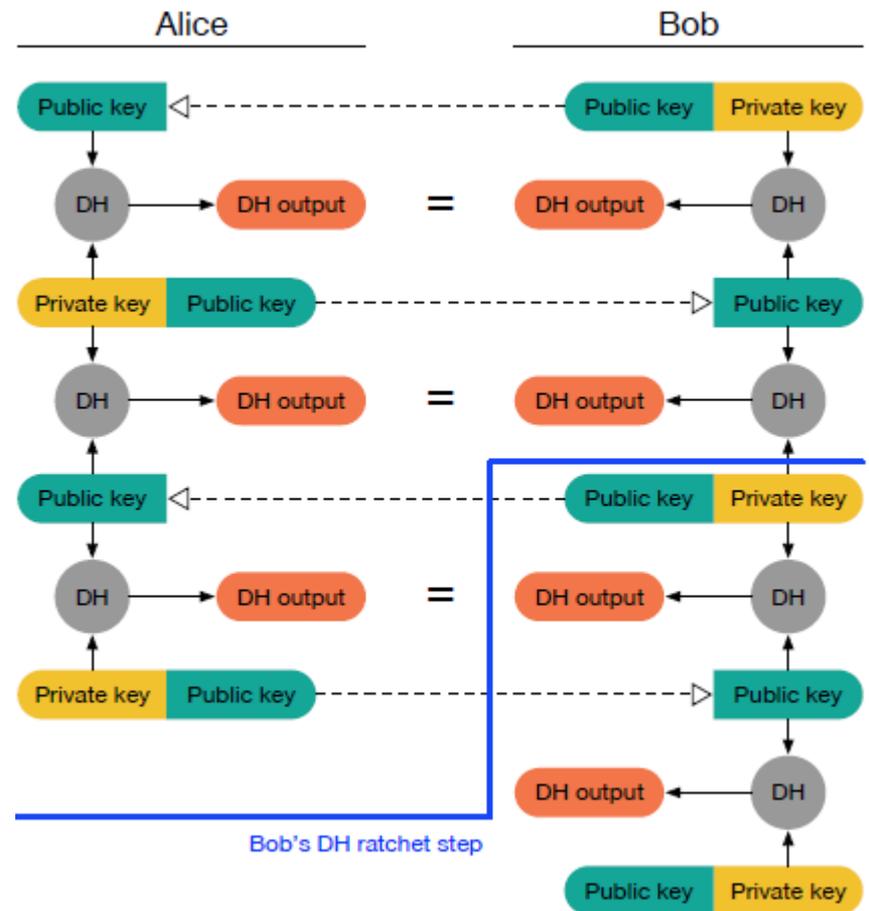
- Messages sent by Bob advertising his new Public key are received by Alice, who does a similar step comprising:
  - ▣ A (DH) operation using her current Private key and bob's new Public key will result in a DH output identical to the one calculated by Bob
  - ▣ She creates a new Private / Public key and calculates a new DH output :



# Stepping through the Diffie-Hellman Ratchet: Step 4+

71

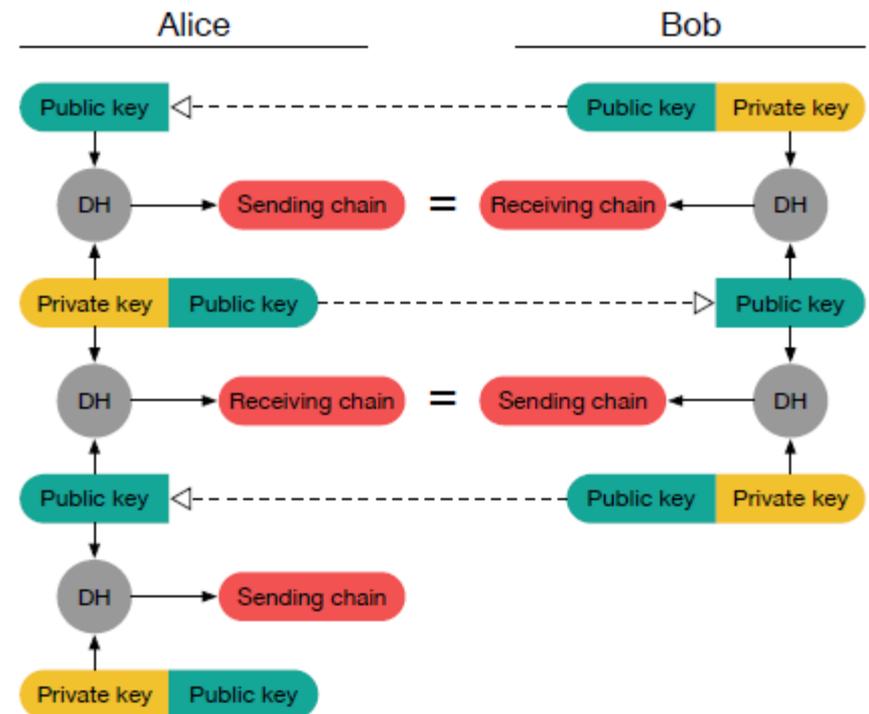
- Messages sent by Alice advertise her new public key
- Bob receives one of these messages and perform a second DH ratchet step, and so on



# Deriving Sending and Receiving Chains Keys

72

- The **DH outputs** generated during each DH ratchet step are used to derive new sending and receiving chain keys for Alice's and Bob's symmetric key ratchets
- The DH outputs are not used directly, but go through a DH ratchet first (see next slide)



# Deriving Sending and Receiving Chains

73

- This diagram shows the complete process from Alice's perspective:
  - The **Root Key** is a shared secret with Bob, determined via (ECC-) DH at the beginning of the protocol / session
  - The DH output (as calculated in previous slides), together with the Root key, is processed by the DH ratchet in the centre of the diagram to create a *Receiving chain key*
  - Bob's public key, together with Alice's *Private key* of her 2<sup>nd</sup> generated keypair is used for another KDF invocation that generates the *Sending chain key* and a new *Root key*



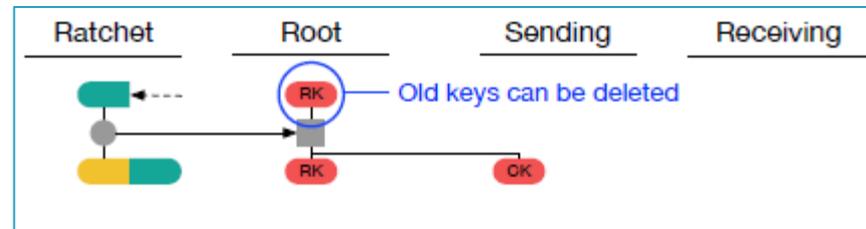
# A Double Ratchet Walk-Through

74

- The following example shows a double ratchet walk-through from Alice's perspective, including only messages she is receiving from Bob

- Step 1:

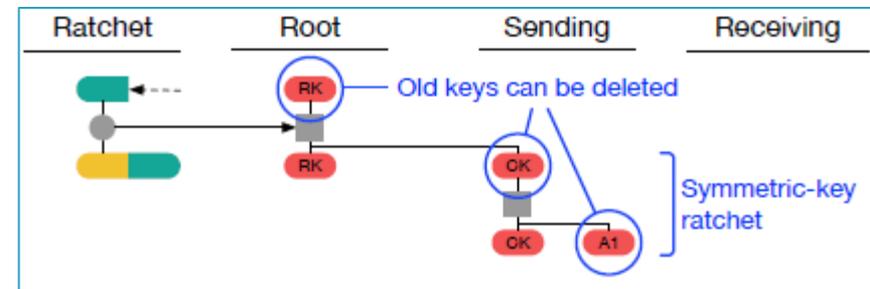
- Alice receives Bob's public key and generates a new root key (RK) and sending chain key (CK)



- Step 2:

- When Alice sends her first message, she applies a symmetric-key ratchet step to her sending chain key (CK), resulting in a

- message key (A1)
- new chain key (CK) (ignore for now)

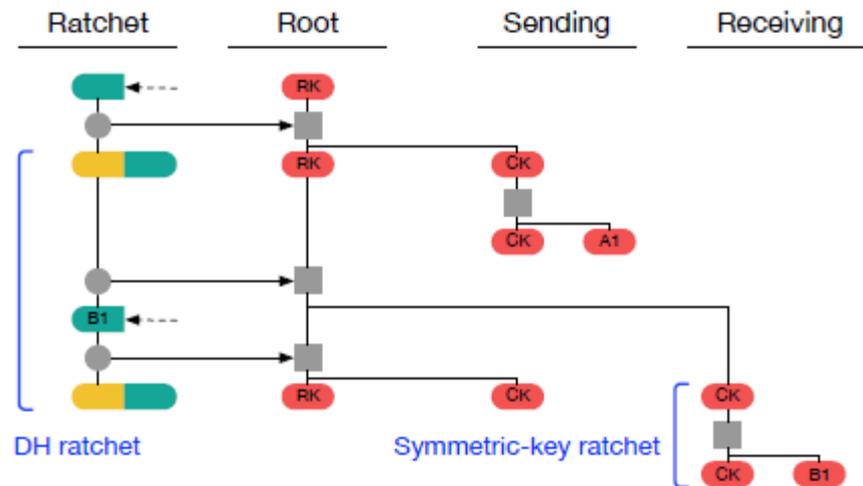


# A Double Ratchet Walk-Through

75

## □ Step 3:

- Alice receives a response from Bob; it contains his new DH ratchet public key **B1**
- Alice applies a DH ratchet step to derive a new receiving chain key **(CK)** ...
  - She then applies a symmetric-key ratchet step on **(CK)** to get the message key **(B1)** for the received message, as well as a new chain key **(CK)**
- ... and to derive a new sending chain key **(CK)**
  - In the next step (shown on the next slide), she applies the ratchet on **(CK)** as well to create the sending key **(A2)**

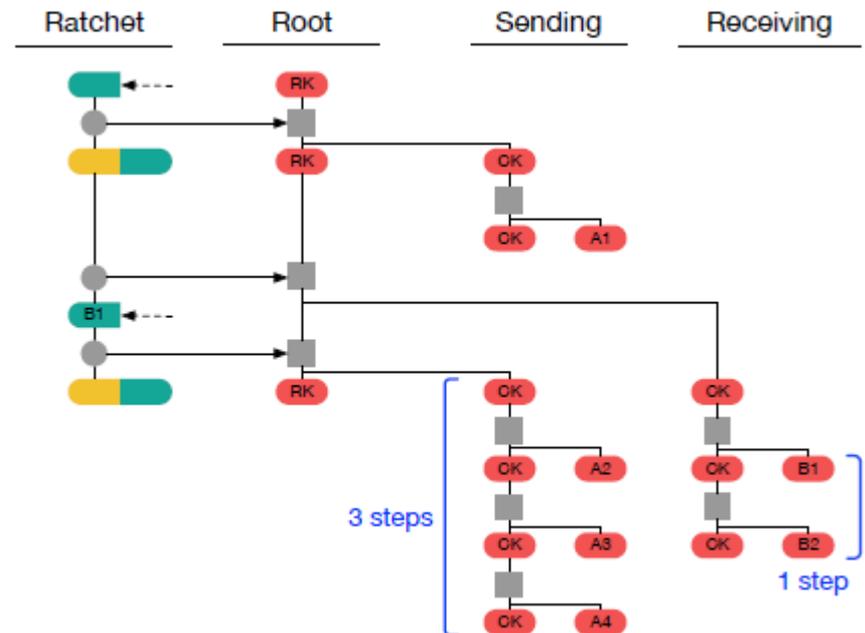


# A Double Ratchet Walk-Through

76

## □ Step 4:

- Here Alice next sends a message using (A2), and applies two more ratchet steps to create sending message keys (A3) and (A4) for 2 additional messages
  - Note that the DH-ratchet wasn't invoked to create new chain keys, as seen before, i.e. Alice sent a sequence of messages to Bob without prior receiving his new public key
- Alice receives a message encrypted with (B2)
- Since Alice didn't receive a new public key from Bob, she simply applies the receiving key ratchet again, to derive (B2)

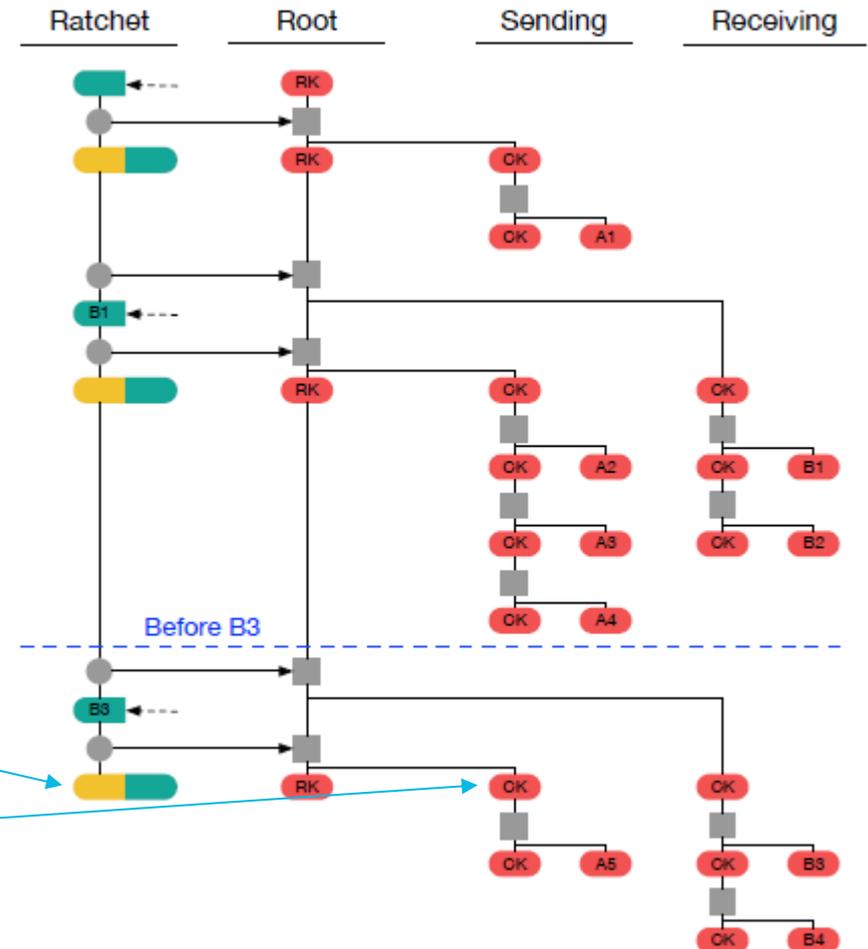


# A Double Ratchet Walk-Through

77

## Step 5:

- Alice then receives Bob's new public key (B3), as well as messages encrypted with (B3) and (B4)
- She generates these keys, by
  - Applying the DH ratchet and creating a new receiving chain key (CK)
  - Executing the receiving key ratchet twice to generate (B3) and (B4)
- Alice also generates a new sending message key (A5), by
  - calculating a new private key
  - Applying the DH ratchet
  - Creating a new sending chain key
  - Executing its ratchet once to create (A5)



# Summary: Keys and Key Exchanges in the Double Ratchet Protocol

78

- Initial Key Exchange:
  - Two parties (Alice and Bob) perform an initial key exchange using (a MitM-resilient variation of) ECDH to establish the *Root key*;
  - The Constants in the symmetric key ratchets are derived from the Root key
- Symmetric Key Ratcheting:
  - Every time a message is sent / received, a new symmetric encryption key is provided by the “send” ratchet and the “receive” ratchet
  - This process is known as “ratcheting forward” and ensures that each message has a unique encryption key
- Asymmetric Key Ratcheting:
  - Normally, after each message exchange, both parties generate a new root key by doing a DH key exchange
  - However, if the message receiver is offline, the sender can still use symmetric key ratcheting to create a new message key for each message

# References

79

[1] The Double Ratchet Algorithm; Trevor Perrin and Moxie Marlinspike

CT437

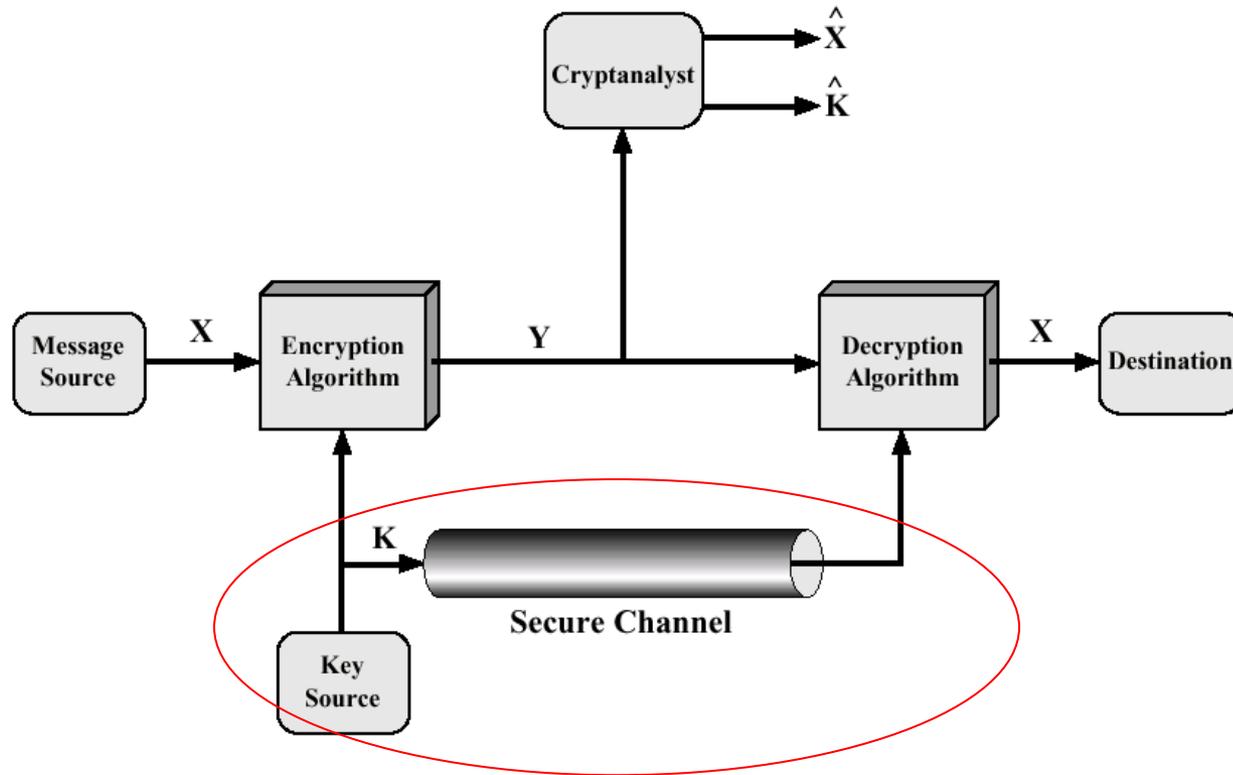
COMPUTER SECURITY AND FORENSIC COMPUTING

MANAGEMENT / DISTRIBUTION OF SYMMETRIC AND  
PUBLIC KEYS

Dr. Michael Schukat

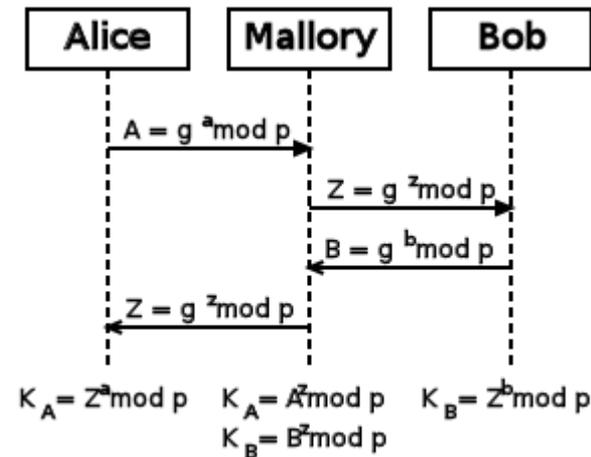
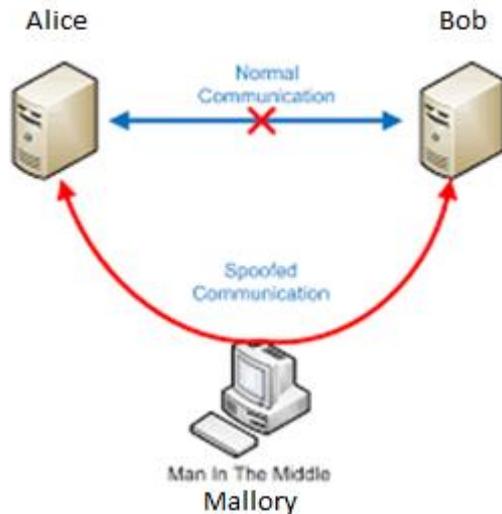


# Recap: Model of a Conventional Cryptosystem



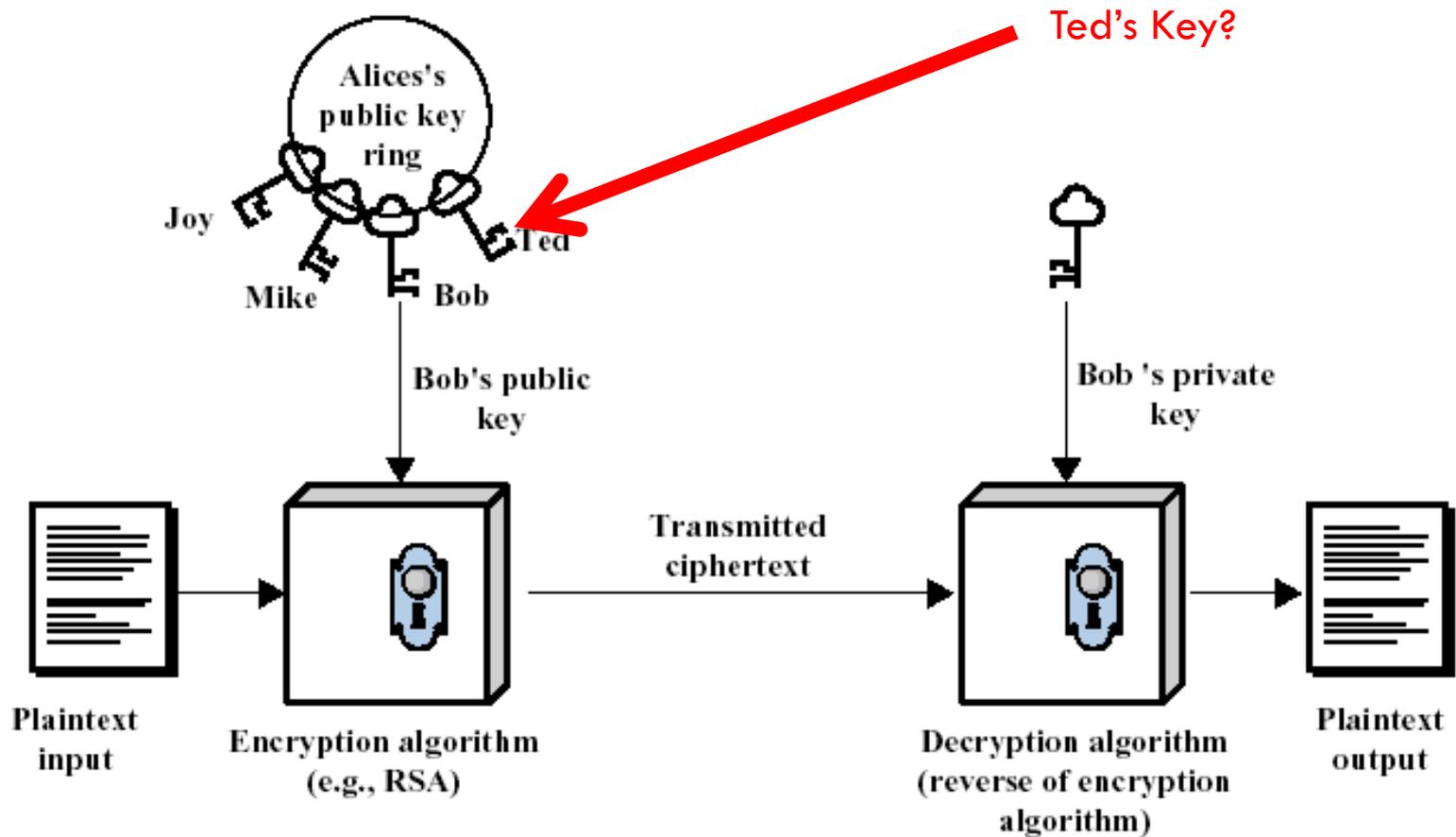
$$Y = E_K(X), X = E_K^{-1}(Y)$$

# Recap: DH and Man-in-the-Middle (MitM) Attacks



- ❑ Mallory is a MitM attacker and performs message interception and message fabrication
- ❑ Mallory establishes two individual (secure) connections with Alice and Bob
- ❑ Both Alice and Bob are unaware of Mallory's existence (as there is no authentication)
- ❑ **DH alone is not sufficient for secure key distribution!**

# Recap: Public-Key Encryption



# Key Issues that need to be addressed

1. Symmetric key encryption
  - ▣ Key distribution mechanism?
  - ▣ Key management (key renewal / generation)?
  - ▣ Key authentication?
2. Public key encryption
  1. Public key distribution / management?
  2. Public key authentication, i.e. validation of owner?

# Terminology

6

- **Key rotation** is the general term for creating a new key and starting to encrypt data with it, while retiring the old key, hence the rotation
  - ▣ Time-Based Key Rotation
    - E.g., every week
  - ▣ Usage-Based Key Rotation
    - E.g., after using it to process x Gigabyte of data
  - ▣ Incident-Triggered Key Rotation
    - Change key if it was compromised
- **Re-keying** involves changing cryptographic keys in an on-going communication channel (e.g., TLS → later)
- **Re-encryption** refers to the process of encrypting previously encrypted data using a new key

# Key Management Lifecycle

7

- **Generation:** Generating strong cryptographic keys using a cryptographically secure random number generator (as seen before)
- **Distribution:** Safely transmit them using encrypted channels / protocols, to authorised parties without risking unintended exposure
  - **Key wrapping** is a common approach here, i.e. encrypt the new key using the old key before circulation
  - Possibly **DH** if hardened against MitM attacks
- **Storage:** Utilize key management systems (KMS) to encrypt, store and manage cryptographic keys to protect them from theft or unauthorised access
  - See next slide
- **Usage:** Utilise keys for encryption / decryption / authentication
- **Rotation:** Replace cryptographic keys regularly or according to a policy to limit their exposure and minimize any data exposure impact from potential key compromise
- **Destruction:** Safely delete keys once they are no longer needed to prevent their recovery or misuse

# Types of KMS and cryptographic Key Stores

8

- Hardware Security Modules (HSMs):
  - ▣ These are physical devices that provide secure key storage and cryptographic operations, e.g. USB HSM
- Cloud-based Key Management Services:
  - ▣ E.g., AWS Key Management Service, Azure Key Vault, and Google Cloud KMS
- Software-based Key Stores:
  - ▣ E.g., OpenSSL, Java KeyStore (JKS), and Microsoft's Cryptographic API
  - ▣ They are used for storing keys in software applications
- Hardware-based Key Stores:
  - ▣ Devices like TPM (Trusted Platform Module) and smart cards can securely store cryptographic keys and perform cryptographic operations





# Symmetric Key Cryptosystems

# Key Distribution Case Study

## □ Problem

- ▣ Two parties PA and PB want to securely communicate over a public network using symmetric key encryption
- ▣ How can the key distribution be achieved?

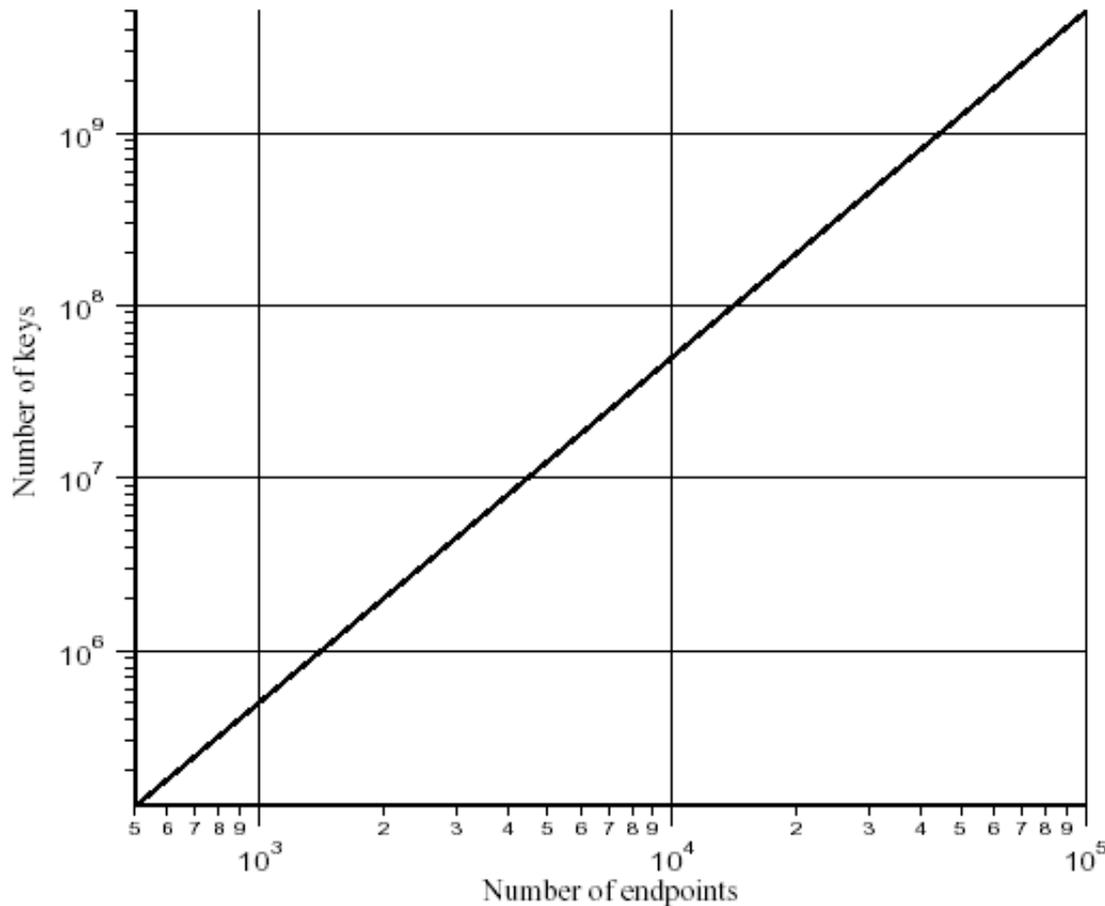
## □ Simple solutions

1. A key is selected by PA and physically delivered to PB
2. Some independent authority PC selects a key and physically delivers it to PA and PB

## □ Drawbacks of both solutions:

- ▣ Manual delivery of keys → this is tedious and is cumbersome
- ▣ The solution does not scale, as for N parties (e.g. endpoints in a computer network)  
 $N * (N - 1) / 2$  unique keys are required

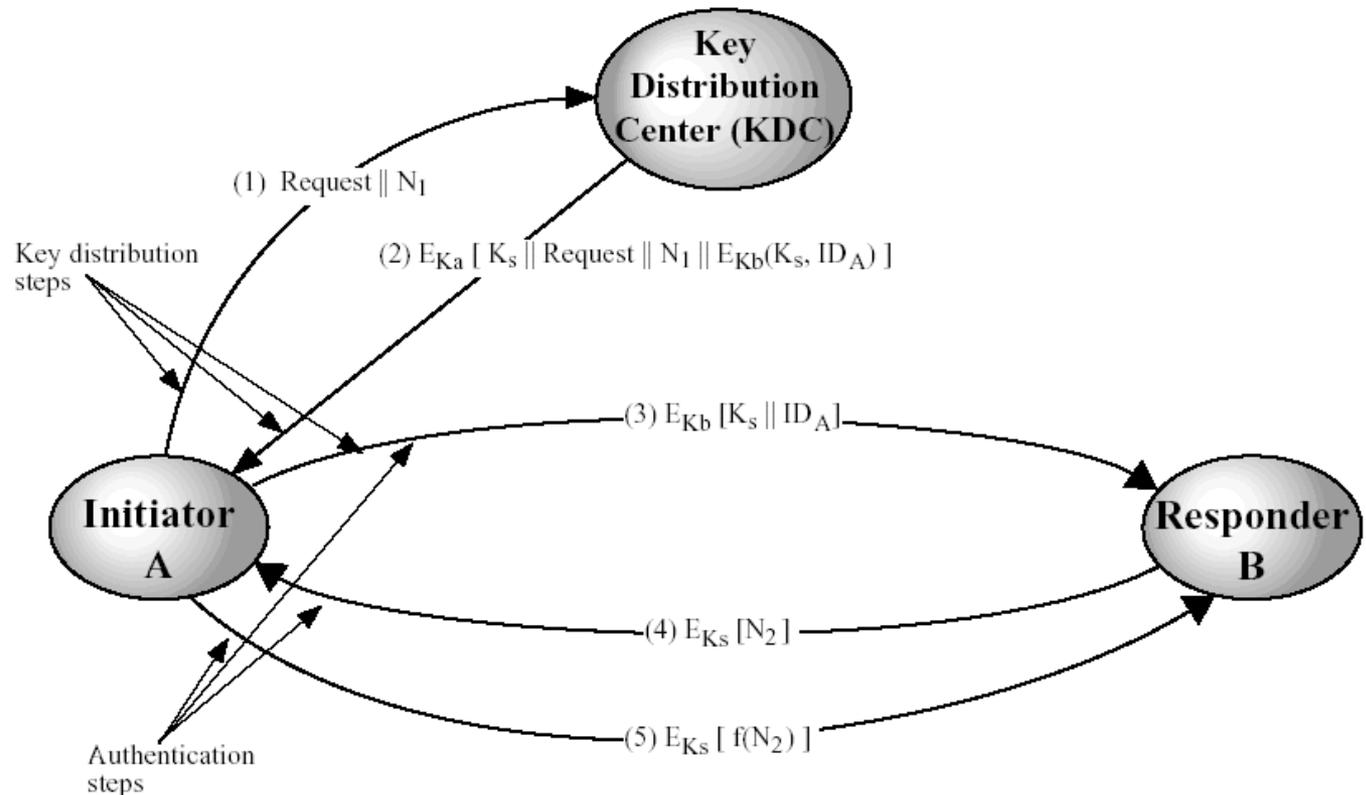
# Number of (unique) Keys versus Number of Endpoints



# Key Distribution using a KDC

- Solution 3 overview
  - ▣ PA and PB can rely on a secure (encrypted) connection to a **key distribution centre (KDC)**
  - ▣ The KDC delivers a key via the encrypted links to A and B on demand
- Details:
  - ▣ Each endpoint and the KDC already share a unique **master key**
  - ▣ This key is used to securely exchange messages between both ( $E_{K_x}$  in the next slide)
  - ▣ For N hosts, N master keys are required
  - ▣ Two hosts communicate securely with each other, by using a secure session key  $K_s$ , which is provided by the KDC

# KDCs and the Needham-Schroeder Protocol



1.  $A \rightarrow KDC: ID_A \parallel ID_B \parallel N_1$
2.  $KDC \rightarrow A: E(K_a, [K_s \parallel ID_B \parallel N_1 \parallel E(K_b, [K_s \parallel ID_A])])$
3.  $A \rightarrow B: E(K_b, [K_s \parallel ID_A])$
4.  $B \rightarrow A: E(K_s, N_2)$
5.  $A \rightarrow B: E(K_s, f(N_2))$

# The Needham–Schroeder Protocol explained

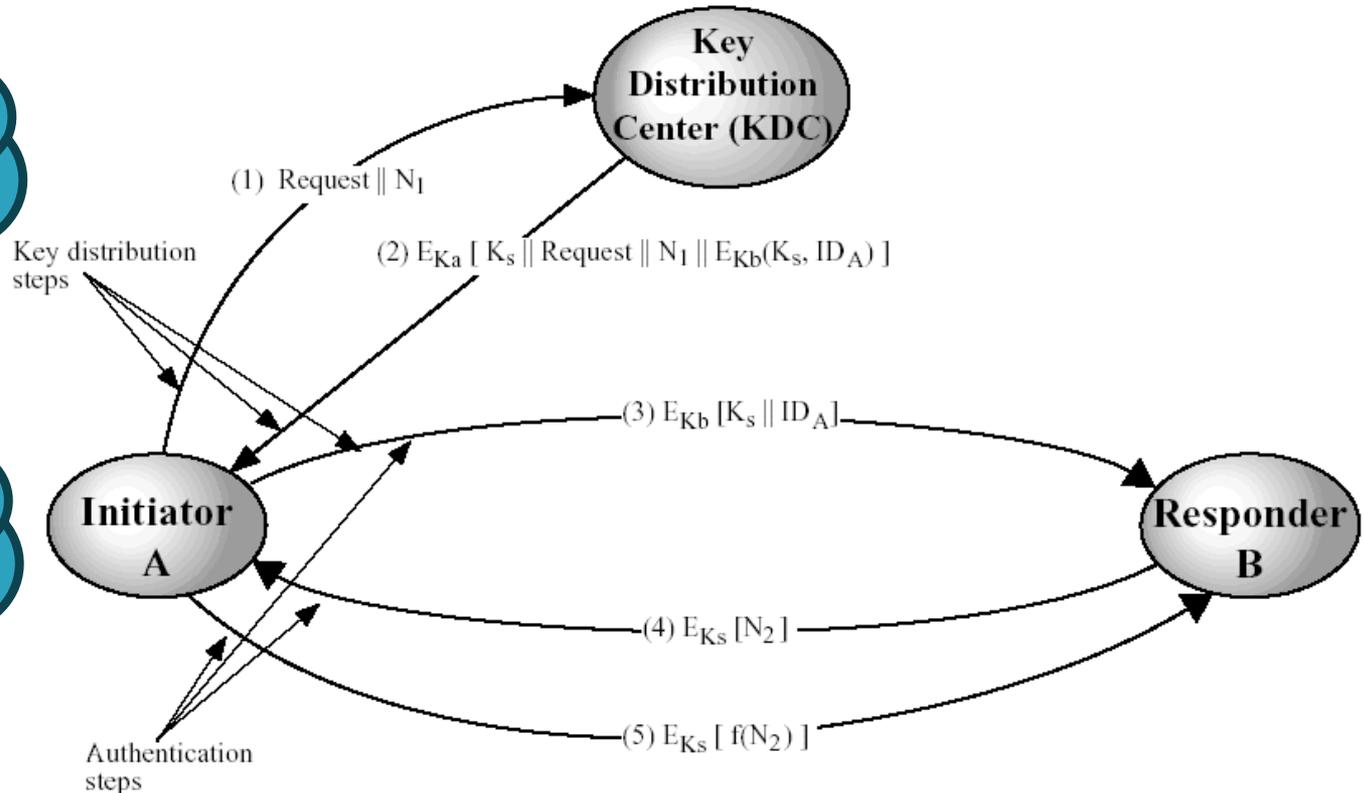
14

- Initiator A (IA) and responder B (RB) share a unique master key each with the KDC ( $K_{KA}$  and  $K_{KB}$ )
  1. IA issues a message to the KDC for a session key to be shared with RB; it includes:
    - The Request, containing the identity of IA and RB (e.g., their network addresses)
    - A unique nonce  $N_1$  for this transaction
  2. The KDC responds with a message encrypted using  $K_{KA}$  that contains:
    - The session key  $K_S$
    - The original Request and  $N_1$
    - A message for the responder RB that is encrypted using  $K_{KB}$  and that includes:
      - The session key  $K_S$
      - The identity of A,  $ID_A$  (e.g., its network address)
  3. IA decrypts / validates the response and sends only the above message to RB
  4. RB:
    - Decodes the message using  $K_{KB}$  and validates IA to be the message sender
    - Sends a new nonce  $N_2$  to IA, encrypted using  $K_S$
  5. IA:
    - Decrypts the message using its copy of  $K_S$
    - Processes the nonce in an agreed fashion (e.g.,  $N_2 = N_2 + 1$ )
    - Encrypts  $N_2$  and sends it to RB
- RB validates the content of the message and by doing so authenticates IA

# KDCs and the Needham-Schroeder Protocol

How can the KDC be compromised?

How can the protocol be compromised?



1.  $A \rightarrow KDC: ID_A \parallel ID_B \parallel N_1$
2.  $KDC \rightarrow A: E(K_a, [K_s \parallel ID_B \parallel N_1 \parallel E(K_b, [K_s \parallel ID_A])])$
3.  $A \rightarrow B: E(K_b, [K_s \parallel ID_A])$
4.  $B \rightarrow A: E(K_s, N_2)$
5.  $A \rightarrow B: E(K_s, f(N_2))$

# Possible Attacks on the Needham–Schroeder Protocol

16

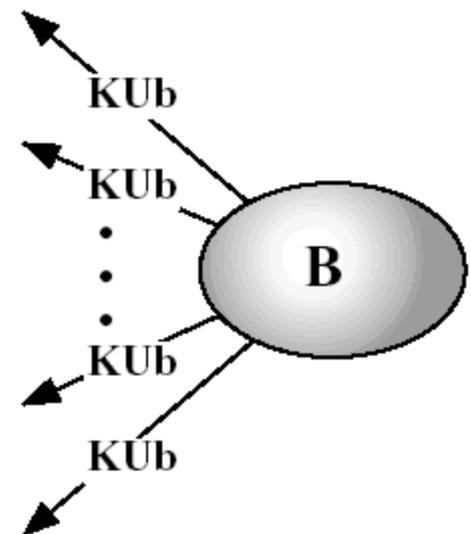
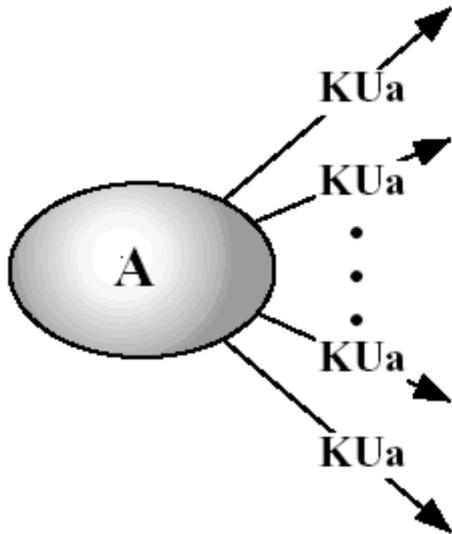
- Assume an attacker is positioned between IA and KDC
- The MitM intercepts (1), identifies IA and RB, and intercepts (2)
- The protocol is completed as before, and  $K_S$  is used by IA and RB
- At some stage in the future  $K_{KA}$  is compromised
- **The MitM can now**
  - ▣ **decode (2)**
  - ▣ **impersonate IA (by using  $ID_A$ )**
  - ▣ **resend  $X = EKB(K_S, ID_A)$  to RB (3),**
  - ▣ **complete the protocol**
- **RB believes it is talking to IA**
- **Solution:**
  - ▣ X must be complemented with a timestamp (when  $K_S$  was created) and / or  $K_S$  validity period, so RB can validate that KS is not stale (and must not be used any more)
  - ▣ all entities must be time-synchronised ( $\rightarrow$  NTP / PTP)



# Public Key Cryptosystems

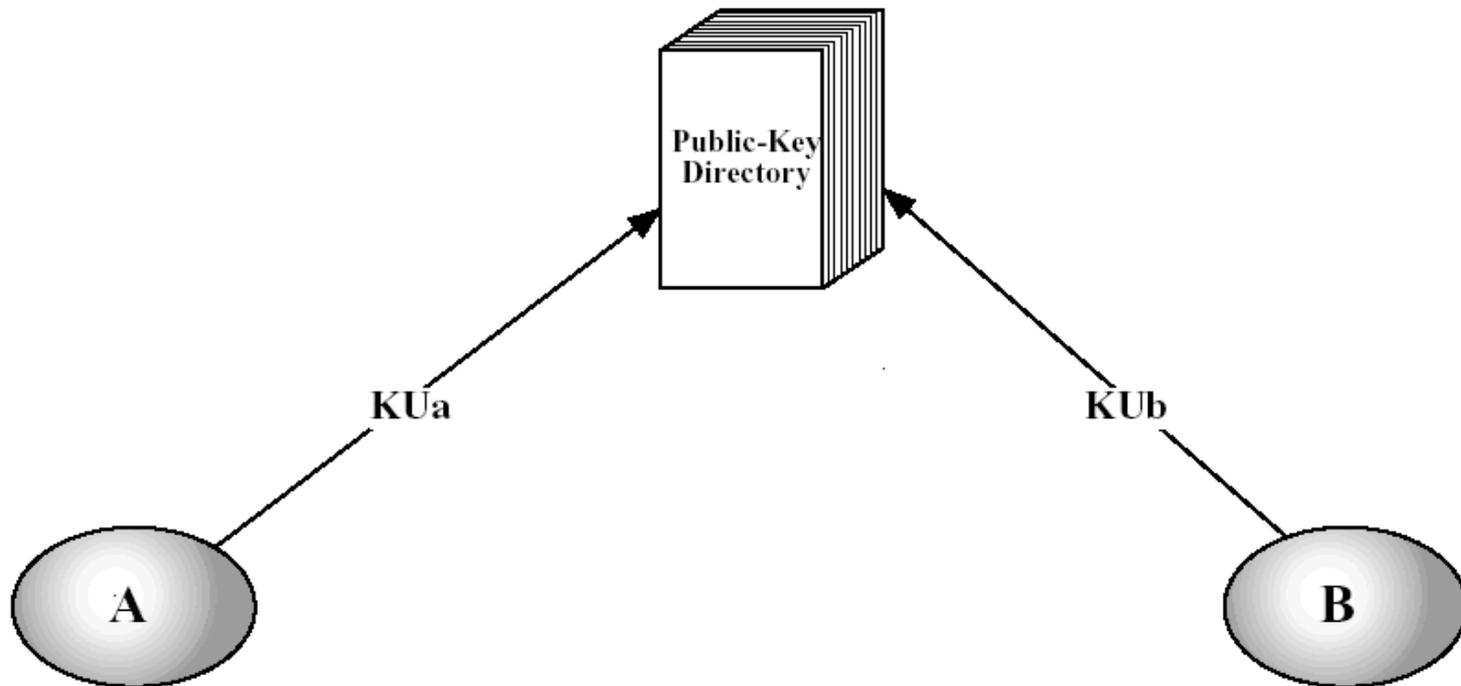
# Key Management via uncontrolled Public-Key Distribution

- Simplistic approach, but easy to forge, e.g., anyone could pretend to be user A

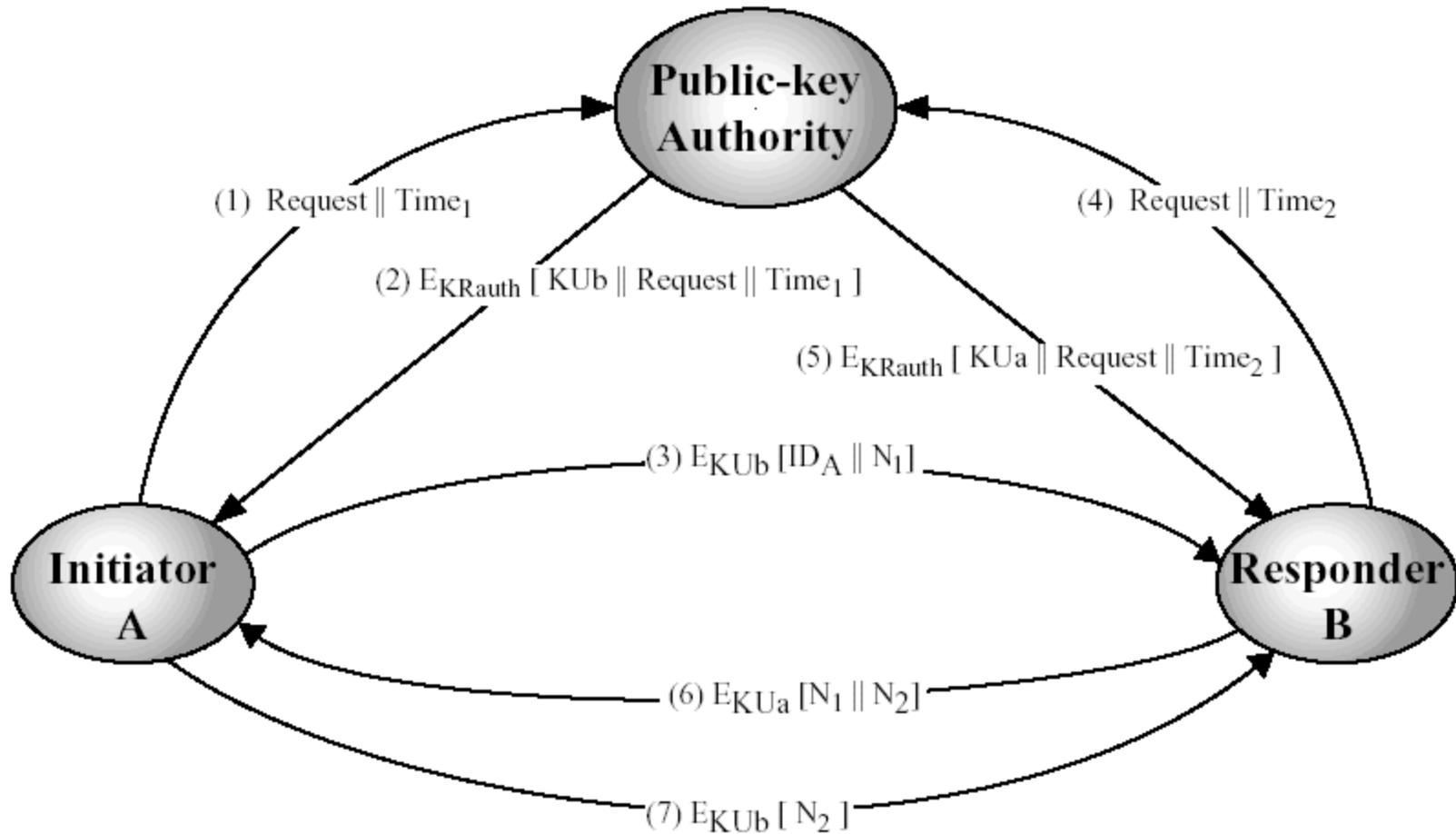


# Key Management via Public-Key Directory

- The directory is just a public platform where everybody can upload their public key
- Similar issues as before



# Key Management using a Public-Key Authority

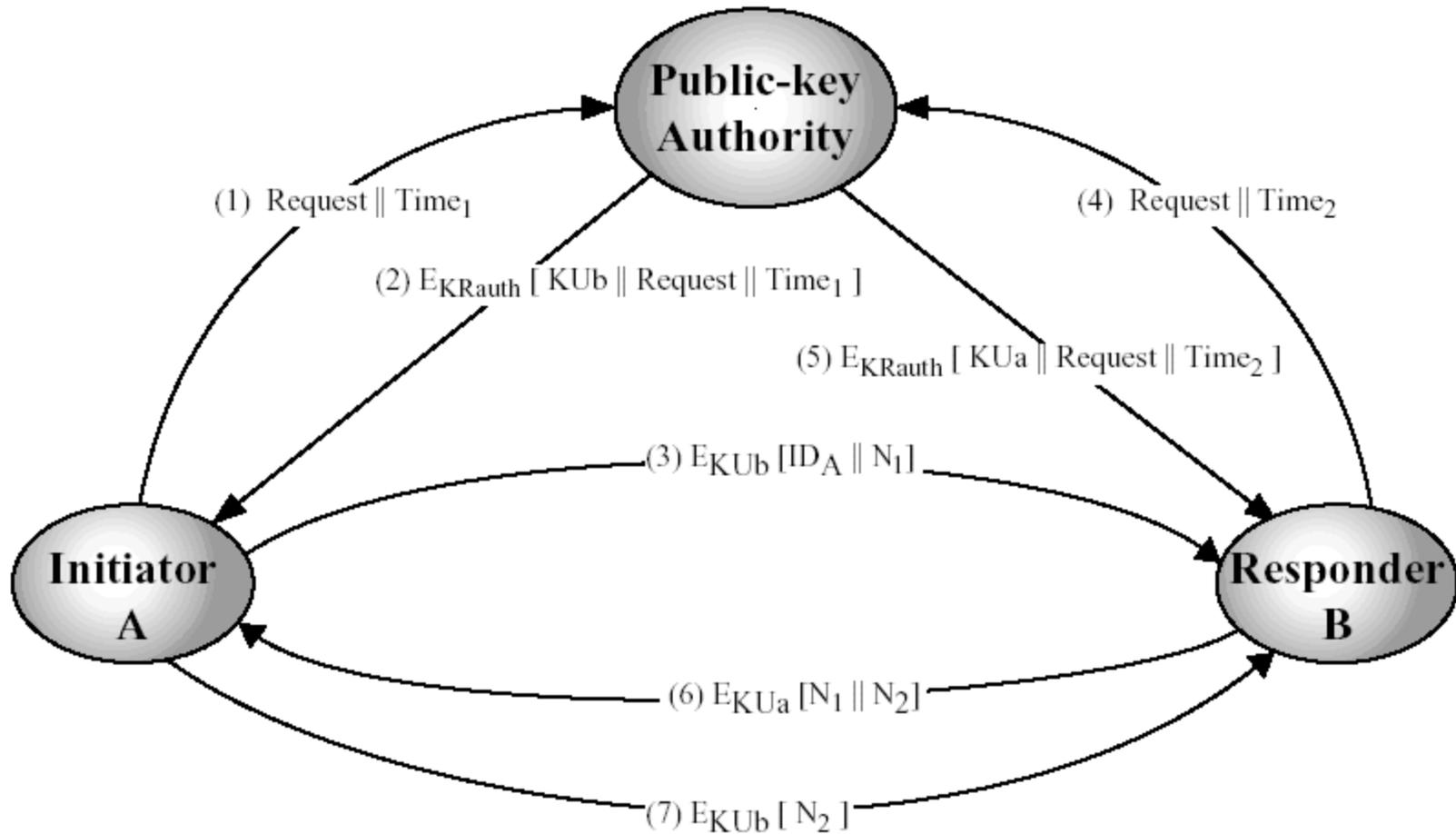


# Key Management using a Public-Key Authority

21

- Based on the Needham–Schroeder Protocol, but with some improvements
- The public-key authority (PKA) has a public / private key pair with:
  - ▣ Private key  $K_{RAuth}$
  - ▣ Public key  $K_{UAuth}$  being shared with all clients
- Initiator A (IA) and responder B (RB) have a public / private key pair each
  - ▣  $K_{UA}$  and  $K_{RA}$
  - ▣  $K_{UB}$  and  $K_{RB}$
- $K_{UA}$  and  $K_{UB}$  are managed by the PKA
- IA requests for  $K_{UB}$  in order to setup a secure connection with RB

# Key Management using a Public-Key Authority



# The Protocol explained

23

1. IA issues a message to the PKA to get  $K_{UB}$ ; it includes:
  - ▣ The Request, containing the identity of IA and RB (e.g., their network addresses)
  - ▣ The timestamp  $Time_1$  of this transaction
2. The PKA responds with a message authenticated using  $K_{Rauth}$  that contains
  - ▣ RB's public key  $K_{UB}$
  - ▣ The original Request and  $Time_1$
3. IA:
  - ▣ Validates the authenticity of the response by decoding the message using  $K_{rauth}$  and validating Request and  $Time_1$ ; IA extracts  $K_{UB}$
  - ▣ Use this key to encrypt a message containing its (network) id  $ID_A$  and a nonce  $N_1$
  - ▣ IA sends the message to RB

# The Protocol explained

24

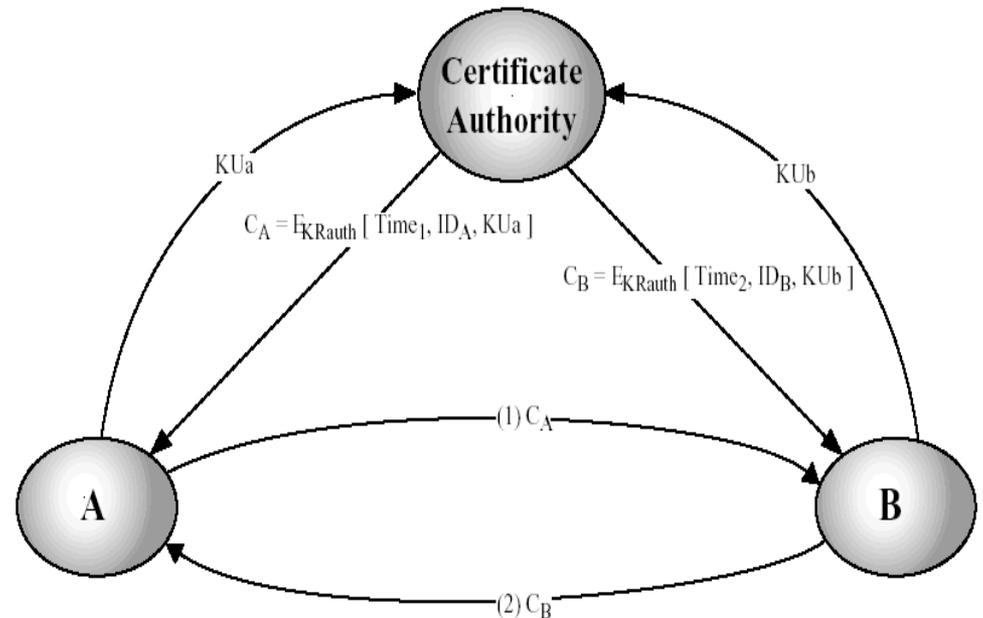
- RB:
  - ▣ Decodes the message using  $K_{RB}$
  - ▣ Validates the message sender's id to be  $ID_A$
  - ▣ Extract  $N_1$
  - ▣ Requests IAs public key in steps (4) and (5)
- 6. RB sends a new nonce  $N_2$  together with  $N_1$  to IA, encrypted using  $K_{UA}$
- 7. IA:
  - ▣ Decrypts the message using  $K_{RA}$
  - ▣ Validates the message origin (RB) by checking  $N_1$
  - ▣ Encrypts  $N_2$  using  $K_{UB}$  and sends it to RB
- RB:
  - ▣ Decrypts the message using  $K_{RB}$
  - ▣ Validates the message authenticity by checking  $N_2$

# Key Management via Public-Key Authority

- Main problem:
  - ▣ The public-key authority is a single point of failure! If it is compromised (e.g., via a DoS attack), keys cannot be distributed
- Therefore:
  - ▣ Introduction of digital certificates, that can be used by nodes to exchange public keys without contacting a public-key authority
- Requirements:
  - ▣ Any participant can read a certificate to determine the name and public key of the certificate's owner
  - ▣ Any participant can verify that the certificate originated from the certificate authority and is not counterfeit
  - ▣ Only the certificate authority can create and update certificates
  - ▣ Any participant can verify the currency of the certificate

# Key Management via Certificate Authority (CA)

- The CA is the root of trust
- Participants (A and B in the diagram) acquire a digital certificate each **that binds their public key  $KU_x$  to their identity  $ID_x$**
- These certificates are subsequently exchanged to
  - ▣ setup a secure connection
  - ▣ authenticate both endpoints



# Key Management via a Certificate Authority: Acquiring a Certificate

- The CA receives a request from A (or B) to certify their public key
- The CA creates a document that contains A's (or B's) identity  $ID_x$ , public key  $KU_x$  and the document's validity period  $Time_1$
- The CA signs, i.e. encrypts, this document using its private key  $K_{R_{auth}}$ , and returns it to A (or B)
- Every entity that possesses CA's public key can validate the authenticity of a (signed) document by decoding it

# Key Management via a Certificate Authority

- A and B have acquired their certificates from the CA at some stage in the past, and have a copy of CA's public key
- Now A wants to securely communicate with B, resulting in the following steps:
  - ▣ A sends  $C_A$  to B, and B in return sends  $C_B$  to A
  - ▣ Both mutually validate
    - the other party's certificate by decoding it using the CA's public key
    - the certificate's sender by comparing  $ID_x$  in the received certificate with the network address of the sender
- However, certificates are public documents and either side's network address could have been spoofed by an attacker, that impersonates A or B
- Therefore, additional steps as shown shortly are required

# Example for a simple unsigned XML-based Certificate

```
<SimpleCertificate>
  <Authority> NUI-Galway </Authority>
  <SignatureType> SimpleSignature </SignatureType>
  <Created> 15-NOV-2019 </Created>
  <Expires> 14-NOV-2024</Expires>
  <OwnerName> William Simpson </OwnerName>
  <KeyType> RSA </KeyType>
  <KeyLength> 256 </KeyLength>
    <PublicKey>
      gHJgjh57JKf#j'\;gkwg@45tRET46$Ed
    </PublicKey>
</SimpleCertificate>
```

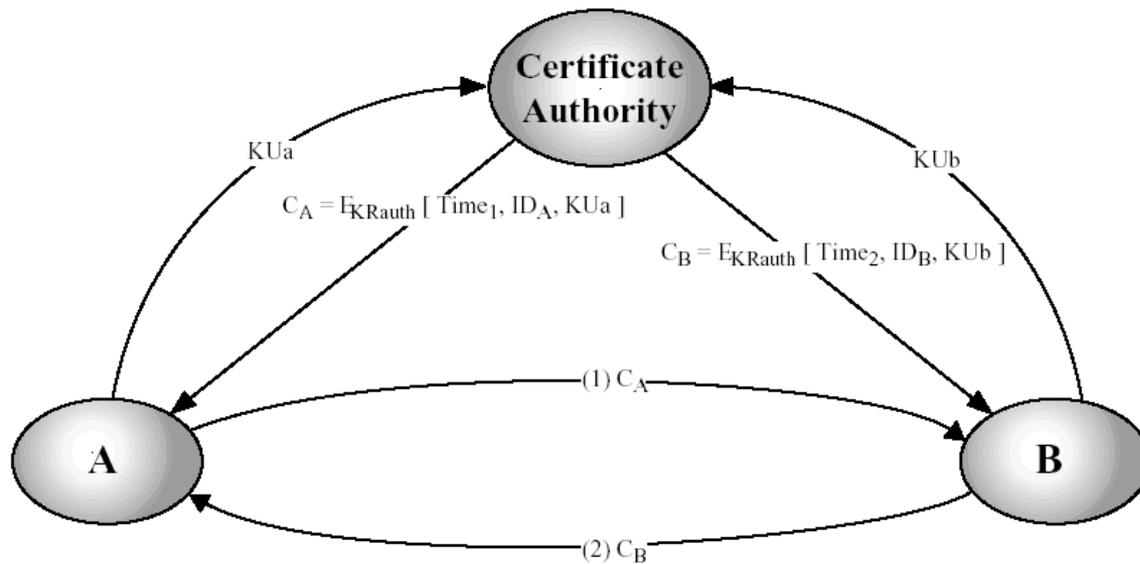
# Example for a signed simple XML-based Certificate

hi6IGHJ^gu#" :HGLFdyUf56EEdx3X5XxXuAzyI;\*6/.,:g  
wqui^09udfsqfhaspfaj#w994HK51'fjg095u321\er3f2875  
gyor23ro32rj6yhggIGUoowqru07t99Y)\*-36wrqwUluill  
No891 u[ `[c0 t6Rt\*(v858e3w70-v794x3xz7c8c9799999s  
9udfsqfhaspfaj7t99 -v794x3xz7c8c9799 09udfsqfhaspfaj#  
w994HK51'fjg095u32nfjewYU87Deffe7s%Rk936-J0D9d

- The signed certificate is just undecipherable text
- Its validation requires the decoding of the entire document
- Later → X.509 digital certificates provide a much neater solution

# In-Class Activity

- Can you identify any “weak spots” in the CA system below?

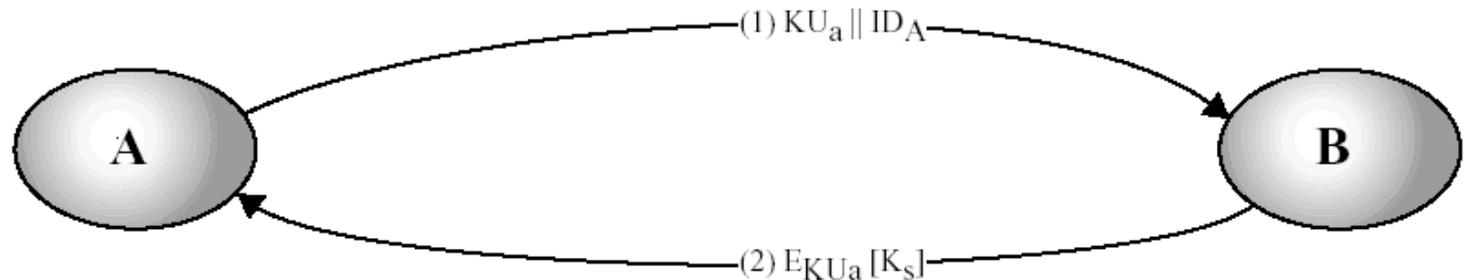




# Symmetric Key Distribution with Public Key Cryptosystems

# Symmetric-Key Distribution Using a Public Key Encryption

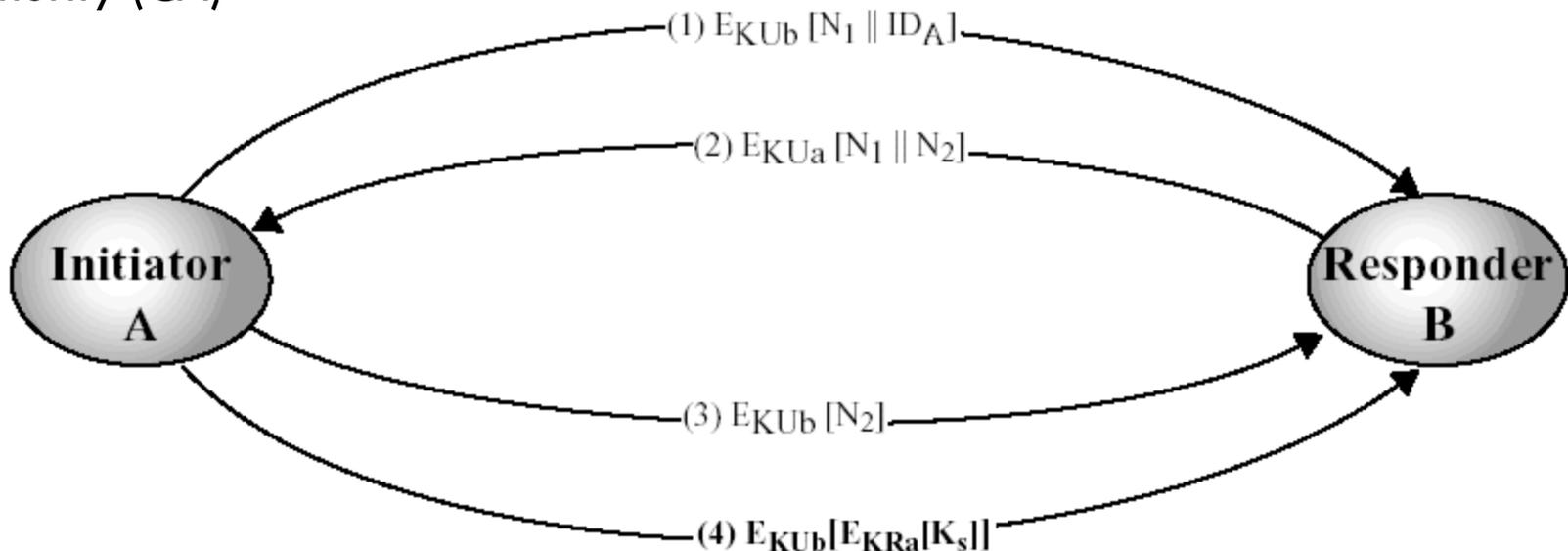
- ❑ Public-key encryption is slow
- ❑ Therefore, it is often used for the distribution of a secret (session) key to be used for conventional symmetric encryption
- ❑ This is an example for a simple secret-key distribution, where A shares its public key  $KU_a$  with B:



- ❑ Problem: B cannot authenticate A or their public key (and vice versa), therefore
  - ❑ A or B could be impersonated via network address spoofing
  - ❑ A MitM attacker could place itself between A and B

# Secret-Key Distribution with Confidentiality and Authentication

- In this protocol both sides have already acquired and validated the other side's certificate (that contains the owner's identity  $ID_x$ ) and public key
- The 4-step authentication process guarantees that
  - mutual authentication is provided (no network address spoofing possible)
  - a MitM attacker cannot place itself between A and B
- It is the logical continuation of the protocol "Key Management via Certificate Authority (CA)"



CT437  
COMPUTER SECURITY AND FORENSIC COMPUTING

DIGITAL CERTIFICATES

Dr. Michael Schukat

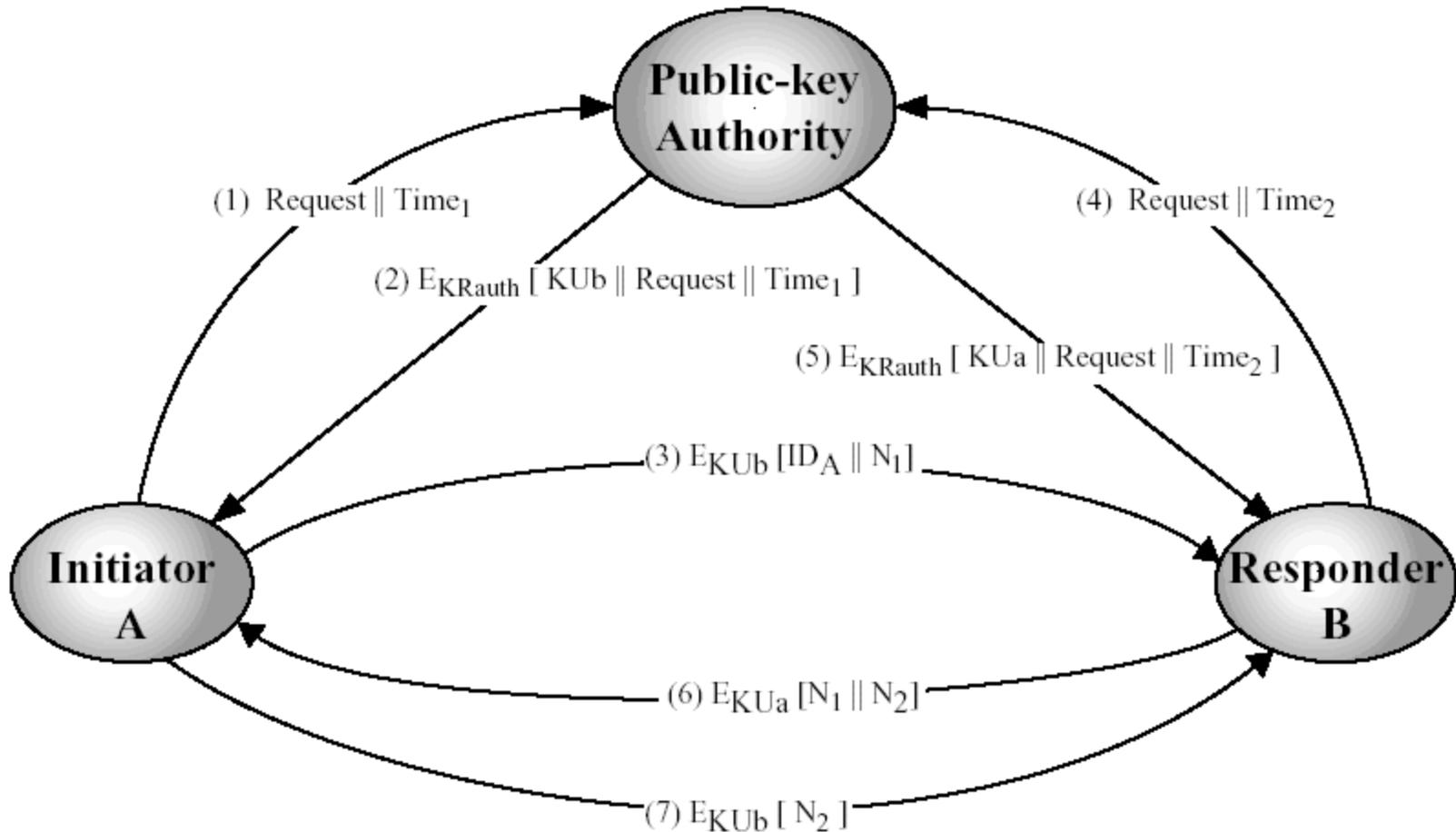


# Lecture Content

2

- Recap motivation digital certificates
- Digital certificates and certificate authorities
  - ▣ Concepts
  - ▣ Applications
  - ▣ Case studies

# Recap: Key Management via Public-Key Authority



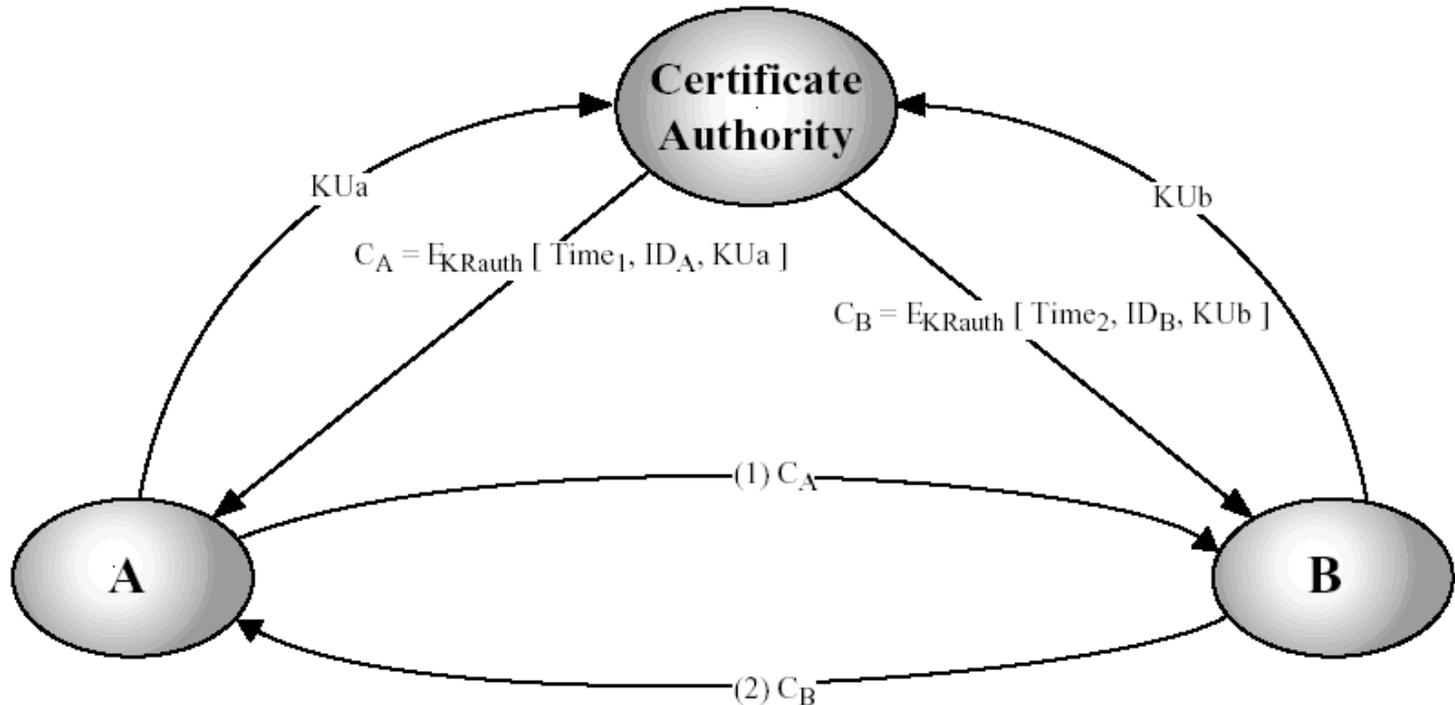
□ Please see also lecture notes “Public Key Encryption”

# Recap: Key Management via Public-Key Authority

- Drawback of public-key authority:  
**Authority is a bottleneck! If it is compromised (e.g. via a DoS attack), public keys cannot be requested or distributed**
- Therefore: Introduction of certificates, that can be used by participants to exchange keys without contacting a public-key authority
- Requirements:
  - ▣ Any participant can read a certificate to determine the name and public key of the certificate's owner
  - ▣ Any participant can verify that the certificate originated from the certificate authority and is not counterfeit
  - ▣ Only the certificate authority can create, renew and revoke certificates
  - ▣ Any participant can verify the validity (i.e., expiration or revocation) of the certificate

# Recap: Key Management via Certificate Authority

- Architecture allows exchange of public-key certificates (PKC):



# Recap: Example for a Simple XML-Based Signature: Plaintext

```
<SimpleSignature>  
  <Authority> NUI-Galway </Authority>  
  <SignatureType> SimpleSignature </SignatureType>  
  <Created> 15-NOV-2019 </Created>  
  <Expires> 14-NOV-2020</Expires>  
  <OwnerName> William Simpson </OwnerName>  
  <KeyType> RSA </KeyType>  
  <KeyLength> 256 </KeyLength>  
    <PublicKey>  
      gHJgjh57JKf#j'\;gkwg@45tRET46$Ed  
    </PublicKey>  
</SimpleSignature>
```

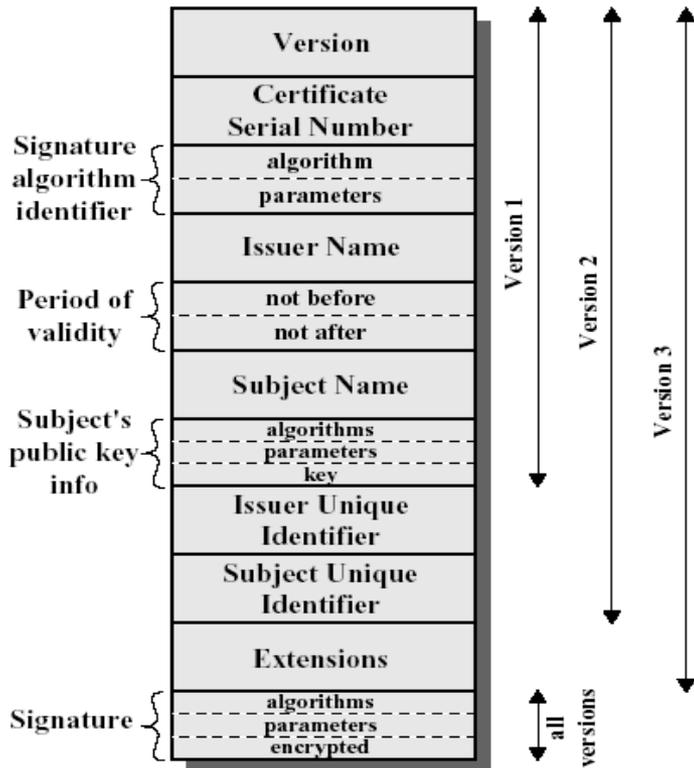
# Recap: Example for a Simple XML-Based Signature: Ciphertext

hi6lGHJ^gu#"':HGLFdyUf56EEdx3X5XxXuAzyI;\*6/.,:g  
wqui^09udfsqfhaspfaj#w994HK51'fjg095u321\er3f2875  
gyor23ro32rj6yhggIGUoowqru07t99Y)\*-36wrqwUluill  
No891 u[ `[c0 t6Rt\*(v858e3w70-v794x3xz7c8c9799999s  
9udfsqfhaspfaj7t99 -v794x3xz7c8c9799 09udfsqfhaspfaj#  
w994HK51'fjg095u32nfjewYU87Deffe7s°%Rk936-J0D9d

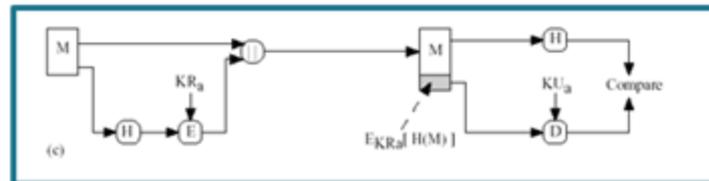
# X.509 Certificates

- X.509 is an International Telecommunication Union (ITU) standard defining the format of Public Key Certificates (PKC)
  - ▣ Public key management generally involves the use of PKCs
  - ▣ PKCs bind an identity (the subject) to a public key,
    - usually with other info such as period of validity, rights of use etc.
    - with all contents signed by a trusted Certification Authority (CA), the issuer
  - ▣ Therefore, X.509 certificates are also called **identity certificates**
  - ▣ In all PKC use cases (e.g., peer-to-peer data communication), involved parties either already know, or can securely obtain and verify the public key of the CA to verify the certificate
  
- X.509 certificates are widely used in secure email (S/MIME - Secure Multipurpose Internet Mail Extensions), secure web browsing (TLS / HTTPS), secure software patching, etc.

# X.509 Certificate Structure



- The certificate is issued by a CA, who signs the certificate
  - ▣ The certificate is hashed, and the hash is encoded (signed) by the CA using its private key
  - ▣ In the diagram below,  $M$  is the entire certificate excluding the signature, which in turn is the encrypted hash
- The certificate can be validated by anyone who has a trusted (!) copy of the issuer's (CA's) public key:

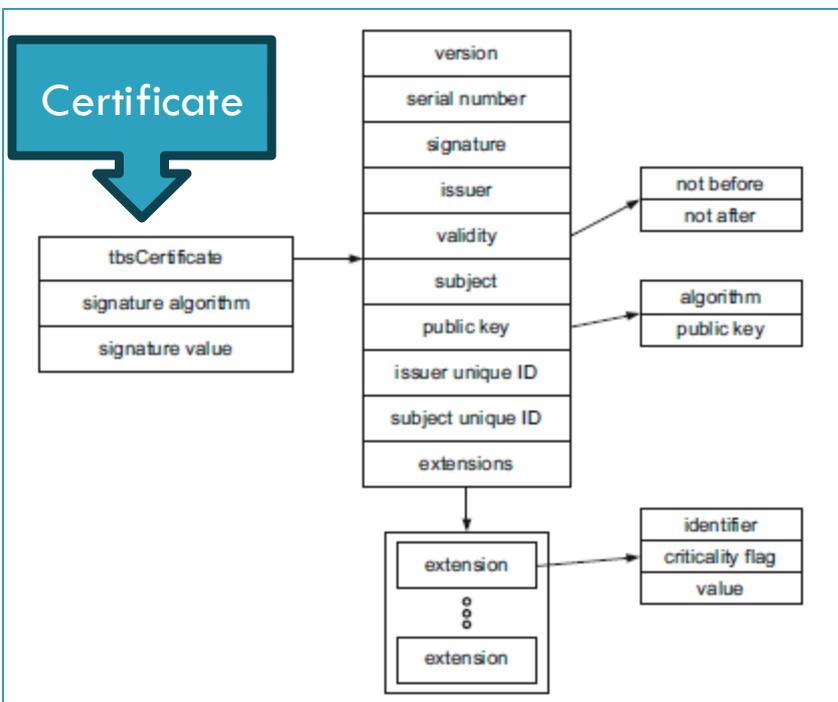


$KR_a$  = CA private key  
 $KU_a$  = CA public key

# X.509 Certificate Specification

10

- Digital certificates are described via ASN.1
- Abstract Syntax Notation One (ASN.1) is a standard interface description language for defining data structures that can be serialised and de-serialised in a cross-platform way (→ later)

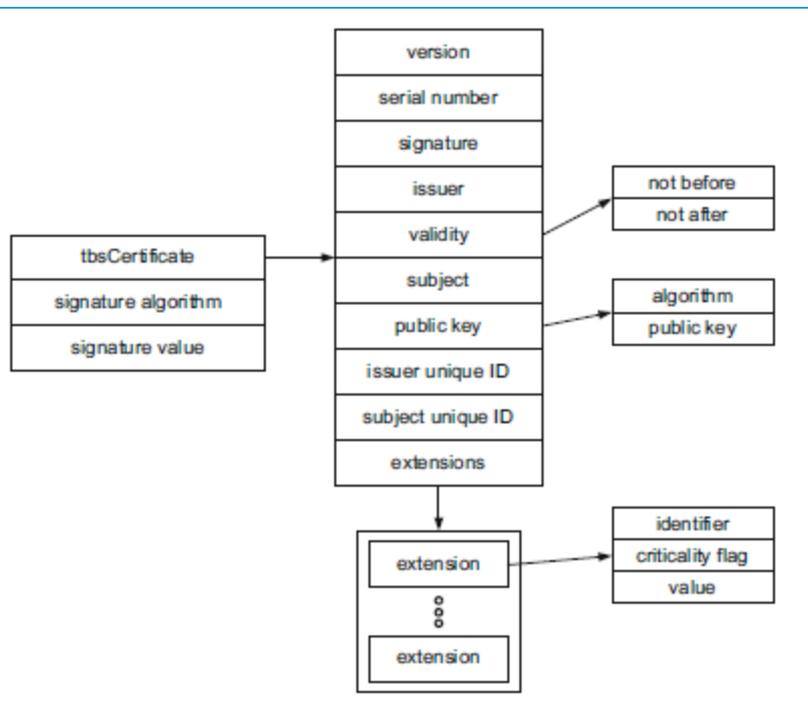


```
Certificate ::= SEQUENCE {
    tbsCertificate      TBSCertificate,
    signatureAlgorithm  AlgorithmIdentifier,
    signatureValue      BIT STRING }
```

```
TBSCertificate ::= SEQUENCE {
    version             [0] EXPLICIT Version DEFAULT v1,
    serialNumber        CertificateSerialNumber,
    signature           AlgorithmIdentifier,
    issuer              Name,
    validity            Validity,
    subject             Name,
    subjectPublicKeyInfo SubjectPublicKeyInfo,
    issuerUniqueID     [1] IMPLICIT UniqueIdentifier OPTIONAL,
    subjectUniqueID    [2] IMPLICIT UniqueIdentifier OPTIONAL,
    extensions          [3] EXPLICIT Extensions OPTIONAL }
```

# X.509 Certificates and OID

11



- X.509 digital certificates contain various fields containing mandatory and optional attributes
  - ▣ Mainly extension are optional
- These attributes are described / encoded using **Object Identifiers (OID)** → next slide
- A digital certificate is a structured list of OIDs and attribute values
- This list is converted into a data structure encoded using BER (Basic Encoding Rules) → later

# Object Identifiers (OID)

- ❑ OIDs are a standardised identifier mechanism for naming any object, concept, or "thing" with a globally unambiguous persistent name
- ❑ OIDs are dotted numbers, with similar concepts often having identical or similar OID pre-fixes
- ❑ X.509 attribute values are either instances of primitive data types (e.g., an integer for version number), or are described by an OID
- ❑ For example, all (standardised) cryptographic algorithms used / supported by X.509 have their unique OID – see also the table above

Algorithm	Type	OID
MD5	Cryptographic hash function	1.2.840.113549.2.5
SHA1	Cryptographic hash function	1.3.14.3.2.26
SHA256	Cryptographic hash function	2.16.840.1.101.3.4.2.1
SHA384	Cryptographic hash function	2.16.840.1.101.3.4.2.2
SHA512	Cryptographic hash function	2.16.840.1.101.3.4.2.3
SHA256withDSA	Digital signature	2.16.840.1.101.3.4.3.2
SHA256withECDSA	Digital signature	1.2.840.10045.4.3.2
SHA384withECDSA	Digital signature	1.2.840.10045.4.3.3
SHA512withECDSA	Digital signature	1.2.840.10045.4.3.4
MD5withRSA	Digital signature	1.2.840.113549.1.1.4
SHA1withRSA	Digital signature	1.2.840.113549.1.1.5
SHA1withDSA	Digital signature	1.2.840.10040.4.3
SHA1withECDSA	Digital signature	1.2.840.10045.4.1
AES with 128 bit key in ECB mode	Secret key encryption	2.16.840.1.101.3.4.1.1
AES with 256 bit key in CBC mode	Secret key encryption	2.16.840.1.101.3.4.1.42
HMAC-MD5	MAC	1.3.6.1.5.5.8.1.1
HMAC-SHA1	MAC	1.3.6.1.5.5.8.1.2
RSA	Public key encryption	1.2.840.113549.1.1.1

# OIDs in Digital Certificates

- In the mock-up example attribute OIDs are replaced with their **name**
- **Other descriptors** don't appear in a certificate and are only added to increase readability
- Note that **Issuer** / **Subject** and **NotBefore** / **NotAfter** attributes can be only distinguished via their position in the cert (i.e, **Subject** appears after the **Issuer**; **notAfter** appears after **NotBefore**)

**Version:** 3

**Serial Number:** 3c:50:33:c2:f8:e7:5c:ca:07:c2:4e3:f2:e8:0e:4f

**Issuer:** O=VeriSign, Inc., OU=VeriSign Trust Network,

OU=www.verisign.com

CN=VeriSign Class 1 CA

**Validity NotBefore:** Jan 13 00:00:00 2021 GMT **NotAfter:** Mar 13 23:59:59 2026 GMT

**Subject:** O=VeriSign, Inc., OU=VeriSign Trust Network, OU=www.verisign.com CN=Lawrie Brown

Email=lawrie.brown@canb.aug.org.au

**Subject Public Key Info:** rsaEncryption RSA Public Key: (512 bit):

00:98:f2:89:c4:48:e1:3b:2c:c5:d1:48:67:80:53: d8:eb:4d:4f:ac:31:a9:fd:

11:68:94:ba:44:d8:48: 46:0d:fc:5c:6d:89:47:3f:9f:d0:c0:6d:3e:9a:8e:ec:

82:21:48:9b:b9:78:cf:aa:09:61:92:f6:d1:cf: 45:ca:ea:8f:df

**Signature Algorithm:** SHA1 withRSA

**Signature Value:** 5a:71:77:c2:ce:82 ...

OID of Version:

2.5.29.19

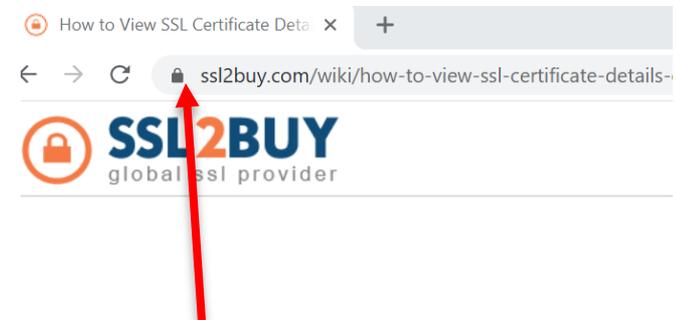
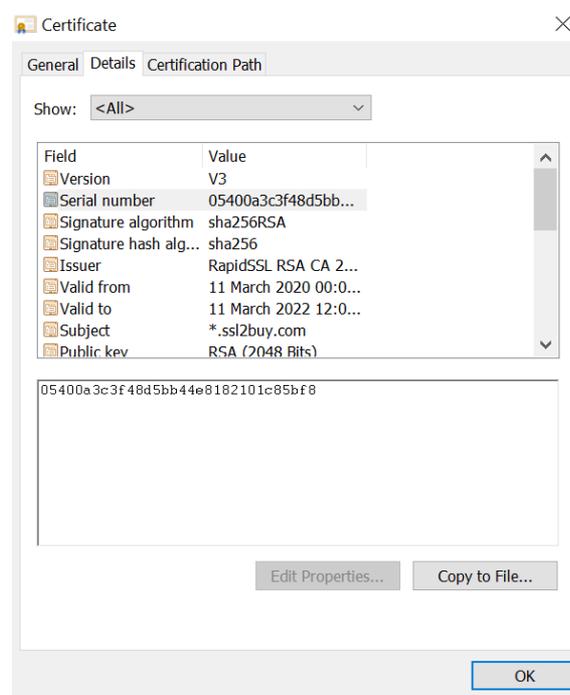
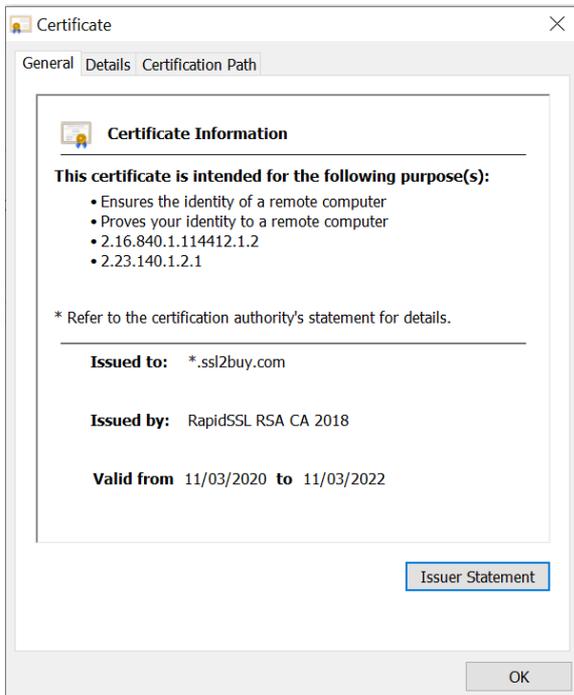
# In Class Activity: Inspect Digital Certificates on your Device / Browser

14

- ❑ Android (version 11):
  - ❑ Open Settings
  - ❑ Tap “Security”
  - ❑ Tap “Encryption & credentials”
  - ❑ Tap “Trusted credentials.” This will display a list of all trusted certs on the device
- ❑ In Chrome (Windows OS):
  - ❑ Goto Settings
  - ❑ Open “Security and Privacy” and “Security”
  - ❑ Open “Manage device certificates”
  - ❑ iOS devices require you to open the keystore
- ❑ iOS devices:
  - ❑ Tap Settings > General > About
  - ❑ Scroll to the bottom of the list
  - ❑ Tap Certificate Trust Settings
  - ❑ Follow the link
- ❑ Generic: <https://www.ssllabs.com/ssltest/?form=MG0AV3>

# Example: X.509 Certificates in Web Browsers

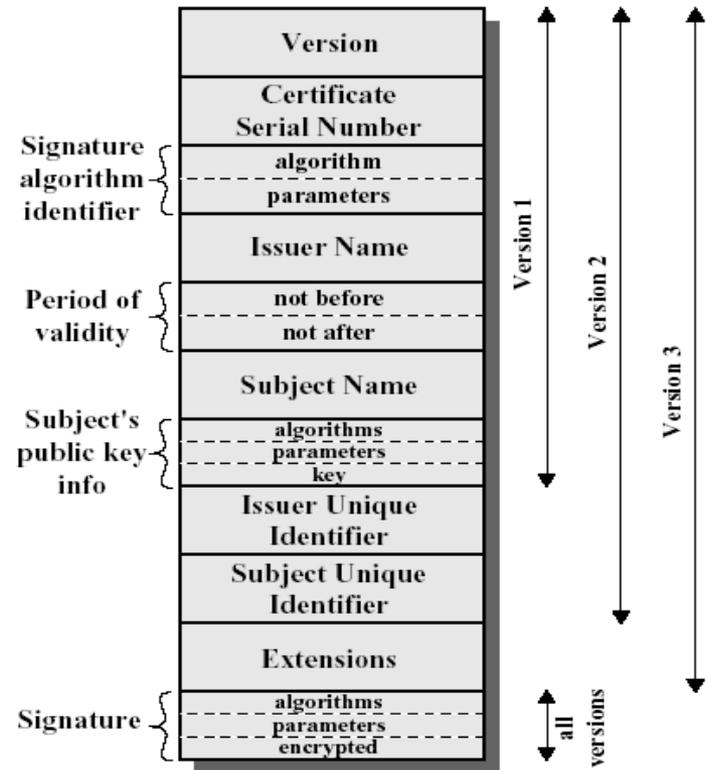
- In Chrome: see <https://www.ssl2buy.com/wiki/how-to-view-ssl-certificate-details-on-chrome-56>



# X.509 Certificates in Detail: Field *version*

16

- X.509 certificates went through three iterations before v3 was finally released in 1996 (!)
- The value of the *version* field (OID 2.5.29.19) is an integer
  - ▣ X.509v1 → 0
  - ▣ X.509v2 → 1
  - ▣ X.509v3 → 2



# Fields *Issuer* and *Subject*

17

- **Issuer** is the certificate authority (CA) that signed the certificate

- **Subject** is the owner of the cert

- Both their descriptions are provided via a string called the **Distinguished Name (DN)**

- A DN is a sequence of OID encoded attributes and their values

- Example: CN=Alice, OU=Administration, O=TU Darmstadt, C=DE

- ▣ This DN describes a person with common name (CN) Alice, who belongs to the organisational unit (OU) “administration” of the organization (O) “TU Darmstadt” that operates in the country (C) Germany

- ▣ Here the DN reflects a logical hierarchy of a person belonging to an organisational unit which is part of an organisation located in a country

- ▣ The DN as string would look like “2.5.4.3Alice2.5.4.11Administration ...”

Attribute type	String representation	OID
countryName	C	2.5.4.6
organizationName	O	2.5.4.10
organizationalUnitName	OU	2.5.4.11
commonName	CN	2.5.4.3
localityName	L	2.5.4.7
stateOrProvinceName	ST	2.5.4.8

# Field *serialNumber*

18

- The certificate issuer assigns a unique serial number to each signed certificate, composed as follows:
  - ▣ *SerialNumber* (OID 2.5.4.5)
  - ▣ a positive 20 byte long integer
  - ▣ E.g. “2.5.4.501234567890123456789”
    - As we will see later, each item is in fact represented as a Type-Length-Value triplet
- The serial number field is mandatory
- Therefore, the combination of the issuer name **and** the serial number uniquely identifies a certificate
  - ▣ Consider a subject could have multiple certificates signed by the same CA
  - ▣ Note that different CA can issue a certificate that has the same serial number

# Field *signature*

19

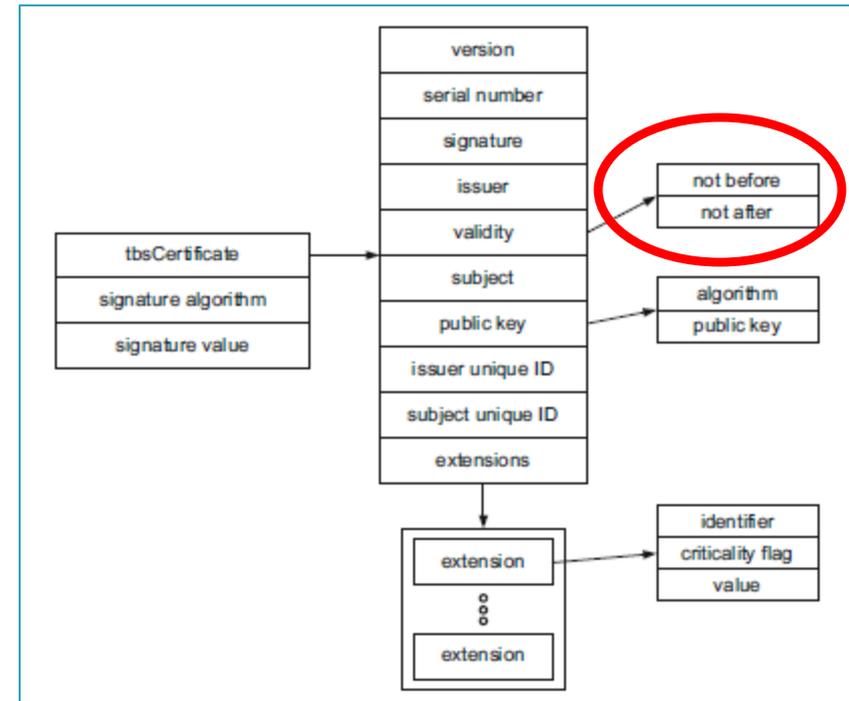
- The issuer of an X.509 certificate signs the certificate
- The mandatory field *signature* describes the signature algorithm that was used by the issuer to sign the certificate
- The field is of type *AlgorithmIdentifier* (OID 1.3.6.1.1.15.7)
- It is complemented by the OID of the signature algorithm that is used (see table) and optional additional parameters

Algorithm	Type	OID
MD5	Cryptographic hash function	1.2.840.113549.2.5
SHA1	Cryptographic hash function	1.3.14.3.2.26
SHA256	Cryptographic hash function	2.16.840.1.101.3.4.2.1
SHA384	Cryptographic hash function	2.16.840.1.101.3.4.2.2
SHA512	Cryptographic hash function	2.16.840.1.101.3.4.2.3
SHA256withDSA	Digital signature	2.16.840.1.101.3.4.3.2
SHA256withECDSA	Digital signature	1.2.840.10045.4.3.2
SHA384withECDSA	Digital signature	1.2.840.10045.4.3.3
SHA512withECDSA	Digital signature	1.2.840.10045.4.3.4
MD5withRSA	Digital signature	1.2.840.113549.1.1.4
SHA1withRSA	Digital signature	1.2.840.113549.1.1.5
SHA1withDSA	Digital signature	1.2.840.10040.4.3
SHA1withECDSA	Digital signature	1.2.840.10045.4.1
AES with 128 bit key in ECB mode	Secret key encryption	2.16.840.1.101.3.4.1.1
AES with 256 bit key in CBC mode	Secret key encryption	2.16.840.1.101.3.4.1.42
HMAC-MD5	MAC	1.3.6.1.5.5.8.1.1
HMAC-SHA1	MAC	1.3.6.1.5.5.8.1.2
RSA	Public key encryption	1.2.840.113549.1.1.1

# Field validity

20

- The validity field (OID 2.5.29.16) indicates the validity period of the certificate
- This field contains just two dates, which have no OID and are just referenced as *notBefore* and *notAfter*
- Between these two dates the certificate is valid unless it has been revoked (→ later)



# Field *subjectPublicKeyInfo*

21

- The *subjectPublicKeyInfo* field (OID 1.2.840.1.1.3549.1.1.1) contains the public key data that is certified by the certificate
- This data is described as a sequence containing the OID of an algorithm followed by optional parameters and the public key
- The example below shows the ASN.1 structure of an EC public key and its parameters

```
ECParameters ::= SEQUENCE {  
    version      ECPVer,  
    fieldID      FieldID,  
    curve        Curve,  
    base         ECPoint,  
    order        INTEGER,  
    cofactor    INTEGER OPTIONAL }
```

# Fields *issuerUniqueID* and *subjectUniqueID*

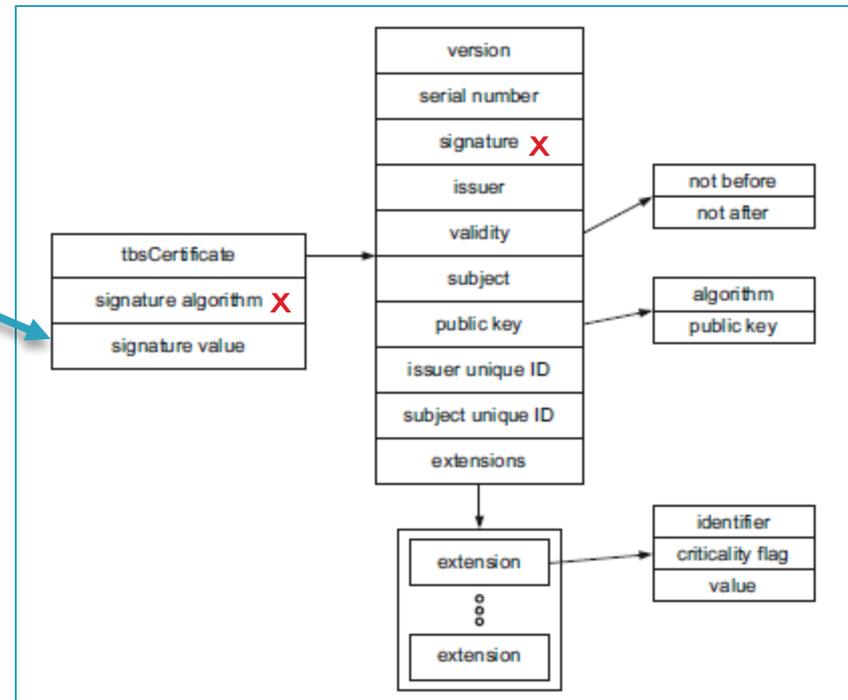
22

- The *subjectUniqueID* and *issuerUniqueID* fields were introduced with X.509v2
- It may happen that the same distinguished name is assigned to different entities
- For example, if a *subjectDN* is used twice by an issuer, then the owner of the corresponding certificate is not uniquely determined by the subject DN
- To make the owner description unique, the *subjectUniqueID* field may be added
- The content of that field is a binary string that is a unique identifier for the owner of the certificate
- Likewise, several issuers may share the same DN
- In this case the *issuerUniqueID* field resolves the situation
- **However, the use of these fields is not recommended because they make certificate use more complicated**

# Field *signatureAlgorithm* and *signatureValue*

23

- The signature algorithm that was used to sign the certificate is specified twice in an X.509 certificate:
  - In the *tbsCertificate* structure (under *signature*), as seen before
  - In the *signatureAlgorithm* field
- *signatureValue* holds the signature on the *tbsCertificate* content of the certificate, i.e. the encrypted hash of *tbsCertificate* (but not *signatureAlgorithm*)



# X.509 Certificate Extensions

24

- The contents of X.509 version 1 and version 2 certificates turned out to be insufficient in practice
- X.509v3 certificates may contain extensions which support various PKI processes
- The ASN.1 structure of X.509 certificate extensions can be seen below:
  - ▣ The first field in such an extension is *extnID*, which contains the OID of the extension
  - ▣ Next, any extension contains a criticality indicator *critical*
    - If its value is true, then all applications that use this certificate must evaluate the extension; If an application is unable to do so, then it must consider the certificate to be invalid
  - ▣ The third field contains the extension description

```
Extension ::= SEQUENCE {  
    extnID      OBJECT IDENTIFIER,  
    critical    BOOLEAN DEFAULT FALSE,  
    extnValue   OCTET STRING }
```

# Extension Field *AuthorityKeyIdentifier*

25

- Problem:
  - ▣ An issuer / CA may have multiple key pairs to sign a digital certificate
  - ▣ If a given certificate is to be validated, the correct public key must be chosen
  - ▣ The information in the issuer field just points to the CA, but not to the correct key
- Solution:
  - ▣ This extension, also known as AKI extension or AKIE, is to support applications in identifying the public key of the issuer, to be used to verify the certificate signature
- The authority key identifier extension must be present in any X.509v3 certificate unless the certificate is self-signed (→ later)
- Also, this extension must not be marked critical
- Typically, this value is a 20-byte SHA-1 hash of the public key belonging to the private key of the issuer that was used to sign the certificate
- Similarly, the extension field *SubjectKeyIdentifier* can be used to hash the subject's public key (more later)

# Extension Field *KeyUsage*

26

- The *KeyUsage* extension indicates what the public key contained in a certificate can be used for
- Possible uses are:
  - ▣ *digitalSignature*  
The public key can be used to verify digital signatures, for example, to validate the authenticity and origin of signed emails
  - ▣ *nonRepudiation*  
The public key can be used to verify signatures to provide nonrepudiation
    - E.g. denial of a digitally contract being signed
  - ▣ *keyEncipherment*  
The public key may be used to encrypt symmetric session keys
  - ▣ *dataEncipherment*  
The public key may be used to encrypt data
  - ▣ *keyAgreement*  
The public key may be used in a key agreement scheme (i.e., Diffie-Hellman)

```
KeyUsage ::= BIT STRING {  
    digitalSignature      (0),  
    nonRepudiation       (1),  
    keyEncipherment      (2),  
    dataEncipherment     (3),  
    keyAgreement         (4),  
    keyCertSign          (5),  
    cRLSign              (6),  
    encipherOnly         (7),  
    decipherOnly         (8) }
```

# Extension Field *KeyUsage*

27

## □ Possible uses are:

### ▣ keyCertSign

The private key corresponding to the public key in the certificate may be used to sign certificates. The public key is then used to verify certificate signatures

### ▣ cRLSign

The private key corresponding to the public key in the certificate may be used to sign certificate revocation lists (→ later)

### ▣ encipherOnly

Undefined in the absence of the keyAgreement bit

When the encipherOnly bit is asserted and the keyAgreement bit is also set, the subject public key may be used only for enciphering data while performing key agreement

### ▣ decipherOnly

ditto

## □ Many clients and applications evaluate the key usage extension

### ▣ Example: An email client that has access to several certificates of the recipient of an email can tell by the key usage extension which certificate is to be used for

- email encryption

- verifying signatures of received emails

```
KeyUsage ::= BIT STRING {  
    digitalSignature      (0),  
    nonRepudiation       (1),  
    keyEncipherment      (2),  
    dataEncipherment     (3),  
    keyAgreement         (4),  
    keyCertSign          (5),  
    cRLSign              (6),  
    encipherOnly         (7),  
    decipherOnly         (8) }
```

# Extension Field *SubjectAlternativeName*

28

- Up to now a subject is identified via its subject field that contains the distinguished name (DN) with all the aforementioned attributes
- This extension binds additional names to the public key in the certificate not covered by the DN
- Typical names contained in this extension are owner's

- email address
- IP address
- domain name (DNS names)
- uniform resource identifier (URIs)

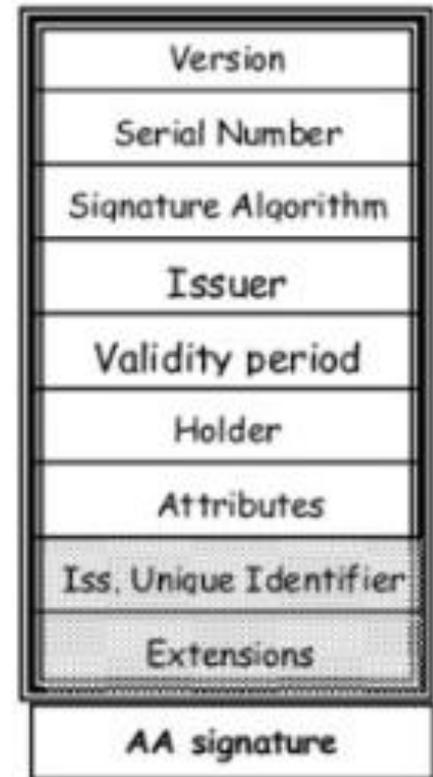
```
X509v3 Subject Alternative Name:  
DNS:*.wikipedia.org, DNS:*.mediawiki.org, DNS:*.wikibooks.org, DNS:*.wikidata.org,  
DNS:*.wikimedia.org, DNS:*.wikimediafoundation.org, DNS:*.wikinews.org, DNS:*.wikipedia.org,  
DNS:*.wikiquote.org, DNS:*.wikisource.org, DNS:*.wikiversity.org, DNS:*.wikivoyage.org, DNS:*.wiktionary.org,  
DNS:*.mediawiki.org, DNS:*.planet.wikimedia.org, DNS:*.wikibooks.org, DNS:*.wikidata.org, DNS:*.wikimedia.org,  
DNS:*.wikimediafoundation.org, DNS:*.wikinews.org, DNS:*.wikiquote.org, DNS:*.wikisource.org, DNS:*.wikiversity.org,  
DNS:*.wikivoyage.org, DNS:*.wiktionary.org, DNS:*.wmfusercontent.org, DNS:*.zero.wikipedia.org, DNS:mediawiki.org,  
DNS:w.wiki, DNS:wikibooks.org, DNS:wikidata.org, DNS:wikimedia.org, DNS:wikimediafoundation.org, DNS:wikinews.org,  
DNS:wikiquote.org, DNS:wikisource.org, DNS:wikiversity.org, DNS:wikivoyage.org, DNS:wiktionary.org,
```

- For example, if the public key in the certificate is used for authentication of the web server of an organisation, the DNS name or the IP address of that server is typically contained in this extension
  - Clients that connect securely to such a server verify that the IP address or the DNS name of the server matches the IP address or DNS name contained in this extension (more later)
- Example: UoG certificate

# Attribute Certificates

30

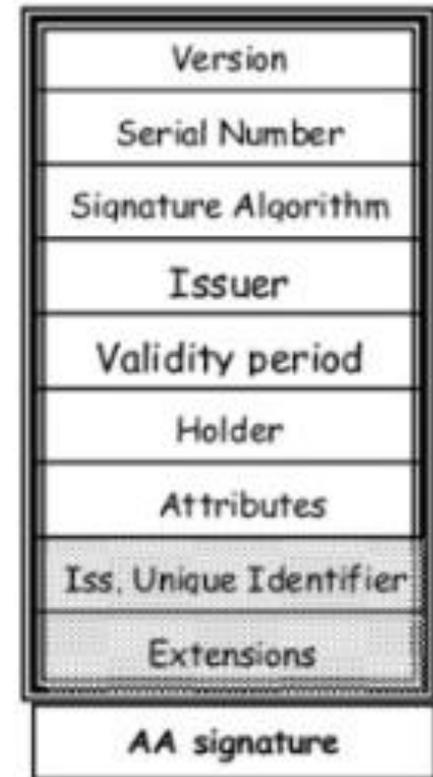
- An attribute certificate binds certain privileges or attributes to their owners
- It is signed by an attribute authority (AA)
- For example, attribute certificates are used in smartphones to provide apps with the permission to access certain phone resources, e.g., a user's address book
- In contrast to identity certificates, an attribute certificate does not contain the owner's public key
- On the other hand, identity certificates could be complemented by additional attributes encoded as new extension fields, and to some extent mimick attribute certificates
  - ▣ Such a certificate is also called a **combined certificate**



# Attribute Certificates

31

- Attributes are TLV triples as well, uniquely identified by their OID
- Attribute certificates are often used in conjunction with X.509 public key certificates
- For example, consider a firmware update for a mobile phone:
  - ▣ It is signed by its issuer and the signature verification key is authenticated by a certificate
  - ▣ In addition, an attached verifiable attribute certificate specifies whether or not this update may be used for a certain type of mobile phone



# Example Home Automation

32

- Consider a range of wireless IoT home automation devices that require
  1. secure inter-device communication
  2. end-point authentication
  3. optimised inter-device communication (i.e. the smart fridge and the electricity smart meter only exchange energy consumption data)
  4. the exclusion of 3<sup>rd</sup> party devices
- All devices are integrated in a home-automation network (HAN) and form P2P connections via some handshake protocol
- Each device has its own X.509 public key certificate
  - ▣ Certificates are exchanged between paired devices to provide end-point authentication (1) and secure session keys for secure wireless data communication (2)



# Example Home Automation

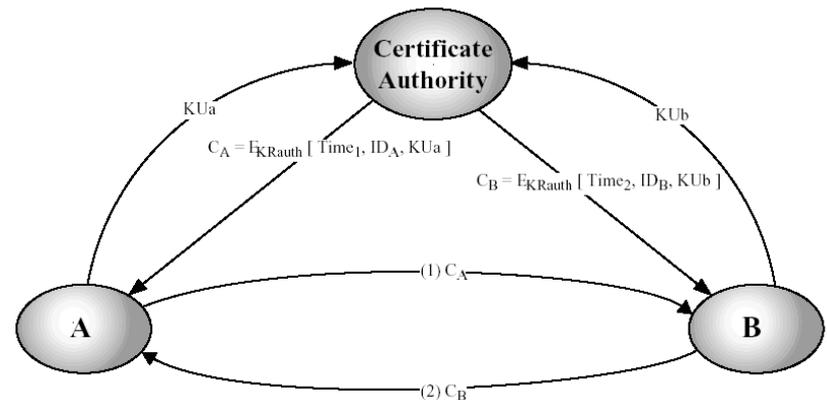
33

- However, in order to address 3. and 4., additional information must be encoded:
  - ▣ The device manufacturer
  - ▣ The device type
  - ▣ Rules that describe other devices it can talk to
- This info can be encoded in
  - ▣ an additional attribute certificate, or
  - ▣ additional extension fields of the public key certificate (creating a combined certificate)
- Subsequently, a device that during the handshake
  - ▣ cannot present these credentials, or
  - ▣ has the incorrect attribute values (e.g. different manufacturer)cannot complete the process and is excluded from the HAN

# Trust Models and Digital Certificates

34

- Problem: Public key cryptography (and subsequently digital certificates) can only be used in practice if users trust the authenticity of the CAs public keys
- For example, in the diagram below, how do A and B acquire the public key of the CA, and why / how can they trust this key?
- The CA is the **root of trust**, but how can this trust be justified?



# Direct Trust

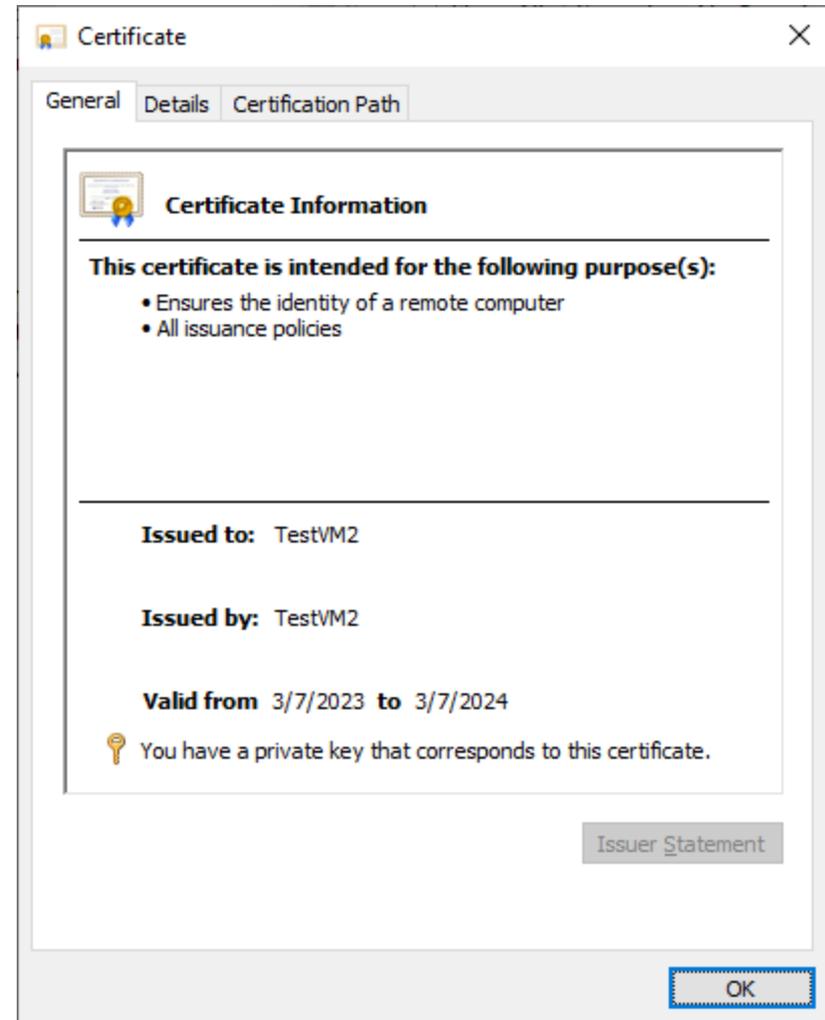
35

- Trust in the authenticity of a public key is direct if the public key is directly obtained from the key owner or its owner directly confirms the authenticity of the key in a way that is convincing for the user
- Example:
  - ▣ Most Linux systems allow the installation of additional software such as updates or services from trusted servers located on the Internet
  - ▣ The authenticity of those software packages is established by a digital signature
  - ▣ The verification of the signature requires a public key, which is embedded in the Linux distribution
  - ▣ The authenticity of this key is guaranteed by the authenticity of the Linux installation image
  - ▣ Such public keys are usually internally stored as **self-signed certificates**
  - ▣ Similarly, self-signed certificates can be found in web browsers

# Self-Signed Digital Certificates

36

- ❑ Self-signed digital certificates are issued by the public key owner themselves, as opposed to a certificate authority (CA) issuing them
- ❑ Subject and issuer fields point to the same identity and the cert is signed using the owner's private key
- ❑ Obviously, they do not provide any trust value per se
  - ❑ However, root CA have self-signed certificates (→ later)
- ❑ See also self-signed browser certificates using OpenSSL
  - ❑ [https://www.akadia.com/services/ssh\\_test\\_certificate.html](https://www.akadia.com/services/ssh_test_certificate.html)



# Commercial CAs

- Self-signed certificates have no value to 3<sup>rd</sup> parties, as different users that need to exchange their certs need a common root of trust
- This is achieved by hundreds of companies worldwide that provide digital certificates to clients
  - ▣ e.g. Verisign ([www.verisign.com](http://www.verisign.com)) and SSL ([www.ssl.com](http://www.ssl.com))
- These CAs form a CA hierarchy

Rank	Issuer	Usage	Market Share
1	IdenTrust	43.4%	48.9%
2	DigiCert	16.6%	18.7%
3	Sectigo (Comodo Cybersecurity)	13.8%	15.5%
4	Let's Encrypt	7.2%	8.2%
5	GoDaddy	5.4%	6.1%
6	GlobalSign	2.4%	2.7%

# Certificate Classes

38

- Certificate classes in digital certificates are typically encoded using specific OIDs within the certificate's extensions
- These classes can indicate different levels of validation and trust, such as
  - ▣ domain validation (DV)
  - ▣ organization validation (OV)
  - ▣ extended validation (EV)

# Certificate Classes

39

Certificate Type	Validation Level	Issuance Time	Use Case	Assurance Level
<b>Domain Validation (DV)</b>	Basic	Minutes	Personal websites, blogs, small businesses	Low, does not verify the identity of the subject
<b>Organization Validation (OV)</b>	Intermediate	Few days	Business websites, organizations	Medium, validates the subject's identity
<b>Extended Validation (EV)</b>	Highest	Several days to weeks	E-commerce sites, financial institutions, websites handling sensitive data	High, as the CA conducts a thorough vetting process, including verifying the legal, physical, and operational existence of the organization

# Domain-Validated Certificates

40

- Digital certificates are usually issued to websites
  - ▣ The public key in it is used to setup a secure connection between client browser and server (by negotiating a symmetric key -> later)
- Practically, many CAs often do not do a thorough check on a website (e.g. malware check) or their owners (id, credentials etc.)
- Instead, automatic checks are done, where it is validated that the applicant has control over the website and the DNS of the website domain, e.g.,
  - ▣ Place a specific file at the specific URL on the website
  - ▣ Add a specific DNS record to the website domain
  - ▣ Create an email address in the site domain and receive a password at that email
- As a result, such (HTTPS) certificates are called domain-validated certificates

# Certificate Signing Request (CSR)

41

- ❑ A CSR is a Base64-and BER-encoded message (formally described using ASN.1) sent from an applicant to a CA of the PKI in order to apply for a digital certificate
- ❑ The most common format for CSRs is the PKCS #10 specification
  - ▣ PKCS stands for "Public Key Cryptography Standards"
- ❑ Before creating a CSR, the applicant first generates a key pair, keeping the private key secret
- ❑ The CSR subsequently contains the public key, as well as the following fields (source: Wikipedia):

DN <sup>[2]</sup>	Information	Description	Sample
CN	Common Name	This is <a href="#">fully qualified domain name</a> that you wish to secure	*.wikipedia.org
O	Organization Name	Usually the legal name of a company or entity and should include any suffixes such as Ltd., Inc., or Corp.	Wikimedia Foundation, Inc.
OU	Organizational Unit	Internal organization department/division name	IT
L	Locality	Town, city, village, etc. name	San Francisco
ST	State	Province, region, county or state. This should not be abbreviated (e.g. West Sussex, Normandy, New Jersey).	California
C	Country	The <a href="#">two-letter ISO code</a> for the country where your organization is located	US
EMAIL	Email Address	The organization contact, usually of the certificate administrator or IT department	

# In-class Activity: Generating a Digital Certificate

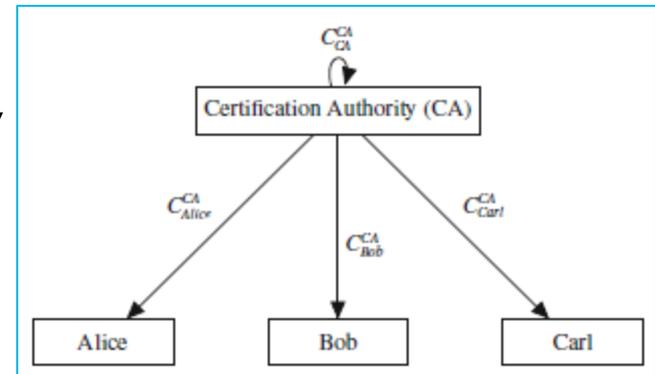
42

- Generate certificate signing request (CSR) via <https://csrgenerator.com/>
- View the CSR <https://lapo.it/asn1js/>
- Create a CSR and submit it to <https://getacert.com/>  
A certificate will be returned
- View the content of this certificate via
  - <https://lapo.it/asn1js/>
  - “Open in PEM format” in <https://getacert.com/>

# Hierarchical Trust

43

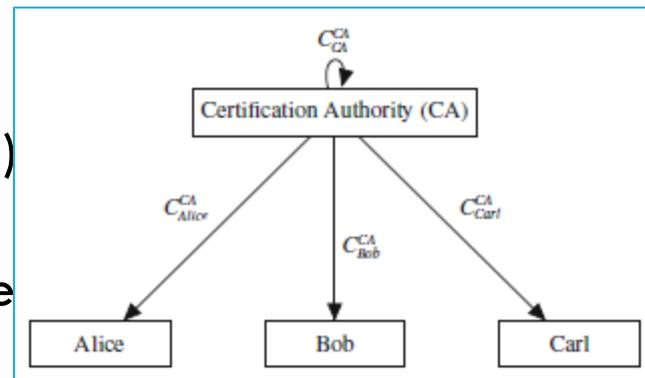
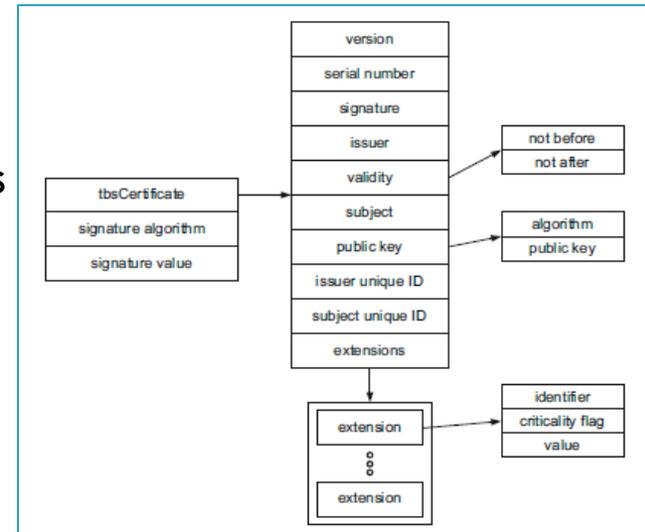
- In this simple hierarchical PKI, a single CA has issued certificates to the entities Alice, Bob, and Carl
- The CA is the trust anchor. It has generated a self-signed certificate, which is issued to Alice, Bob, and Carl too
  - ▣ The self-signing is depicted by a loop arrow from the CA to itself
- All entities in the PKI establish direct trust in the trust anchor
- Since the PKI users trust the trust anchor to sign certificates, the PKI users trust the authenticity of the public keys of Alice, Bob, and Carl, after validating their certificates
- Also, if entities outside the PKI trust the trust anchor and its public key, then they also accept the public keys of Alice, Bob, and Carl as authentic



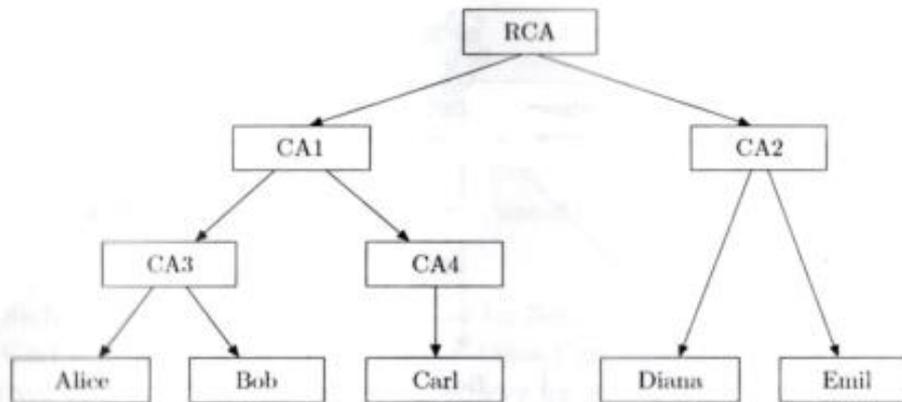
# Simple Hierarchical Trust Example

44

- Alice receives Bob's digital certificate (let's call it BDC) signed by the CA
- Alice checks the issuer section of BDC, which determines the CA being the issuer
- Alice has already a copy of the CAs self-signed certificate (let's call it CDC) and extracts the public key
  - ▣ Alice may even check the integrity of CDC in a similar way as she checks Bob's certificate below
- Alice validates that BDC has not expired
- She checks that the signature algorithm in BDC is compatible to CAs public key (e.g. RSA versus ECC)
- Alice decrypts BDC's signature value and compares it against the hash calculated over BDC excluding the signature value itself
- If both values match, the certificate and Bob's public key stored in it is valid
- Next, Alice validates Bob's authenticity via a challenge-response protocol

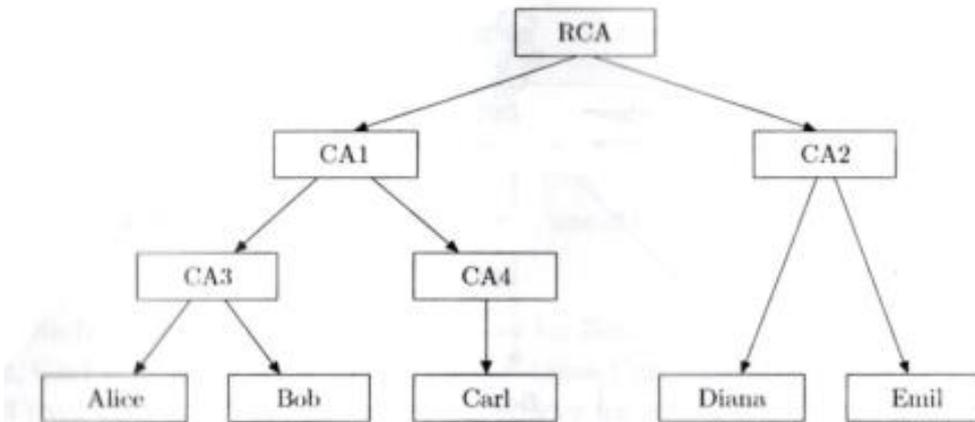


# CA Hierarchy I



- Assume a scenario, where multiple CAs provide certificates
- These CAs form a tree-like hierarchy with a “parent CA“ providing certificates for its “children”:
  - CA1 and CA2 are **intermediate** CAs whose certificates were signed by RCA
  - CA3 and CA4 are intermediate CAs whose certificates were signed by CA1
  - Alice and Bob have certificates signed by CA1
  - Carl’s certificate was signed by CA4
  - Diana’s and Emil’s certificate was signed by CA2
- Note that the leaves of this tree are **end-entities** (or end users)
- RCA could in principal sign end-entity certificates too
- End users and even CAs have no visibility of the entire CA hierarchy

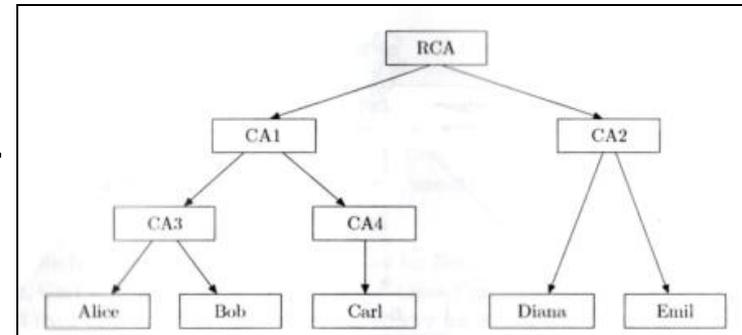
# CA Hierarchy II



- The RCA is the root of trust, and has a self-signed certificate
  - ▣ Remember that anybody could issue a self-signed cert to themselves!
- This RCA root certificate is distributed to all nodes in the hierarchy in a trustworthy fashion, for example via their
  - ▣ internet browser (a browser installation includes typically 200+ intermediate and root certificates) or
  - ▣ operating system installation

# CA Hierarchy III

- During operations, an endpoint may receive a certificate from another user that was signed by a CA unknown to them
  - E.g., Alice receives Emil's certificate that was signed by CA2
- Therefore, the user needs to get and validate the public key from an unknown CA (that is referenced in the received certificate), via a secure methodology, in order to validate the other user's certificate
  - E.g., Alice needs to acquire CA2's public key, and validate its authenticity, before validating Emil's certificate
- This process is called **Certification Path Construction**



# Certification Path Construction

48

- Consists of two phases:
  - ▣ Path construction

Involves building one or more *candidate* certification paths; "candidate" indicating that although the certificates may chain together properly, the path itself may not be valid for other reasons such as exceeding a maximum path length
  - ▣ Path validation

Involves making sure that each certificate in the path is within its established validity period, has not been revoked, and any constraints (e.g. maximum path length) are honoured

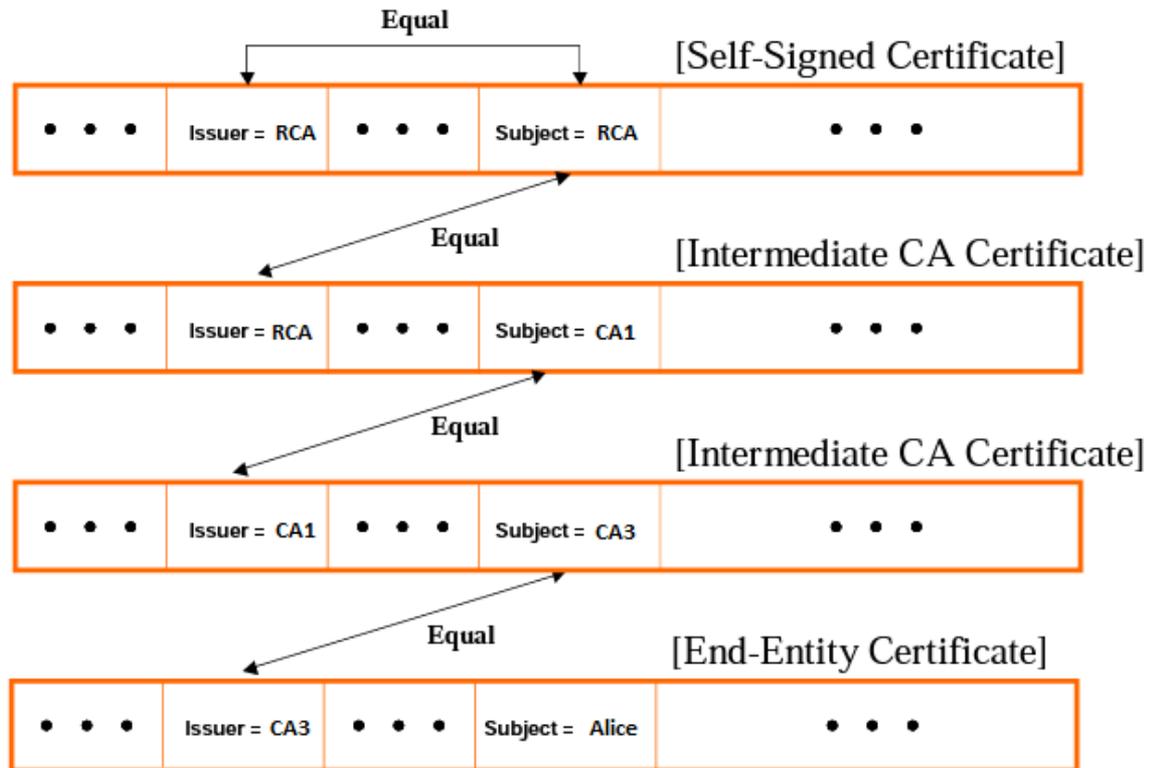
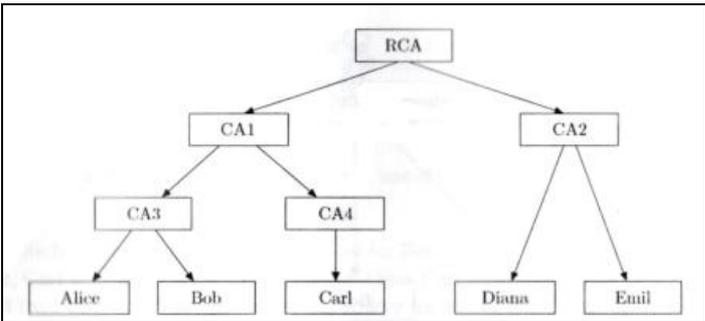
# Certification Path Construction via Name Chaining

49

- A candidate certification path must "name chain" between the recognised trust anchor (example RCA) and the target (example Alice's) certificate
- Working from the trust anchor to the target certificate, this means that the Subject Name in one certificate must be the Issuer Name in the next certificate in the path, and so on

# Name Chaining Example

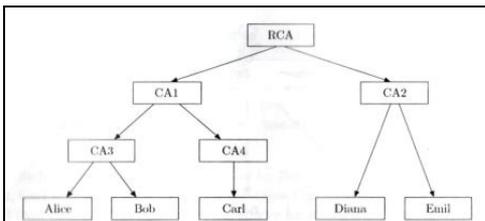
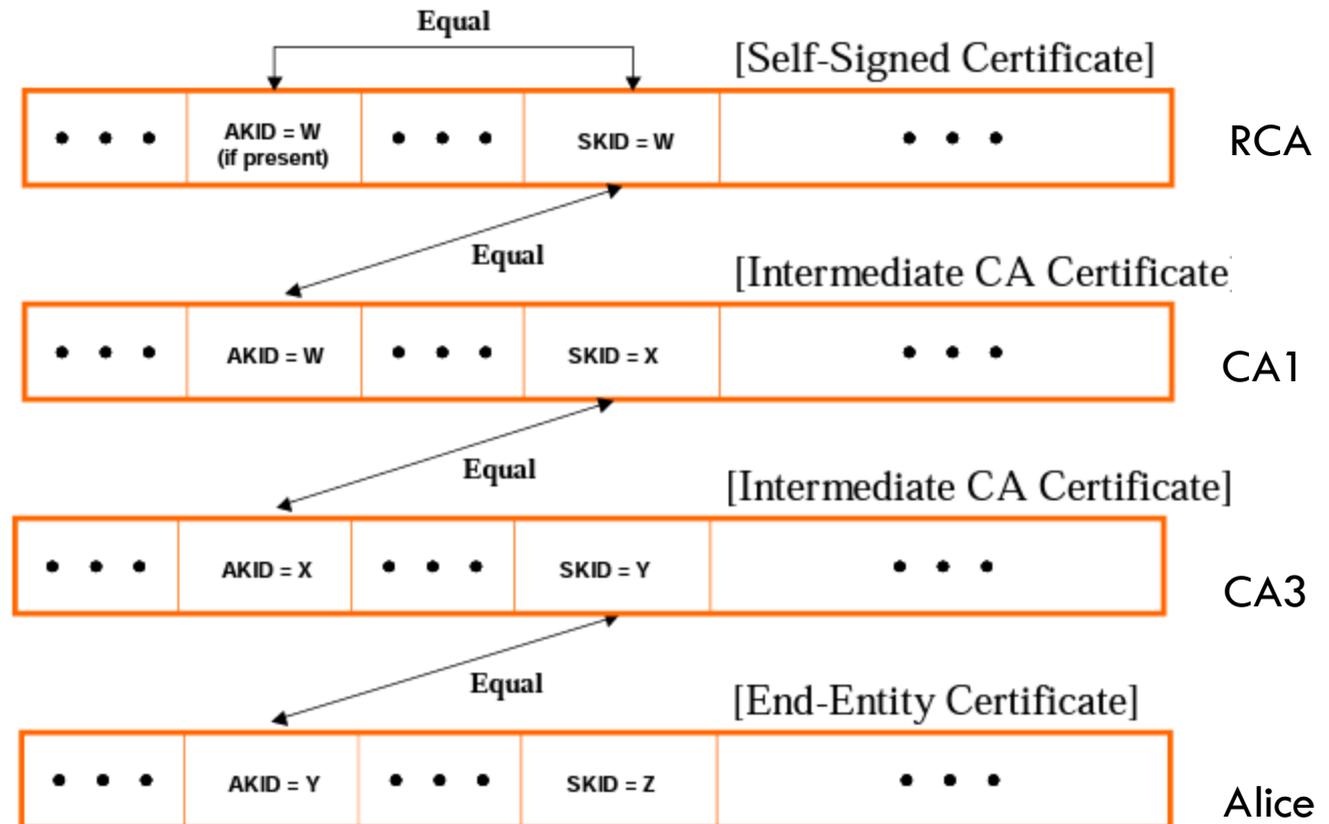
50



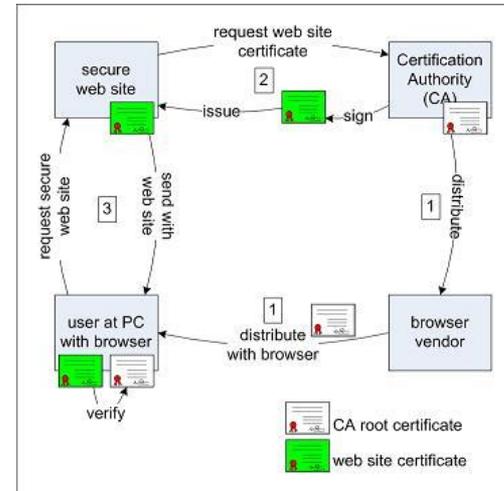
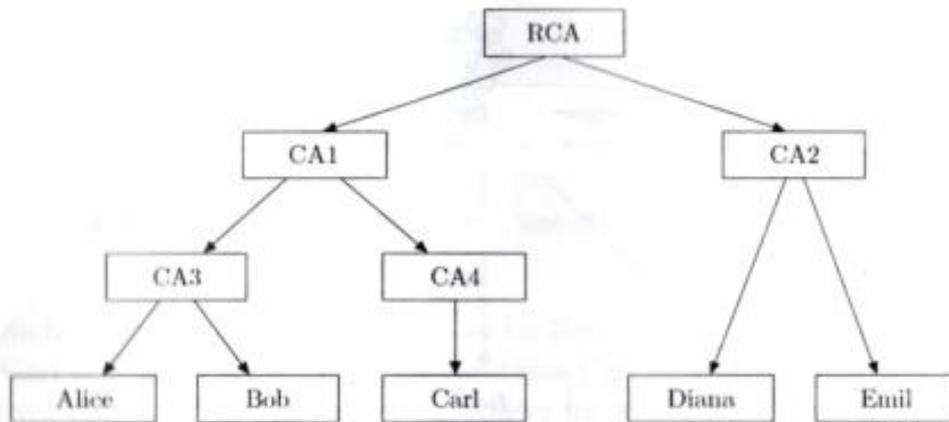
# Certification Path Construction via Key Identifier Chaining

51

- Recall certificate extensions *AuthorityKeyIdentifier* (AKID) and *SubjectKeyIdentifier* (SKID)



# Example Certificate Path Construction

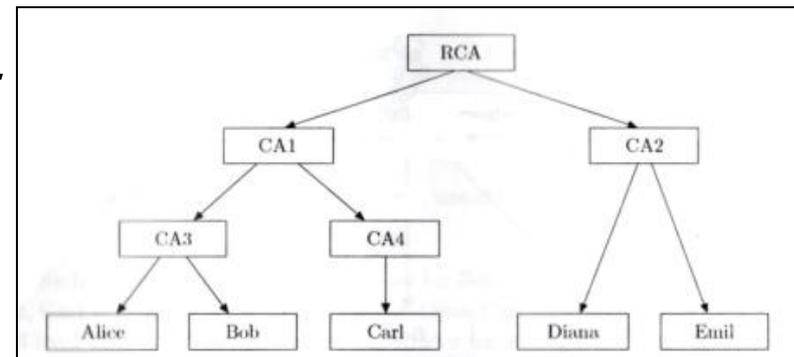


- Consider an example with
  - “Alice” (left) being “secure website” (right)
  - “Emil” (left) being “user at PC” (right)
  - “RCA” (left) being “Certificate Authority (CA)” (right)
- Emil sends a HTTPS connection request to Alice and receives a response containing her digital certificate
- Emil cannot validate Alice’s certificate directly, because it was signed by CA3 (and not RCA or CA2)
- However, if Emil can construct a Certification Path between Alice’s certificate and the RCA, he can validate Alice’s certificate (assuming he acknowledges the RCA as the root of trust)

# Certification Path Construction

53

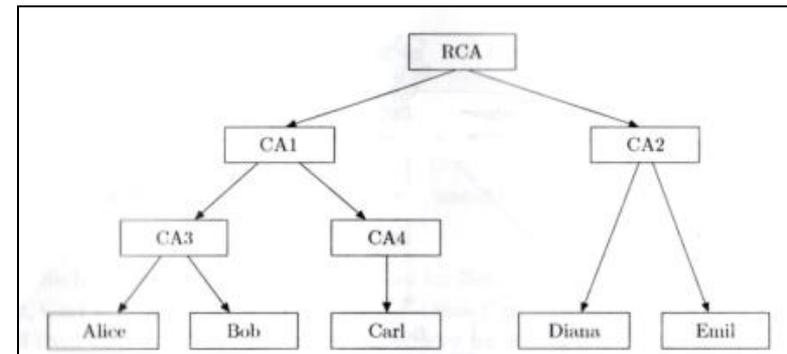
- In order for Emil to build the path, he must get copies of CA3's and CA1's certificates
  - ▣ RCA's self-signed cert is already in Emil's possession
- This can be done in 2 ways:
  1. Alice tags both certificates to hers and send all 3 of them to Emil
  2. Emil uses a directory service to retrieve both CA certificates, for example via LDAP (Lightweight Directory Access Protocol)



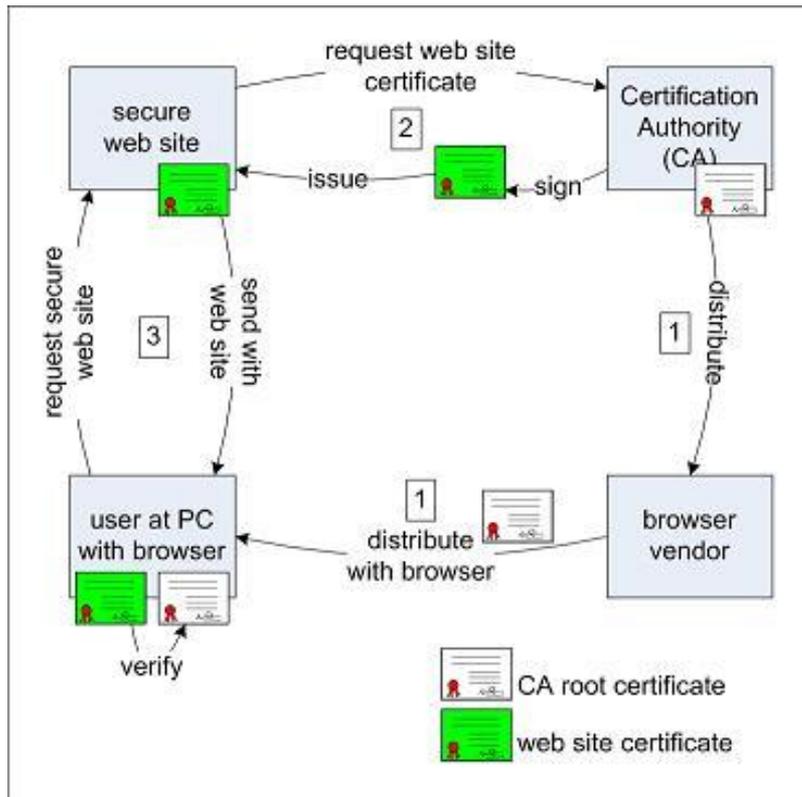
# Path Validation

54

- Now that Emil has a candidate path and all certificates, he must validate everything
  1. Firstly, Emil checks if all certificates have not expired yet (more later!)
  2. Then, using RCA's public key, he validates CA1's certificate as seen before
  3. If CA1's certificate is ok, Emil extracts its public key to validate CA3's certificate
  4. If CA3's certificate is ok, Emil extracts its public key to validate Alice's certificate
  5. If Alice's certificate is ok, and if her domain name (remember Alice is a secure website) matches the URL Emil entered, Emil goes ahead with the connection



# HTTPS Server Authentication Process (→ later)



- HTTPS is a secure version of HTTP
- In HTTPS, HTTP operates on top of TLS (Transport Layer Security), a secure transport layer protocol

# Basic Constraints

56

```
BasicConstraints ::= SEQUENCE {  
    cA                BOOLEAN DEFAULT FALSE,  
    pathLenConstraint INTEGER (0..MAX) OPTIONAL }
```

- ❑ Another X.509v3 extension...
- ❑ It is marked critical if the subject of the certificate is a CA
- ❑ cA is a Boolean value which is true if the certificate belongs to a CA and false otherwise
  - ▣ If this value is true, then the public key contained in the certificate can be used to verify signatures

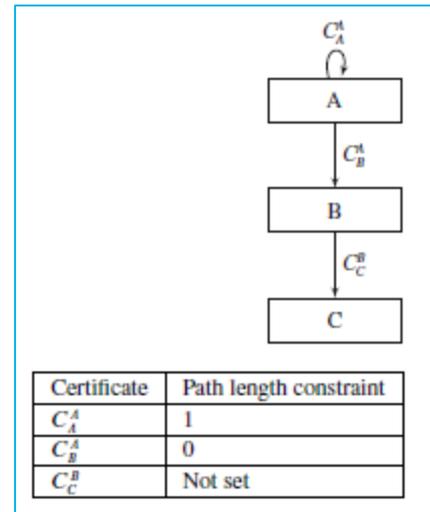
# Basic Constraints

57

□ It has two fields, the 2<sup>nd</sup> field:

```
BasicConstraints ::= SEQUENCE {  
  cA                BOOLEAN DEFAULT FALSE,  
  pathLenConstraint INTEGER (0..MAX) OPTIONAL }
```

- *pathLenConstraint* is used only for CA certificates in which the *cA* field is true and the *keyCertSign* bit is set in the key usage extension
- The value of this field is an integer; it sets a limit on the number of intermediate CA certificates that may be found after this certificate in the certification path before the path is invalid (i.e., when A generates B's certificate, it inserts its *pathLenConstraint* - 1
- Self-issued certificates do not count
- If such a limit is not desired, then this field is empty
- This parameter allows to limit the depth of a CA hierarchy

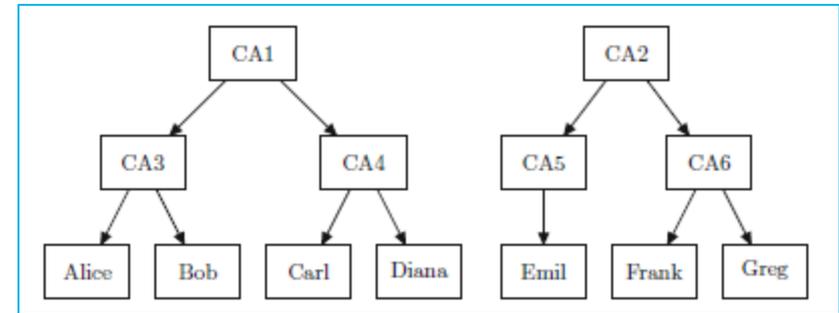


# Combining Trust Hierarchies: Trusted Lists

58

- Assume two independent PKIs with their own trust anchor
- How can Alice validate Greg's certificate?

- Solution 1: Trusted lists

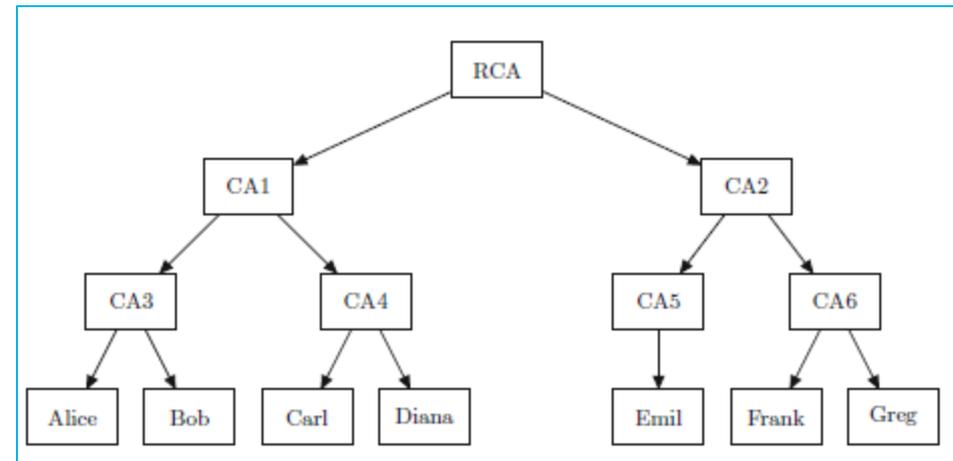


- Here Alice accepts CA2 as another trust anchor (note that her cert is signed by CA3 only)
  - CA2 cert is pre-installed on her browser / OS
- She is then able to construct a certification path (Greg – CA6 – CA2, potentially using a directory service), subsequently
  - validating CA6's cert using the public key in CA2's cert
  - validating Greg's cert using the public key in CA6's cert

# Combining Trust Hierarchies: Provide a common Root

59

- Here each end entity of the combined PKIs replaces its original trust anchor by the new common root

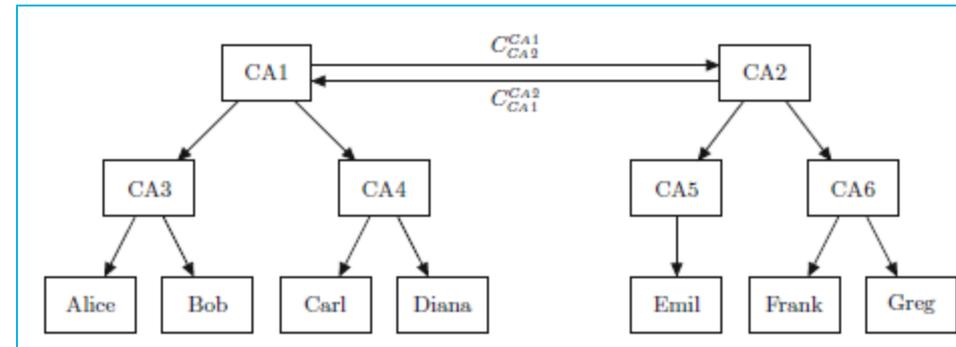


- As a consequence, certification paths that establish the authenticity of a public key have to be changed by prepending the common root

# Combining Trust Hierarchies: Cross Certification

60

- Cross-certification allows users of two PKIs to authenticate each other's public keys without replacing their trust anchors
- The idea is that the two root CAs certify each other's public keys using so-called **cross-certificates**
- In fact, the two CAs that cross-certify each other may also be only intermediate CAs
  - ▣ However, this implies that only the users covered by these CAs can validate each other's public keys
  - ▣ E.g. a single cross-certificate between CA4 and CA5 provides only interoperability between Carl, Diana and Emil



# Certificate Revocation

- The validity period of certificates may be quite long
  - ▣ For example, X.509 server certificates issued by SSL are typically valid for at least 2 years
- However, it may happen that during the validity period a certificate has to be invalidated
  - ▣ Example: the private key that corresponds to the public key in the certificate has been compromised
- The process of invalidating the certificate before its expiration time is called revocation

# Certificate Revocation Lists (CRL)

63

- A CRL is a list of revoked certificates which is digitally signed to prove its authenticity
- CRLs are regularly updated and made available at predictable points in time
  - ▣ When a CRL is updated, newly revoked certificates are inserted into the CRL
- There are direct CRLs and indirect CRLs:
  - ▣ Direct CRLs only contain certificates of one issuer and are issued and signed by that issuer
  - ▣ Indirect CRL may contain certificates of several issuers and is signed by the so-called CRL issuer
- Users who wish to obtain revocation information
  - ▣ download the CRL and verify its digital signature
  - ▣ check whether the certificate that they are interested in is contained in the CRL
- CRLs may become quite large since expired certificates are not always removed
- Therefore, delta CRLs have been introduced which only contain the certificates that have been revoked after the publication of the last full CRL
- The full CRL (i.e. complete CRL) contains all revoked certificates

# Online Certificate Status Protocol (OCSP)

64

- ❑ CRLs may become very large, downloading them becomes time consuming, and storing may need a lot of (unavailable) space
- ❑ Also, due to the potentially long time intervals between the publication of two subsequent lists, revocation information may not be up to date when it is used, in particular, shortly before the next update
- ❑ OCSP allows clients to query an OCSP server about the revocation status of individual certificates
- ❑ Here users may obtain revocation information immediately after the certificate is revoked
  - ▣ Unless of course the server just queries a CRL
- ❑ OCSP responses are digitally signed by the OCSP server, so they can be validated for their authenticity
- ❑ On the other hand, in contrast to the CRL method, OCSP requires the applications that need revocation information to be online

# Validity Models for Digital Signatures

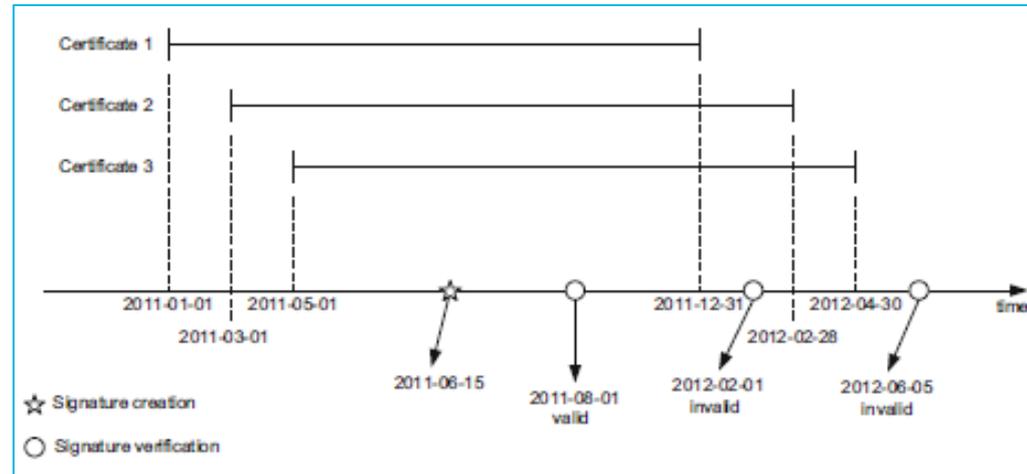
65

- Certificates in a validation path may have different expiry dates (because they were generated by different entities with different policies at different times), which poses the question, for how long an end-user certificate may be deemed valid, i.e. when does its path validation invalidate
- Simple example:
  - ▣ Assume Paul sells his house to Anna on 1 October 2023
  - ▣ Paul signs the sales contract digitally
  - ▣ The certificate that authenticates Paul's signature verification key expires on 31 July 2024
  - ▣ Should Paul's signature still be considered valid after his certificate has expired?

# The Shell Model

66

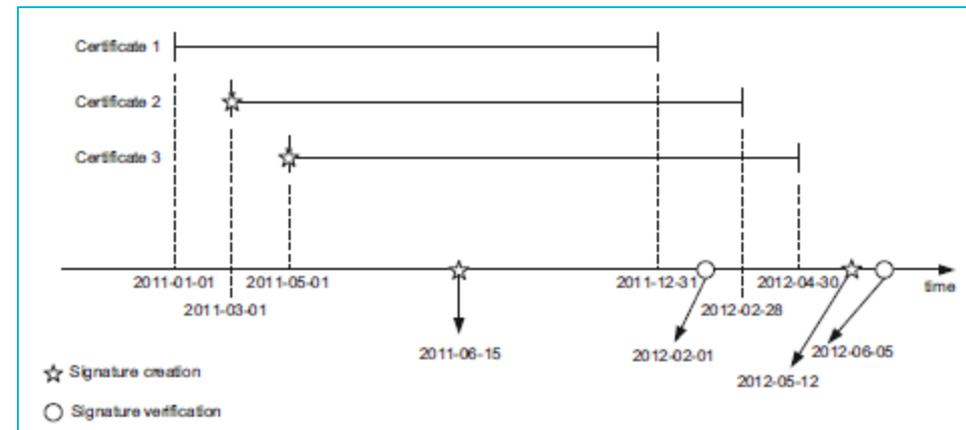
- In this model all certificates along the certification path must be valid when the signature is checked
- This model is appropriate in all applications, where signing and verification times are very close to each other
- Examples of such applications are
  - ▣ challenge-response authentication
  - ▣ mechanisms or email authentication
- However, for contract signing (with a legal binding long into the future) this model is inappropriate



# The Chain Model

67

- In the chain model the validity of a signature is independent of the verification time for this signature
- The chain model is often used for verifying legally binding electronic signatures because such signatures may be used for contract signing
- The chain model supports long validity periods for digital signatures
- However, it has certain drawbacks:
  - ▣ If Alice issues a signature and later a certificate in the chain that certifies Alice's verification key is revoked, the signature remains valid
  - ▣ This may have serious effects if the revocation reason is key compromise
- In the above example, the "2011-06-01" signature is valid at the point "2012-06-06", the signature "2012-05-12" is not



# PKI Architecture Components

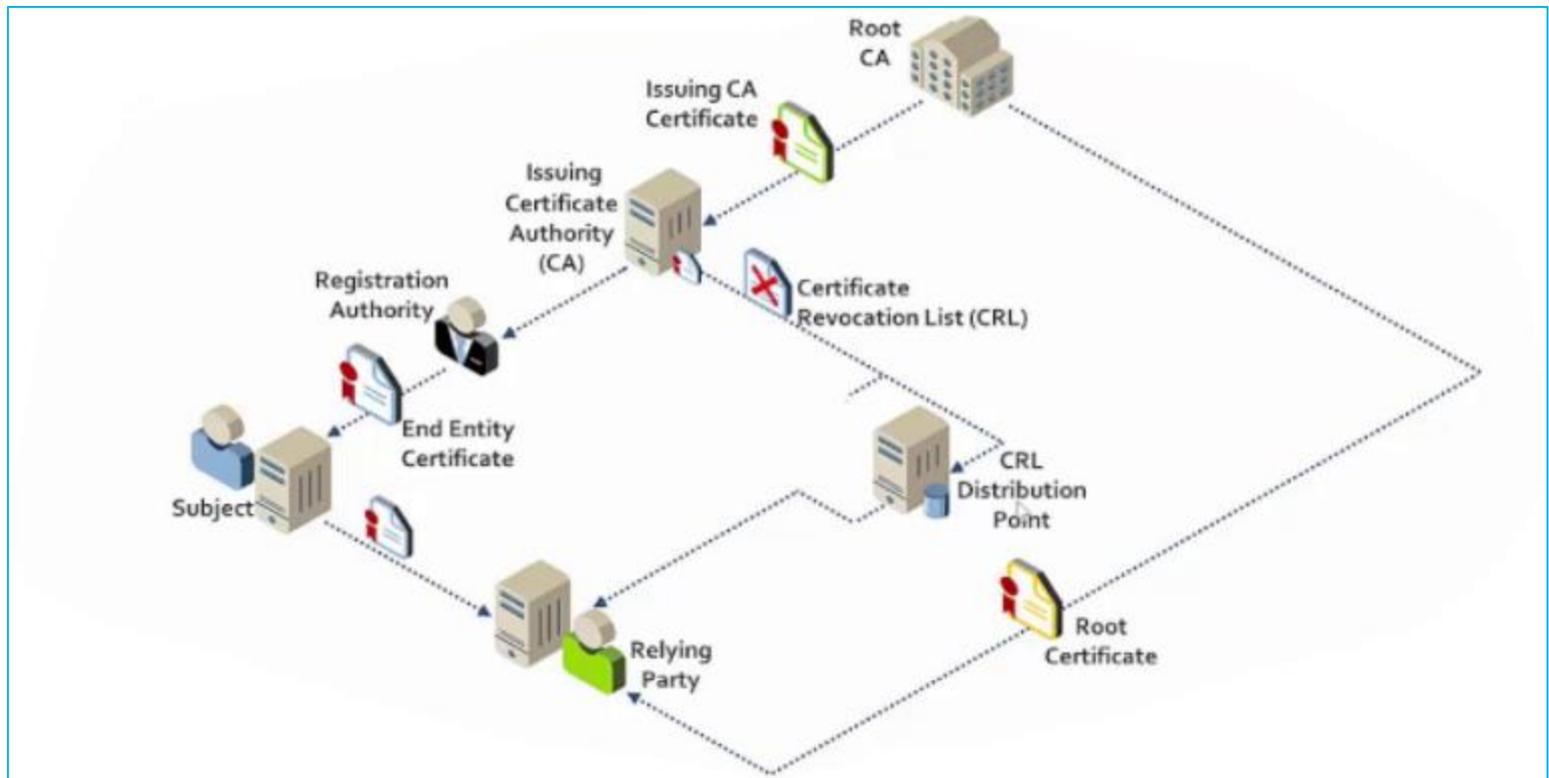
68

- ❑ A CA is a very well protected infrastructure that should only generate / sign certificates and CRLs
- ❑ Often, a RCA is only turned on on-demand (as a means of protecting it against attacks) to generate certificates for intermediate CA
- ❑ Such intermediate CA do all the signing work
  - ▣ It accepts CSRs (as seen before) from clients
- ❑ However, in order to reduce the attack surface of such a CA, client / end user communication including the processing of CSR, is done by a registration authority (RA)
- ❑ Similarly, CRL are distributed via dedicated CRL distribution points

# Example for a PKI Architecture

69

- Putting all components together, results in an architecture as shown below
- The Relying Party may be a web browser
- The Subject may be a web server



# FYI: ASN.1

70

- ❑ Abstract Syntax Notation One (ASN.1) is a standard interface description language for defining data structures that can be serialised and de-serialised in a cross-platform way
- ❑ Originally introduced to describe network data packets exchanged between endpoints, it is also widely used in cryptography and biometrics
- ❑ It is closely associated with a set of encoding rules that specify how to represent a data structure as a series of bytes, i.e.,
  - ❑ Basic Encoding Rule (BER)
  - ❑ Distinguished Encoding Rules (DER)
- ❑ Here encoded elements are typically type-length-value (TLV) sequences

# FYI: ASN.1 Basic Syntax

71

- ASN.1 is case sensitive
- Keywords start with capital letter
- Comments start with "--"
- The underscore is forbidden in identifiers and keywords
- Assignments use symbol "::="
- The top-level container of a type declaration is a module, e.g.

```
myModule DEFINITIONS ::= BEGIN
```

```
...
```

```
END
```

# FYI: ASN.1 Basic Syntax

72

- The available basic types are:
  - BOOLEAN
  - INTEGER
  - ENUMERATED
  - REAL
  - NULL
- Examples:
  - Automatic ::= BOOLEAN
  - Color ::= ENUMERATED {red, blue, green}
  - Pi REAL ::= 3.141
- Important: All types are abstract, e.g. there is no length or size associated with an INTEGER
- There are 3 types of strings (character, binary and hexadecimal), e.g.
  - IA5STRING ::= "Hello World" – International alphabet 5 with 7-bit characters
  - encryptionKey BIT STRING ::= '00100'B
  - encryptionKey OCTET STRING ::= 'ABC01'H

# FYI: ASN.1 Restricted Types

73

- Range:
  - ▣ Example: Age ::= INTEGER (0..50)
- Value set:
  - ▣ Example: Age ::= INTEGER {5, 10, 15, 20}
- Enumerated values
  - ▣ Example: Color ::= ENUMERATED {red(1), blue(2)}
- Default type
  - ▣ Example: Age ::= INTEGER DEFAULT 42

# FYI: ASN.1. Structured Types

74

- SEQUENCE
  - ▣ Like a struct in C
  - ▣ Example: See next slide
- SEQUENCE OF
  - ▣ Sequence of the same type
  - ▣ Example: `myCars ::= SEQUENCE OF Car`
- SET
  - ▣ Like a set
- SET OF
  - ▣ Set of the same type
- CHOICE
  - ▣ Similar to a union in C

# Example ASN.1 (Wikipedia)

75

Consider the following ASN.1 definition:

```
FooProtocol DEFINITIONS ::= BEGIN
  FooQuestion ::= SEQUENCE {
    trackingNumber INTEGER(0..199),
    question      IA5String
  }
  FooAnswer ::= SEQUENCE {
    questionNumber INTEGER(0..199),
    answer          BOOLEAN
  }
  FooHistory ::= SEQUENCE {
    questions SEQUENCE(SIZE(0..10)) OF FooQuestion,
    answers   SEQUENCE(SIZE(1..10)) OF FooAnswer,
    anArray   SEQUENCE(SIZE(100)) OF INTEGER(0..1000),
    ...
  }
END
```

Example for FooQuestion:

```
FooQuestion ::= SEQUENCE {
  trackingNumber INTEGER(5),
  question      "Anybody there?"
}
```

ASN.1 description of a simple application layer question / response protocol between a client and a server

# ASN.1 Encoding Formats

76

- There are three ASN.1 encoding formats:
  - ▣ Basic Encoding Rules (BER)
    - The original rules laid out by the ASN.1 standard for encoding data into a binary format
  - ▣ Canonical Encoding Rules (CER)
  - ▣ Distinguished Encoding Rules (DER)
- Both CER and DER are subsets of BER
  - ▣ Whereas BER gives choices as to how data values may be encoded, CER (together with DER) selects just one encoding from those allowed by the basic encoding rules
    - For example: In BER a Boolean value of true can be encoded as any positive integer up to 255, while in DER it has to be a 1

# BER Overview

- ❑ BER specifies a self-describing and self-delimiting format for encoding ASN.1 data structures
- ❑ Each data element is encoded as a type identifier, a length description, the actual data elements (TLV format), and, where necessary, an end-of-content marker
  - ❑ These types of encodings are commonly called type-length-value (TLV) encodings



First length octet

Form	Bits							
	8	7	6	5	4	3	2	1
Definite, short	0	Length (0–127)						
Indefinite	1	0						
Definite, long	1	Number of following octets (1–126)						
Reserved	1	127						

Long form example, length 435

	Octet 1								Octet 2								Octet 3							
	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	1	1	0	0	1	1
Long form	2 length octets								110110011 <sub>2</sub> = 435 <sub>10</sub> content octets															

# Some BER Identifier Octets and their Encodings (Wikipedia)

Name	Permitted construction	Tag number	
		Decimal	Hexadecimal
End-of-Content (EOC)	Primitive	0	0
BOOLEAN	Primitive	1	1
INTEGER	Primitive	2	2
BIT STRING	Both	3	3
OCTET STRING	Both	4	4
NULL	Primitive	5	5
OBJECT IDENTIFIER	Primitive	6	6
Object Descriptor	Both	7	7
EXTERNAL	Constructed	8	8
REAL (float)	Primitive	9	9
ENUMERATED	Primitive	10	A
EMBEDDED PDV	Constructed	11	B
UTF8String	Both	12	C
RELATIVE-OID	Primitive	13	D
TIME	Primitive	14	E

□ The identifier octets encode the ASN.1 tag's class number and type number

Octet 1							Octet 2 ... n Only if tag type > 30 <sub>10</sub>								
8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
Tag class			P/C		Tag type (if 0-30 <sub>10</sub> )			Long Form							
					31 <sub>10</sub> = Long Form			1=More		7 bits of Tag type					

Normally all 0

Identifier octets <i>Type</i>	Length octets <i>Length</i>	Contents octets <i>Value</i>	End-of-Contents octets <i>(only if indefinite form)</i>
----------------------------------	--------------------------------	---------------------------------	--

# Example BER Encoding (Wikipedia)

79

Consider the following ASN.1 definition:

```
FooProtocol DEFINITIONS ::= BEGIN
  FooQuestion ::= SEQUENCE {
    trackingNumber INTEGER(0..199),
    question IA5String
  }
  FooAnswer ::= SEQUENCE {
    questionNumber INTEGER(10..20),
    answer BOOLEAN
  }
  FooHistory ::= SEQUENCE {
    questions SEQUENCE(SIZE(0..10)) OF FooQuestion,
    answers SEQUENCE(SIZE(1..10)) OF FooAnswer,
    anArray SEQUENCE(SIZE(100)) OF INTEGER(0..1000),
    ...
  }
END
```

The FooQuestion structure “5Anybody there?” encoded in DER format:

```
30 13 02 01 05 16 0e 41 6e 79 62 6f
64 79 20 74 68 65 72 65 3f
```

with

- 30 — type tag indicating SEQUENCE
- 13 — length in octets of value that follows
- 02 — type tag indicating INTEGER (see previous slide)
- 01 — length in octets of value that follows
- 05 — value (5)
- 16 — type tag indicating IA5String (i.e. ASCII)
- 0e — length in octets of value that follows
- 41 6e 79 62 6f 64 79 20 74 68 65 72 65 3f (“Anybody there?” in plain ASCII format)

# ASN.1 Encoding of OIDs

80

- Practically, OIDs need to be encoded as TLV triplets
- The TLV triplet begins with a Tag value of 0x06 (see table on the right)
- Each OID integer (i.e., node) is encoded as follows:
  - ▣ The first two nodes of the OID are encoded onto a single byte, by multiplying the first node with 40 and adding the result to the value of the second node
  - ▣ Subsequent bytes are represented using **Variable Length Quantity**, also called base 128

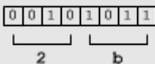
Name	Permitted construction	Tag number	
		Decimal	Hexadecimal
End-of-Content (EOC)	Primitive	0	0
BOOLEAN	Primitive	1	1
INTEGER	Primitive	2	2
BIT STRING	Both	3	3
OCTET STRING	Both	4	4
NULL	Primitive	5	5
OBJECT IDENTIFIER	Primitive	6	6
Object Descriptor	Both	7	7
EXTERNAL	Constructed	8	8
REAL (float)	Primitive	9	9
ENUMERATED	Primitive	10	A
EMBEDDED PDV	Constructed	11	B
UTF8String	Both	12	C
RELATIVE-OID	Primitive	13	D
TIME	Primitive	14	E

# Example: BER Encoding of an OID

81

**OID:**  
1.3.6.1.4.1.311.21.20 (ClientId Attribute)

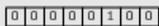
**1) Encoding the First Two Nodes:**

$1 \times 40 + 3 = 43d = 0x2B =$  

**2) Single byte encoding of all remaining nodes other than 311:**

$3 = 0x03 =$  

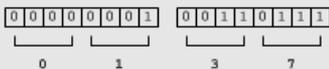
$6 = 0x06 =$  

$4 = 0x04 =$  

$21 = 0x15 =$  

$20 = 0x14 =$  

**3) Multiple byte encoding of 311:**

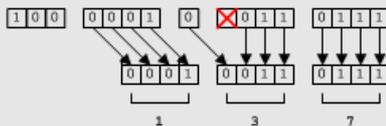
$311d = 0x0137 =$  

This is encoded to 0x82 0x37 by:

- 1) Setting bit 7 of the leftmost byte to 1.
- 2) Ignoring bit 7 of the rightmost byte.
- 3) Shifting the right nibble of the leftmost byte to the left by 1 bit.



When decoded, the bits are assembled in the following manner:



**3) Summary encoding of OID : 1.3.6.1.4.1.311.21.20**

0x2B 0x06 0x01 0x04 0x01 0x82 0x37 0x15 0x14

- This example shows how the *ClientId* attribute (OID: 1.3.6.1.4.1.311.21.20) of a certificate request is encoded: 1.3.6.1.4.1.311.21.20vich3d.jdomcsc.nettest.microsoft.comJDOMCSCadministratorcertreq”

```

06 09 ; OBJECT_ID (9 Bytes)
| 2b 06 01 04 01 82 37 15 14 ; 1.3.6.1.4.1.311.21.20
31 4a ; SET (4a Bytes)
 30 48 ; SEQUENCE (48 Bytes)
  02 01 ; INTEGER (1 Bytes)
  | 09
 0c 23 ; UTF8_STRING (23 Bytes)
  | 76 69 63 68 33 64 2e 6a ; vich3d.j
  | 64 6f 6d 63 73 63 2e 6e ; domcsc.n
  | 74 74 65 73 74 2e 6d 69 ; ttest.mi
  | 63 72 6f 73 6f 66 74 2e ; crossoft.
  | 63 6f 6d ; com
 0c 15 ; UTF8_STRING (15 Bytes)
  | 4a 44 4f 4d 43 53 43 5c ; JDOMCSC\
  | 61 64 6d 69 6e 69 73 74 ; administ
  | 72 61 74 6f 72 ; rator
 0c 07 ; UTF8_STRING (7 Bytes)
  63 65 72 74 72 65 71 ; certreq
    
```

# Base64 Encoding

82

- Problem: How can BER encoded binary data (including certificates) be stored or transported in channels that only reliably support (readable) text content?
- Examples:
  - ▣ Embedding (binary) images inside textual assets such as HTML and CSS files
  - ▣ Embedding attachments (e.g. images) in emails
- Solution: Apply a binary-to-text encoding scheme, e.g. Base64

# Base64 Encoding

- Base64 divides a binary input into 6-bit snippets, with each snippet represented by a printable character
- Example Base64 table from RFC 4648:

Index	Binary	Char	Index	Binary	Char	Index	Binary	Char	Index	Binary	Char
0	000000	A	16	010000	Q	32	100000	g	48	110000	w
1	000001	B	17	010001	R	33	100001	h	49	110001	x
2	000010	C	18	010010	S	34	100010	i	50	110010	y
3	000011	D	19	010011	T	35	100011	j	51	110011	z
4	000100	E	20	010100	U	36	100100	k	52	110100	0
5	000101	F	21	010101	V	37	100101	l	53	110101	1
6	000110	G	22	010110	W	38	100110	m	54	110110	2
7	000111	H	23	010111	X	39	100111	n	55	110111	3
8	001000	I	24	011000	Y	40	101000	o	56	111000	4
9	001001	J	25	011001	Z	41	101001	p	57	111001	5
10	001010	K	26	011010	a	42	101010	q	58	111010	6
11	001011	L	27	011011	b	43	101011	r	59	111011	7
12	001100	M	28	011100	c	44	101100	s	60	111100	8
13	001101	N	29	011101	d	45	101101	t	61	111101	9
14	001110	O	30	011110	e	46	101110	u	62	111110	+
15	001111	P	31	011111	f	47	101111	v	63	111111	/
Padding		=									

# Base64 Encoding Examples (Wikipedia)

84

- “Many hands make light work” is converted into  
TWFueSBoYW5kcyBtYWtlIGxpZ2h0IHdvcmsu
- Generally, 3 bytes are converted into 4 printable  
Base64 characters (with padding character “=” added  
if input length is not multiple of 3), as follows:

Source	Text (ASCII)	M			a			n																	
	Octets	77 (0x4d)			97 (0x61)			110 (0x6e)																	
Bits		0	1	0	0	1	1	0	1	0	1	1	0	0	0	0	1	0	1	1	0	1	1	1	0
Base64 encoded	Sextets	19			22			5			46														
	Character	T			W			F			u														
	Octets	84 (0x54)			87 (0x57)			70 (0x46)			117 (0x75)														

# Example: Base64 Encoded Certificate Signing Request (more later)

85

```
-----BEGIN NEW CERTIFICATE REQUEST-----  
MIICkzCCAXsCAQAwTjELMAkGA1UEBhMCQ0ExCzAJBgNVBAGTAmdmMQswCQYDVQQH  
EwJnZjELMAkGA1UECxmCZ2YxCzAJBgNVBAoTAmdmMQswCQYDVQQDEwJnZjCCASlw  
DQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAMlwsZXhim1CYsCcz5MOwHlLhkxU  
3KAEhr1pg3tOPmzImuXTnWWt4sDb//fsadcZ9EBInUMoRurTLLo8TuNnNhAlkGD0  
9PPSEZPb+loYLASA8DG4SkRyrl2sVhIVmzq8w7/zp561ur5m3wV+c5ru3W/CvjdT  
Z78ReIUTlul2nCJ46PQIYky+2IPGtj+VY/9IDe+iXLs9i/u7k2oppBo70qdzR3vR  
hml55noblmeUcVL21w2jMTz6nZAnsatt4fnrAgM6ZmNzXyaoj3PNWoBYtSBuiYe  
QArBhiOpR1Og9E2XGOvbsyc4+ORNWPSfX0H4uFYZNAS5n4fBrFTskJ9MKEUCAwEA  
AaAAMA0GCSqGSIb3DQEBAQUAA4IBAQCTLS7EWjqVewqrotQ5NZa8IXIFSoGaNOeU  
WVJoXWUIkhd6CSOgxXiDdYIDIVe1EUGUQ5Lx9bVnniBy0F7ssUFBgehG6maxWrq7  
AEPFQESgfsEYH6JGvhZM1Fa9WjxaCiOXpozP1SIF9j6RzNvJudxpDOd80RSjoifg  
f4QXNfdW1fpXa56ED2NBgozXb1IWeu/Kb2JU7AlUmY6Xde1tAyW5I7gIbFapAacv  
//edvQZm1Zfq0/CVSKhxwcg8K8gf1rLfgTNPz7FbvGhDO9YFir7qVK1xx7HEaBe9  
BkQqxArSzTCtKpFbNPQ+A6mxBnVXXFhEOtNeaU/foq0k7l+3k9LD  
-----END NEW CERTIFICATE REQUEST-----
```

□ See <http://lapo.it/asn1js/>

CT437

COMPUTER SECURITY AND FORENSIC COMPUTING

SECURE NETWORK COMMUNICATION PRINCIPALS

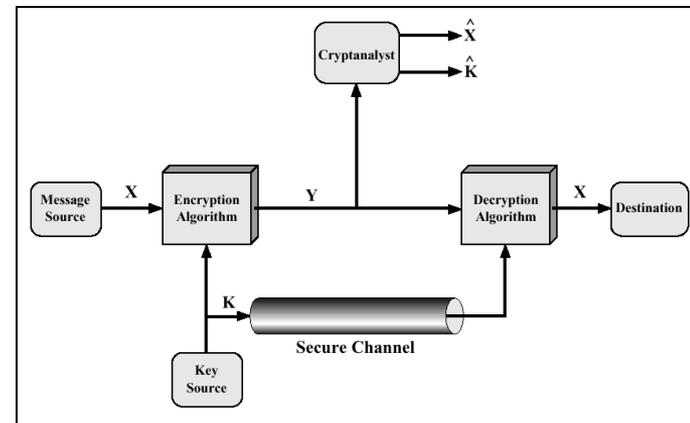
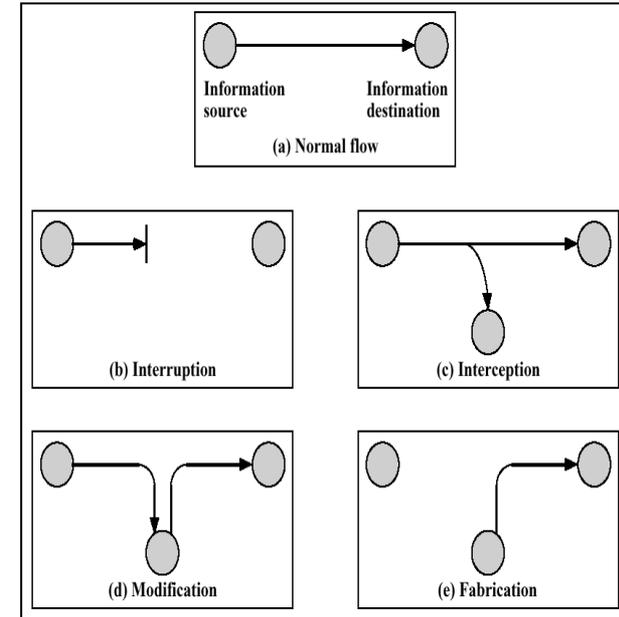
Dr. Michael Schukat



# Lecture Overview

2

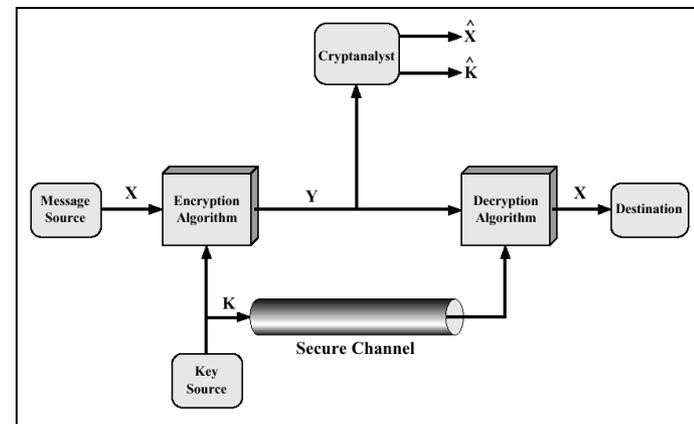
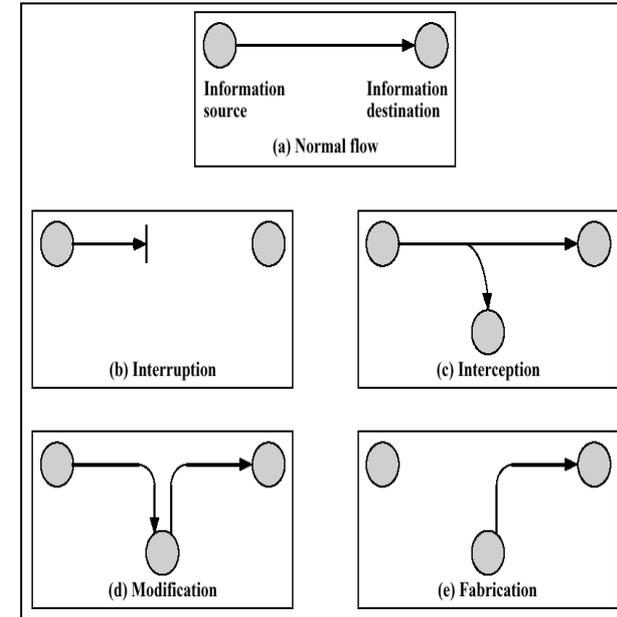
- This lecture will look in more detail into how data encryption and hashing mechanisms can be applied to provide secure peer-to-peer data communication over an (unsecure) network



# Lecture Overview

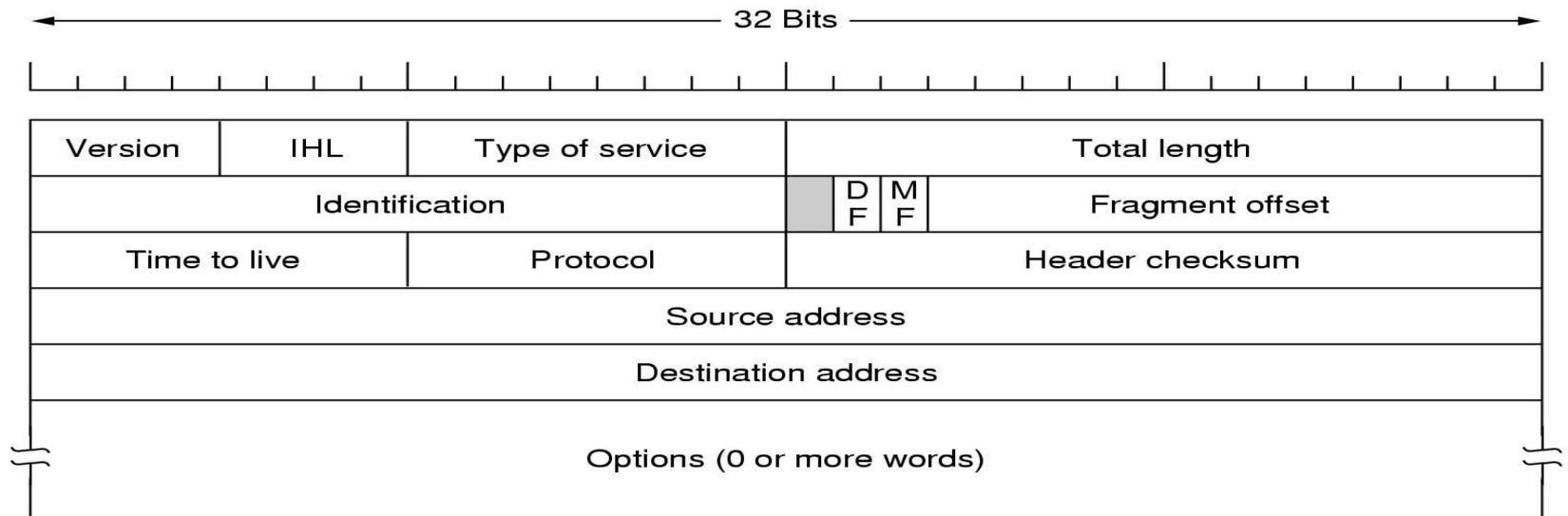
3

- Symmetric block and stream ciphers allow the encryption of data in transit
  - ▣ Needs robust key management and distribution
- Hash functions / MACs allow authentication of data in transit
- Digital certificates allow end-point authentication
- Hashing and encryption provide mechanisms to address some of security attack types on information in transit
  - ▣ Example Wireshark
- This lecture will look in more detail into how these mechanisms can be applied to provide secure peer-to-peer data communication over a network

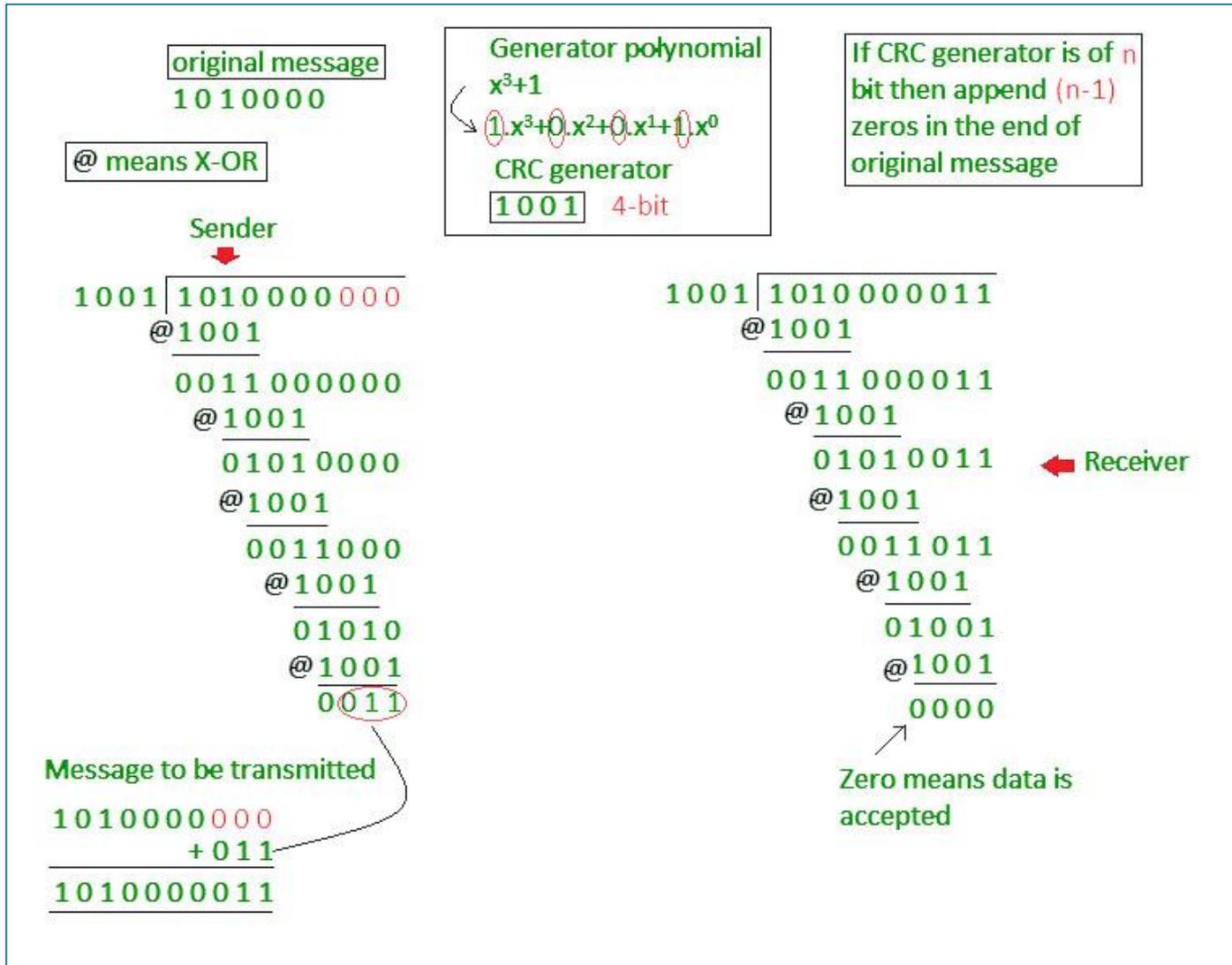


# Issues with the IP Protocol

- ❑ IP payload is not encrypted (no confidentiality) and can be manipulated in transit
- ❑ IP header fields can be manipulated in transit (CRC can be adjusted on-the-fly → next slide)
  - ❑ IP addresses can be spoofed (no authentication)
- ❑ IP header has mutable fields that can change during datagram transport

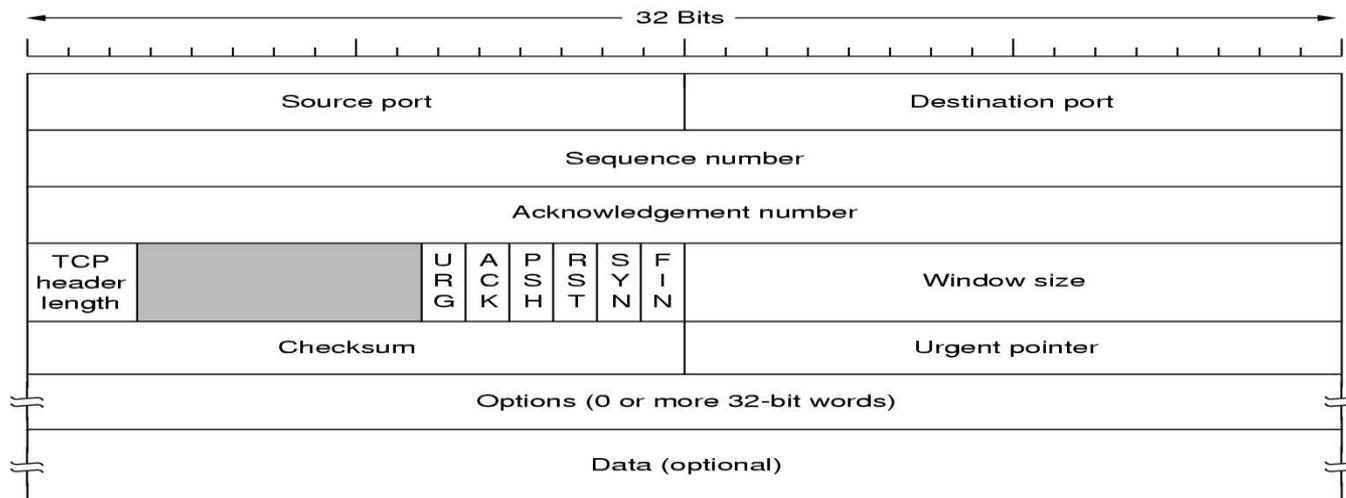


# Recap: Cyclic Redundancy Check (CRC)



# Issues with the Transport Layer Protocol (Example TCP)

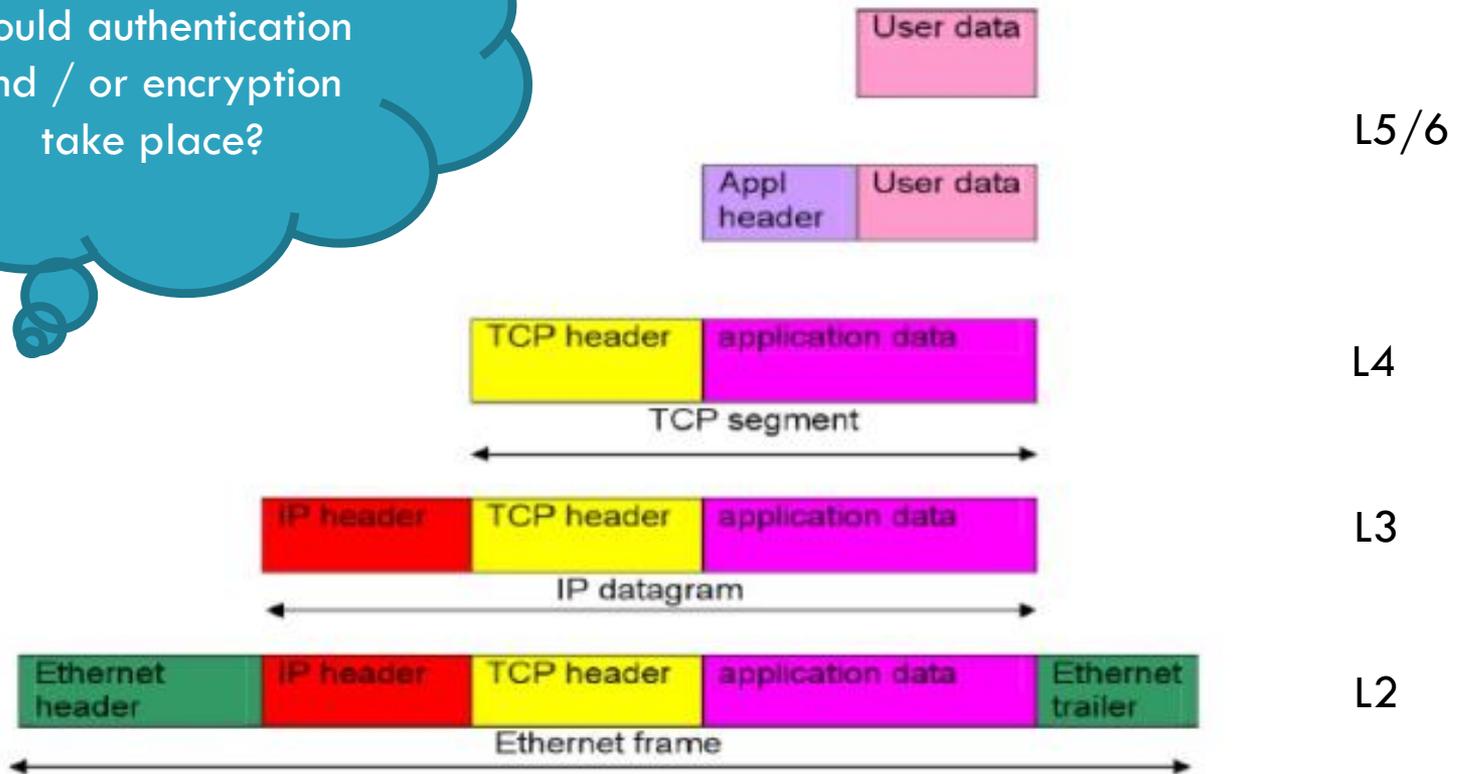
- TCP payload is not encrypted (no confidentiality) and can be manipulated in transit
- TCP header fields can be manipulated in transit (CRC can be adjusted)
  - TPDU's can be rearranged in transit via manipulating sequence and acknowledgement numbers



# TCP/ IP Header Hierarchy

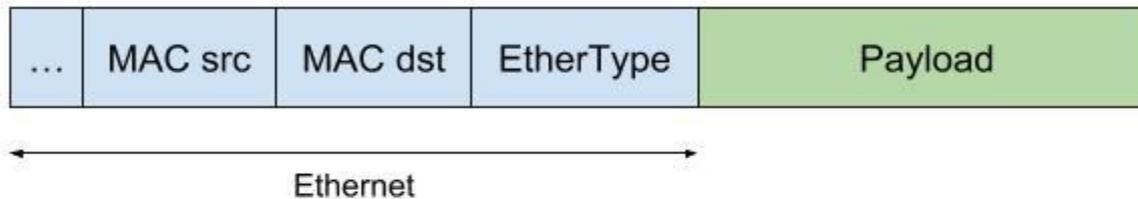
7

On what level(s) should authentication and / or encryption take place?

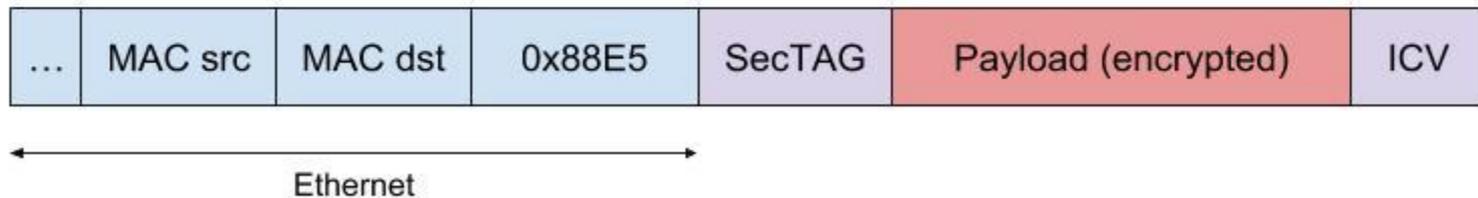


# Example MACsec

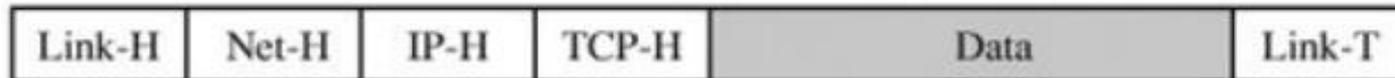
## Ethernet frame and its payload



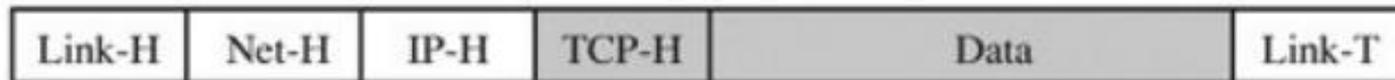
## Ethernet frame and its payload, using MACsec (encryption enabled)



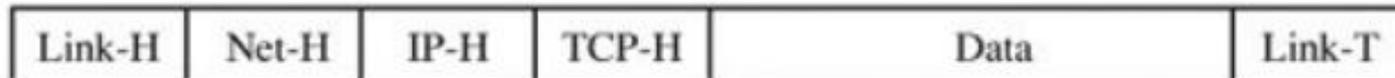
# Encryption Coverage Implications



(a) Application-level encryption (on links and at routers and gateways)



On links and at routers

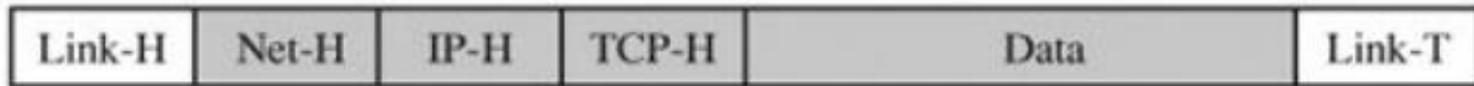


In gateways

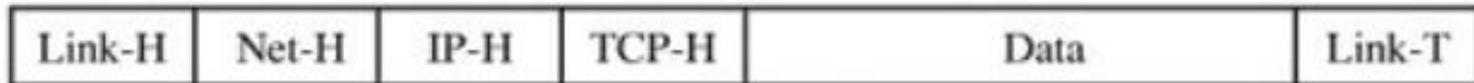
(b) TCP-level encryption

- TCP-H - TCP header
- IP-H - IP header
- Net-H - Network-level header (e.g., X.25 packet header, LLC header)
- Link-H - Data link control protocol header
- Link-T - Data link control protocol trailer

# Encryption Coverage Implications



On links



In routers and gateways

(c) Link-level encryption

Shading indicates encryption.

- TCP-H = TCP header
- IP-H = IP header
- Net-H = Network-level header (e.g., X.25 packet header, LLC header)
- Link-H = Data link control protocol header
- Link-T = Data link control protocol trailer

Example for an unsecure network security protocol

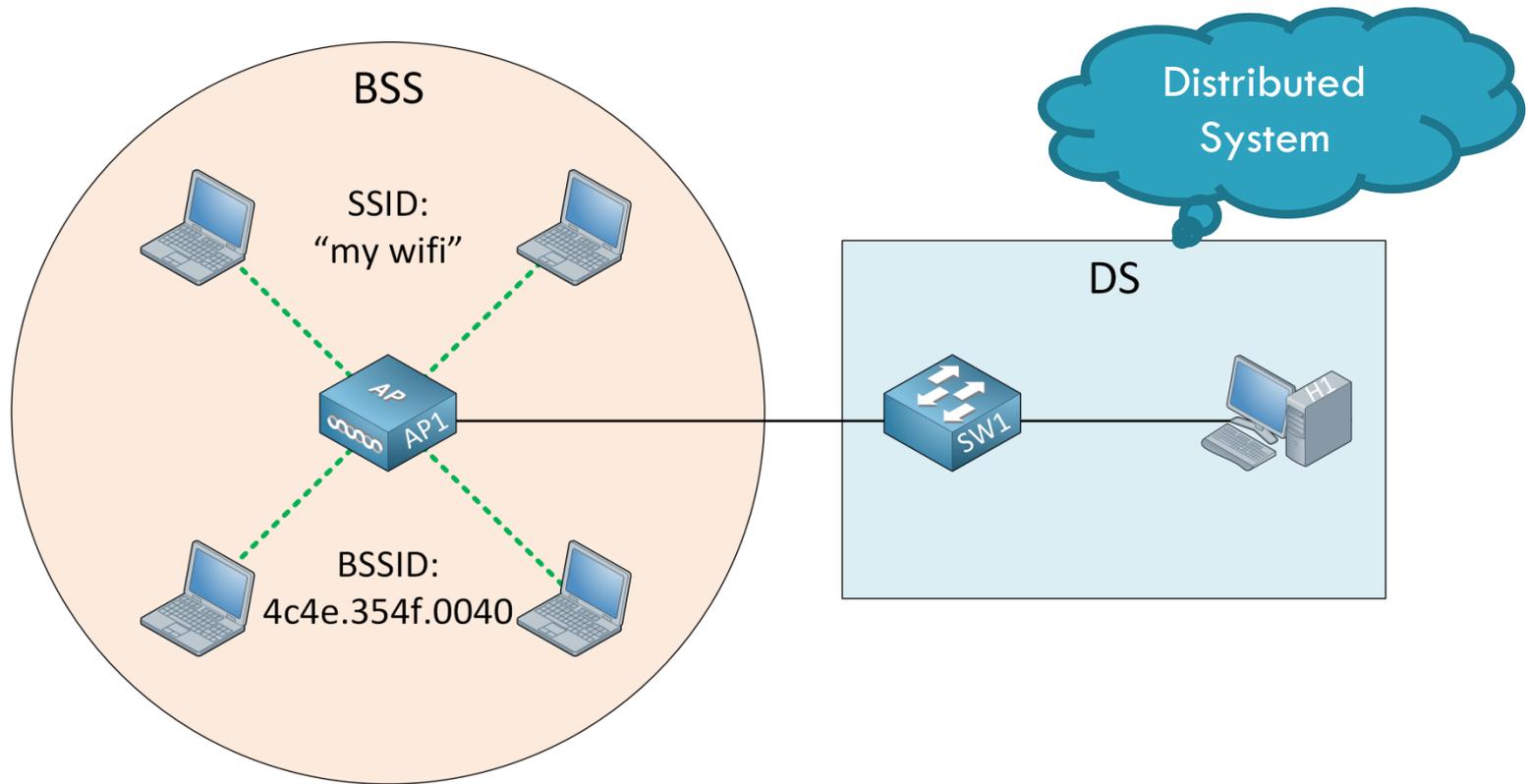
# Wire Equivalent Privacy (WEP)

- The first attempt of encrypting 802.11 (Wi-Fi) communication
- It was the de-facto 802.11 security protocol for a couple of years, implemented in all Wi-Fi routers at the time
- However, it has a flawed design and has been broken in the early 2000s
  - ▣ It is completely obsolete by now – Don't use it!
- Nonetheless it makes a good case study...

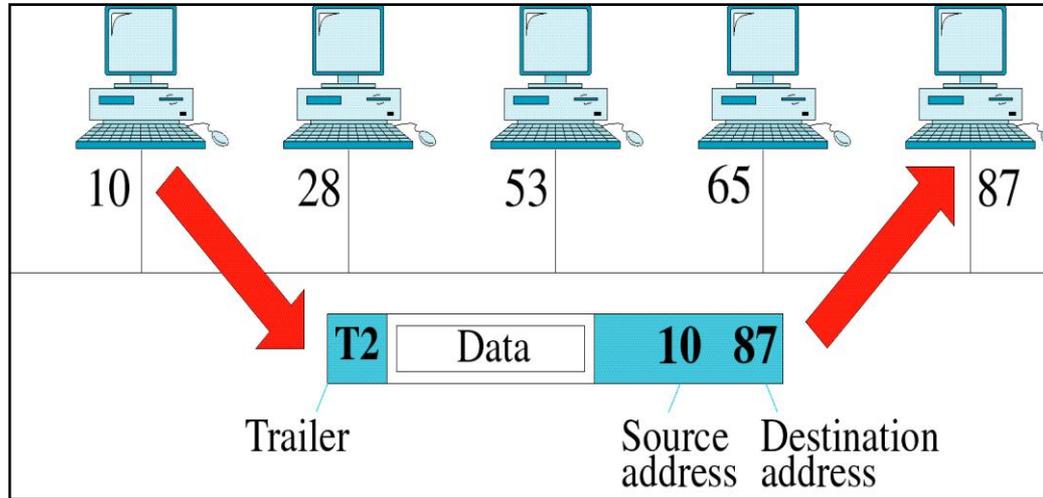
# 802.11 Summary

- Wireless network protocol, operates on 2.4 GHz or 5 GHz carrier frequency
- The base version of this IEEE standard was released in 1997, with various amendments since
- In the common *infrastructure mode* networks are organised as wireless network basic service set (BSS)
- A BSS consists of one redistribution point (i.e., an access point) together with one or more client stations that are associated with it
- Each BSS has a
  - unique id (BSSID), like a 48 medium access sublayer (MAC) address
  - Customisable name, the Service Set ID (SSID)
- 802.11 is based on the exchange of plaintext messages and as such prone to Wi-Fi eavesdropping too (→ Wireshark)

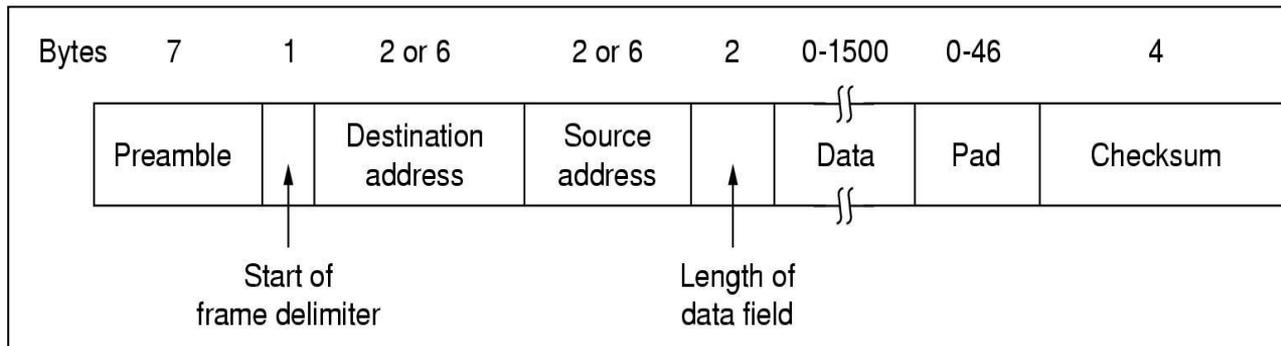
# BSS, BSSID and SSID



# Recall: The 802.3 MAC Sublayer Protocol



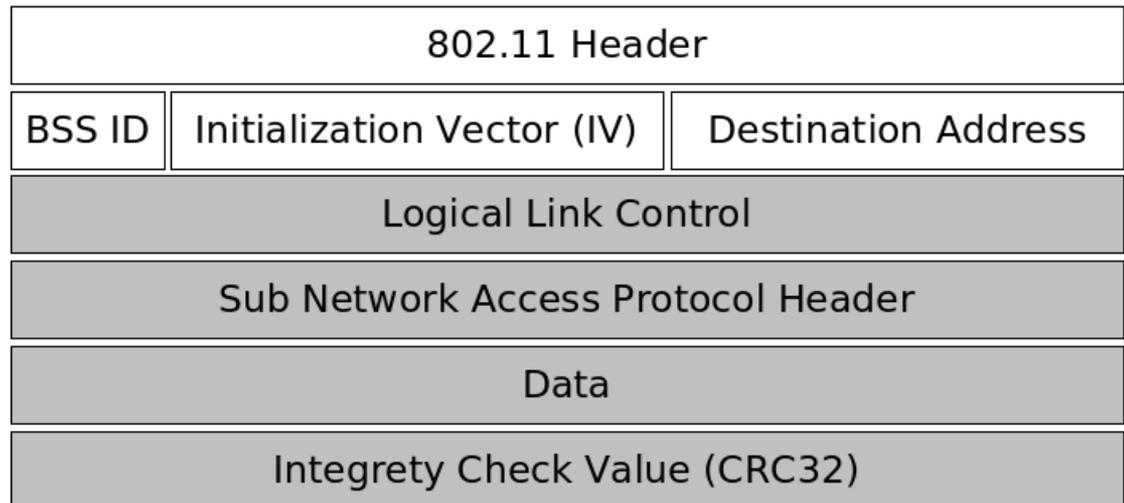
- Simpler than 802.3 packet structure:



# WEP Overview

16

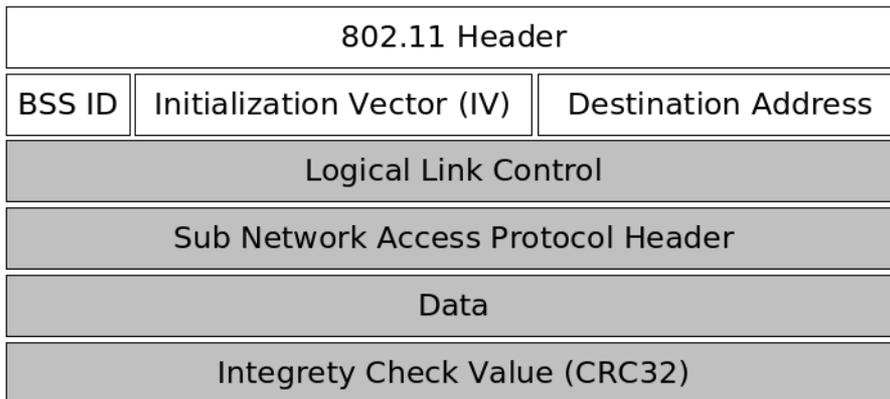
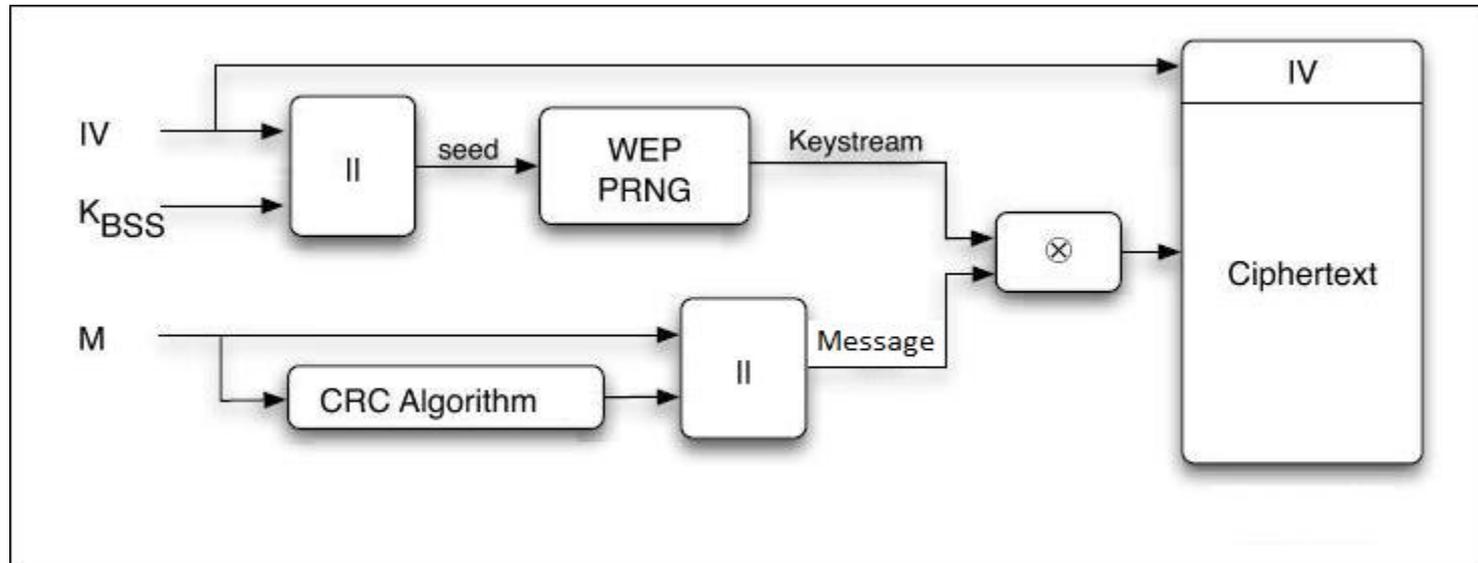
- ❑ WEP was ratified as a Wi-Fi security standard in 1999
- ❑ Two main flavours,
  - ▣ WEP-40 (40-bit secret key plus 24-bit shared initialisation vector), i.e., 64-bit WEP
  - ▣ WEP-104 (104-bit secret key plus 24-bit shared initialisation vector), i.e., 128-bit WEP
- ❑ WEP uses
  - ▣ the stream cipher RC4 for confidentiality
  - ▣ the CRC-32 checksum for integrity
- ❑ Both flavours were deprecated in 2004 (!)
- ❑ The WEP header is shown on the right with encrypted sections highlighted in dark
  - ▣ Note the (24-bit) plaintext initialisation vector is incremented with every packet



# WEP Encoding

- A secret BSS key  $K_{BSS}$  (40 bit or 104 bit) is shared between the AP and all clients
- Every Wi-Fi packet contains a random 24-bit initialisation vector IV chosen by the sender
- $IV \parallel K_{BSS}$  is the seed for an RC4 stream cipher (WEP PRNG)
- The payload  $M$  is complemented by a 32-bit CRC (cyclic-redundancy-checksum) and bitwise EXORed with the key stream
- The resulting encrypted message is complemented with the IV and transmitted

# WEP Encoding



} Encrypted / authenticated

# Recap: RC4 as used in WEP

- RC4 is a stream cipher
- It consists of a
  - ▣ key-scheduling algorithm (KSA) and a
  - ▣ pseudo-random generation algorithm (PRGA)
- The KSA uses  $IV \parallel K_{BSS}$  as a key to initialise the algorithm
- Subsequently the PRGA returns pseudo-random byte at a time

# WEP Weaknesses

20

- Because RC4 is a stream cipher, the same traffic key must never be used twice
- The purpose of an IV, which is transmitted as plain text, is to prevent any repetition, but a 24-bit IV is not long enough to ensure this on a busy network
  - ▣ 16,777,216 different RC4 cipher streams when the IV is just incremented
  - ▣ Even worse, when a new IV is randomly picked for each packet, there is a 50% probability the same IV will repeat after 5,000 packets (Birthday paradox)
- There's a range of WEP attacks that takes advantage of that

# Summary

21

- Network security (i.e., data encryption and / or authentication) is important for obvious reasons
- The layered structure of the TCP/IP stack allows positioning the extra security layer in different levels
- Each of these options has its advantages and disadvantages / limitations, for example with regard to
  - ▣ the portions of a packet that can be secured
  - ▣ compatibility with network routing, NAT, etc.
- WEP as a much weaker and depreciated option shows how encryption / authentication may take place on data-link layer

CT437

COMPUTER SECURITY AND FORENSIC COMPUTING

BUFFER OVERFLOW CASE STUDY – THE HEARTBLEED BUG

Dr. Michael Schukat



OÉ Gaillimh  
NUI Galway

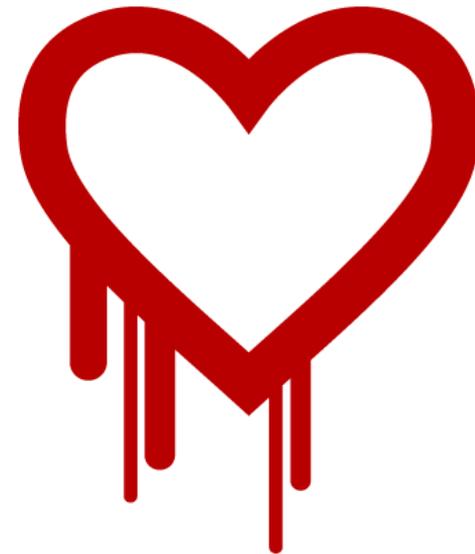
# A Bug with its own Website (heartbleed.com) and Icon

2

## The Heartbleed Bug

The Heartbleed Bug is a serious vulnerability in the popular OpenSSL cryptographic software library. This weakness allows stealing the information protected, under normal conditions, by the SSL/TLS encryption used to secure the Internet. SSL/TLS provides communication security and privacy over the Internet for applications such as web, email, instant messaging (IM) and some virtual private networks (VPNs).

The Heartbleed bug allows anyone on the Internet to read the memory of the systems protected by the vulnerable versions of the OpenSSL software. This compromises the secret keys used to identify the service providers and to encrypt the traffic, the names and passwords of the users and the actual content. This allows attackers to eavesdrop on communications, steal data directly from the services and users and to impersonate services and users.



### What leaks in practice?

We have tested some of our own services from attacker's perspective. We attacked ourselves from outside, without leaving a trace. Without using any privileged information or credentials we were able to steal from ourselves the secret keys used for our X.509 certificates, user names and passwords, instant messages, emails and business critical documents and communication.

### How to stop the leak?

As long as the vulnerable version of OpenSSL is in use it can be abused. [Fixed OpenSSL](#) has been released and now it has to be deployed. Operating system vendors and distribution, appliance vendors, independent software vendors have to adopt the fix and notify their users. Service providers and users have to install the fix as it becomes available for the operating systems, networked appliances and software they use.

X X X

# Overview Heartbleed

3

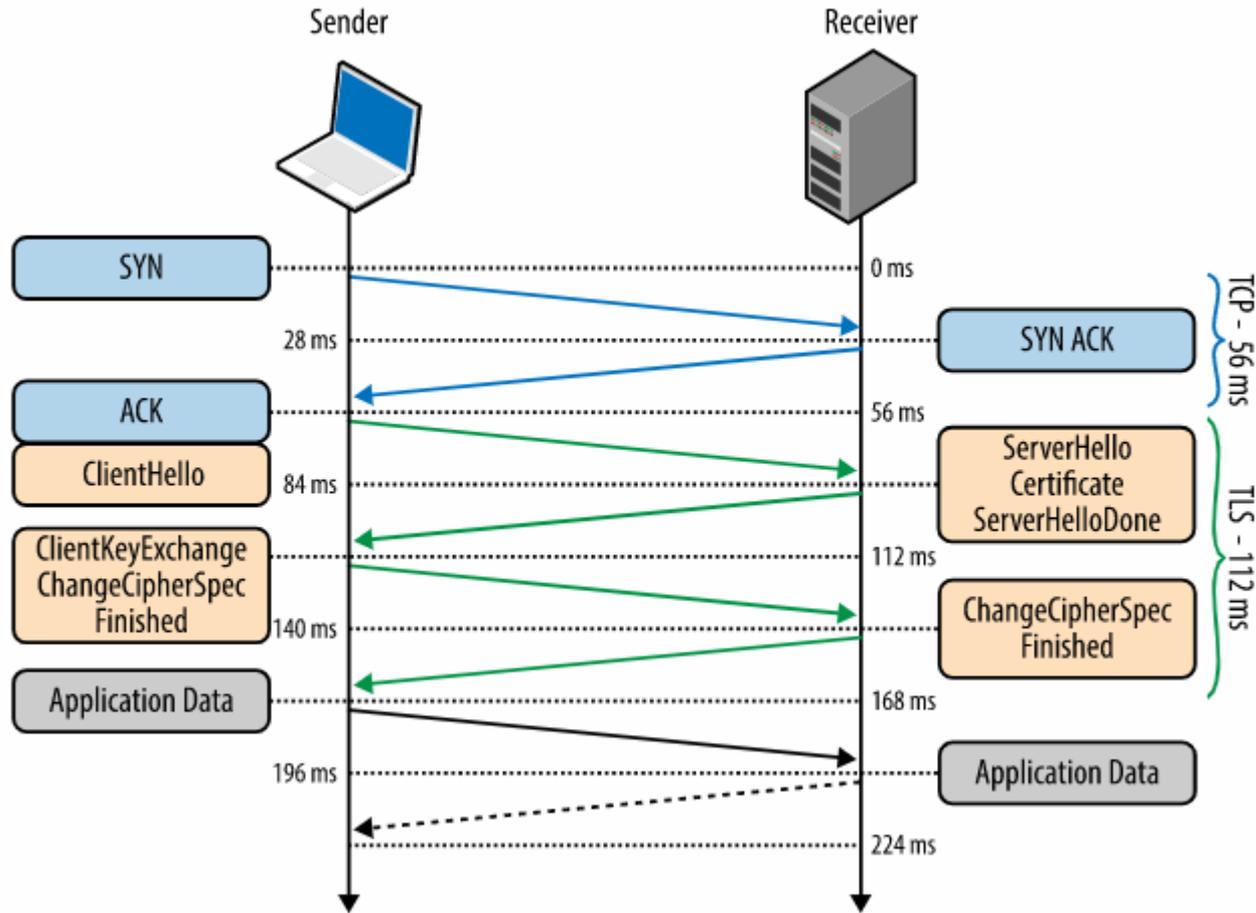
- ❑ Discovered in 2014
- ❑ Exploits a bug in the OpenSSL implementation of the TLS “heartbeat hello” extension
- ❑ Can affect both client and server side



X X

# Recap TLS 1.2 Handshake (Server Authentication only)

4



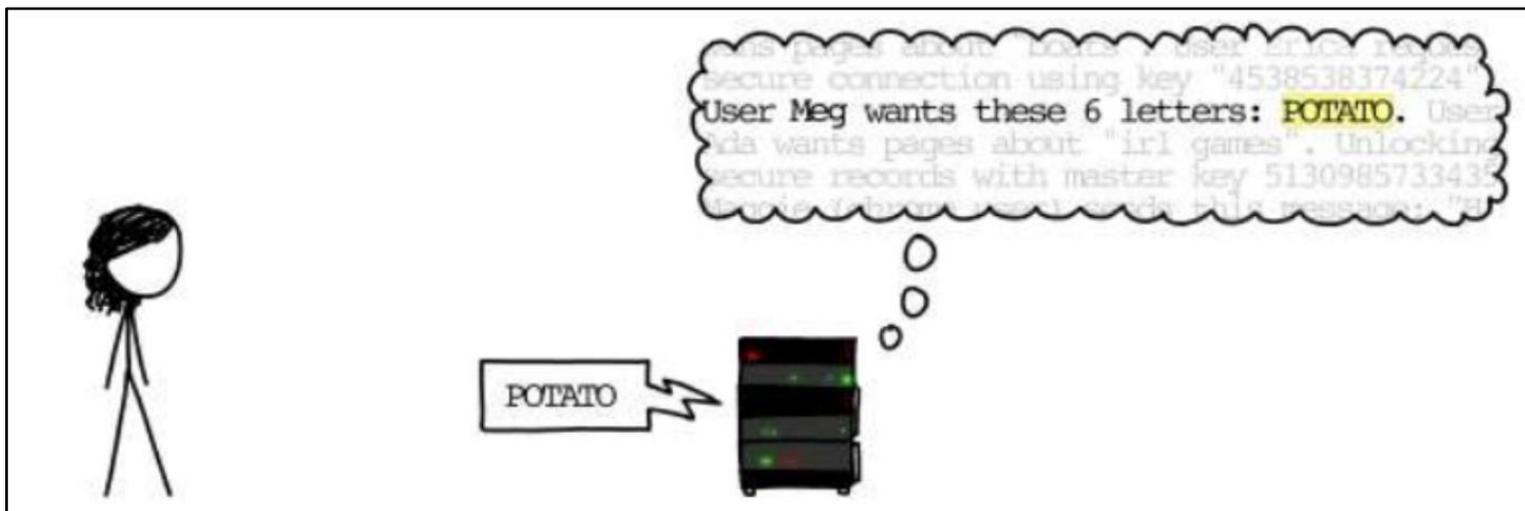
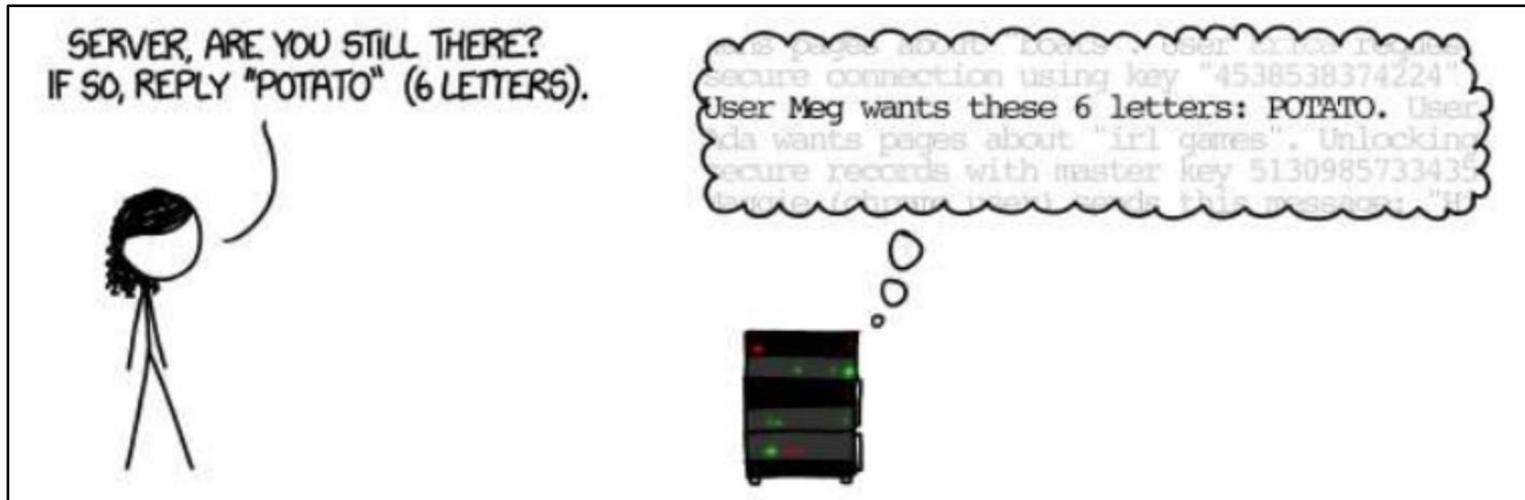
# TLS Heartbeat Extension

5

- ❑ Originally TLS had no provisions to keep a client / server connection alive without continuous data transfer
  - ▣ Idle connections would timeout instead and a (computationally) expensive handshake (224 ms in the previous example) or a reconnect would have to take place
- ❑ The heartbeat extension provides a protocol for “keep-alive” messages that prevent a timeout
  - ▣ One endpoint could send out a *HeartbeatRequest* message, which would be immediately responded with a *HeartbeatResponse* message

# Heartbeat with incoming Message (correctly) buffered

6



# Heartbeat Request / Response Message

7

- The Heartbeat protocol messages consist of their type and an arbitrary payload and padding.

heartbeat\_request or heartbeat\_response

- struct {  
HeartbeatMessageType type;  
uint16 payload\_length;  
opaque payload[HeartbeatMessage.payload\_length];  
opaque padding[padding\_length];  
} HeartbeatMessage;

16+ bytes of random  
content, ignored by receiver

- The sender composes a request message containing a payload with a specified length (i.e. *payload\_length*)
- The receiver returns a response message containing a copy of the sender's payload (with length *payload\_length*)
- “opaque” is a typedef (i.e., unsigned char)

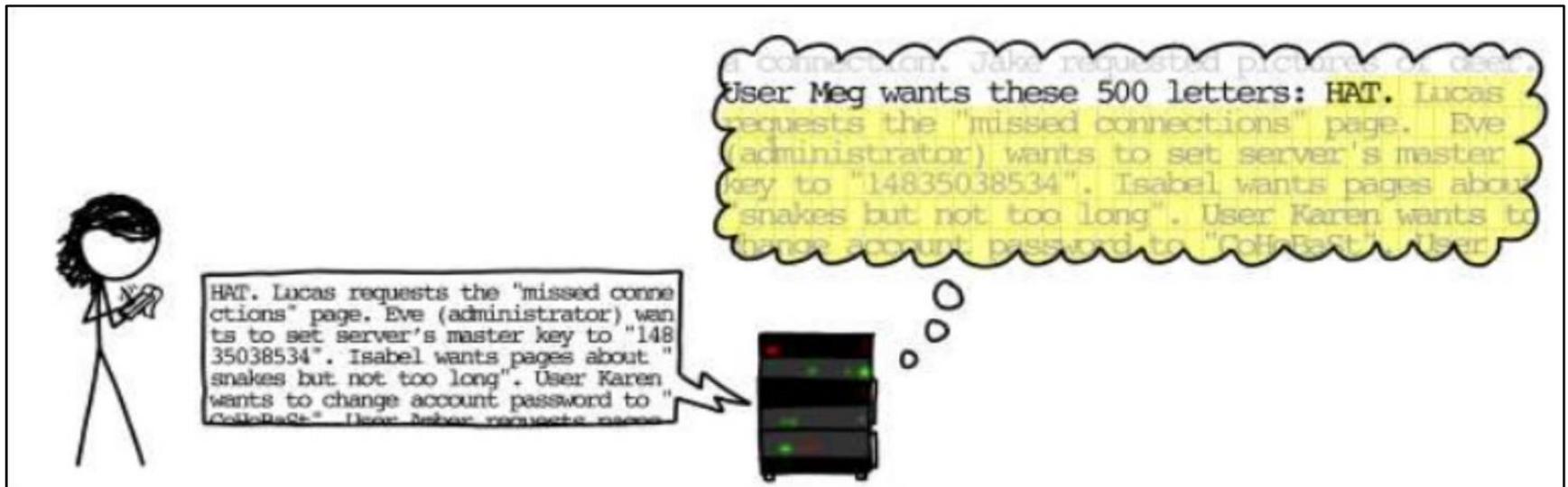
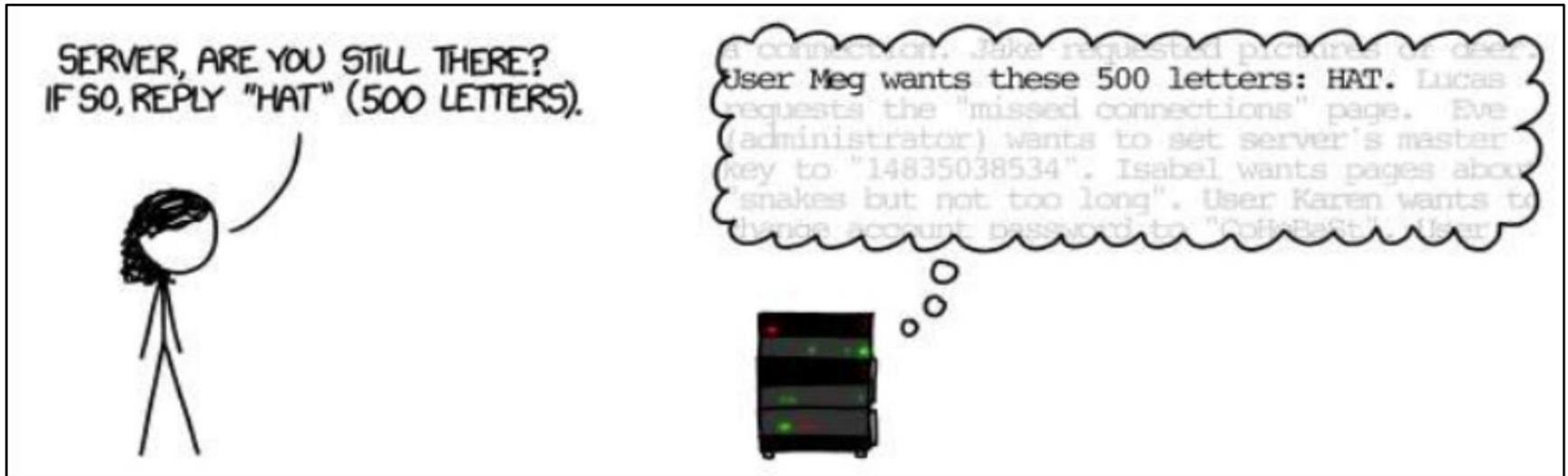
# Heartbleed Exploit

8

- The server receives a Heartbeat request message and copies it into memory, for further processing
  - ▣ However, memory also contains information from other sessions including tokens, keys, session IDs etc.
- If *payload\_length* is actually larger than the *payload[..]*, the server will copy memory content beyond the payload array into the response message's payload array (let's call it *ret\_payload*), which is then sent back to the sender
- *memcpy(ret\_payload, payload, payload\_length);*
- Remember, this is C (and not Java or Python), so array boundaries are not checked!
- This is a typical **buffer over read attack**

# The Heartbleed Attack

9



# Heartbleed Exploit Extract (Python Code)

10

□ <https://gist.github.com/eelsivart/10174134>

Heartbleed (CVE-2014-0160) Test & Exploit Python Script

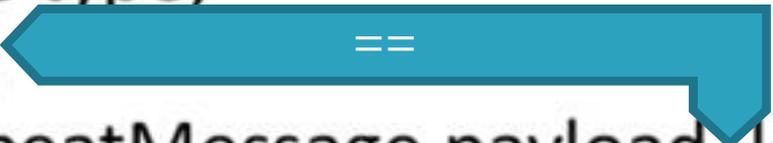
```
heartbleed.py Raw
1  #!/usr/bin/python
2
3  # Modified by Travis Lee
4  # Last Updated: 4/21/14
5  # Version 1.16
6  #
7  # -changed output to display text only instead of hexdump and made it easier to read
8  # -added option to specify number of times to connect to server (to get more data)
9  # -added option to send STARTTLS command for use with SMTP/POP/IMAP/FTP/etc...
10 # -added option to specify an input file of multiple hosts, line delimited, with or without a port specified (host:port)
11 # -added option to have verbose output
12 # -added capability to automatically check if STARTTLS/STLS/AUTH TLS is supported when smtp/pop/imap/ftp ports are entered and automatically
13 # -added option for hex output
14 # -added option to output raw data to a file
15 # -added option to output ascii data to a file
16 # -added option to not display returned data on screen (good if doing many iterations and outputting to a file)
17 # -added tls version auto-detection
18 # -added an extract rsa private key mode (orig code from epixoip. will exit script when found and enables -d (do not display returned data
19 # -requires following modules: gmpy, pyasn1
20
21 # Quick and dirty demonstration of CVE-2014-0160 by Jared Stafford (jspenguin@jspenguin.org)
22 # The author disclaims copyright to this source code.
23
24 import sys
25 import struct
26 import socket
27 import time
28 import select
29 import re
```



# What happened next?

12

- ❑ The Heartbleed bug was fixed (of course)
- ❑ Further checks and balances were added to validate that payload length was correct

```
struct {  
    HeartbeatMessageType type;  
    uint16 payload_length;  ==  
    opaque payload[HeartbeatMessage.payload_length];  
    opaque padding[padding_length];  
} HeartbeatMessage;
```

# Heartbleed Impact

13

- The Heartbleed vulnerability was in all versions of OpenSSL released between March 2012 and April 2014
  - ▣ It was a zero-day (i.e., a vulnerability unknown to its owners, developers or anyone capable of mitigating it) **for almost 2 years**
- According to CVE-2014-0160, the following operating system distributions were potentially affected:
  - ▣ Debian Wheezy (stable)
  - ▣ Ubuntu 12.04.4 LTS
  - ▣ CentOS 6.5
  - ▣ Fedora 18
  - ▣ OpenBSD 5.3
  - ▣ FreeBSD 10.0
  - ▣ NetBSD 5.0.2
  - ▣ OpenSUSE 12.2

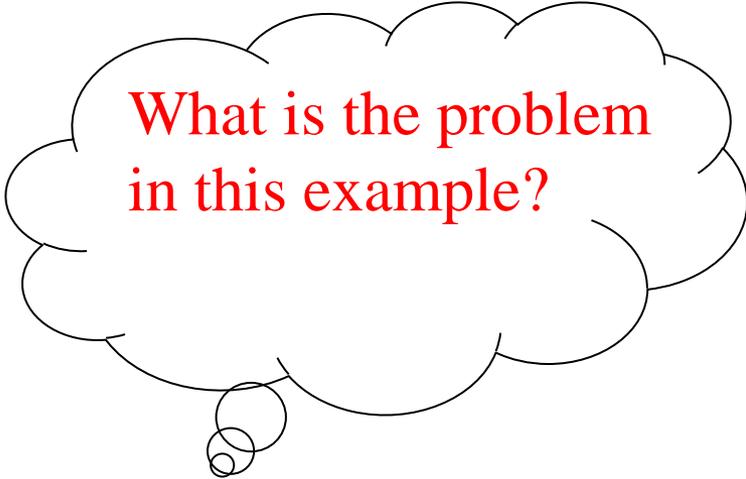
# Lessons learnt

- *OpenSSL core developer Ben Laurie claimed that a security audit of OpenSSL would have caught Heartbleed*
- *Some other quotes from the security community:*
  - ▣ *“Think about it, OpenSSL only has two fulltime people to write, maintain, test, and review 500,000 lines of business-critical code”*
  - ▣ *“The mystery is not that a few overworked volunteers missed this bug; the mystery is why it hasn't happened more often”*
  - ▣ *“There should be a continuous effort to simplify the code, because otherwise just adding capabilities will slowly increase the software complexity. The code should be refactored over time to make it simple and clear, not just constantly add new features. The goal should be code that is “obviously right”, as opposed to code that is so complicated that “I can't see any problems”*

# Related Problem: Buffer Overflow / Stack Overflow

```
#include <string.h>
void foo (char *bar)
{
    char c[12];
    strcpy(c, bar);
}

int main (int argc, char **argv)
{
    foo(argv[1]);
    return(1);
}
```



What is the problem  
in this example?

# Example for a Stack Overflow

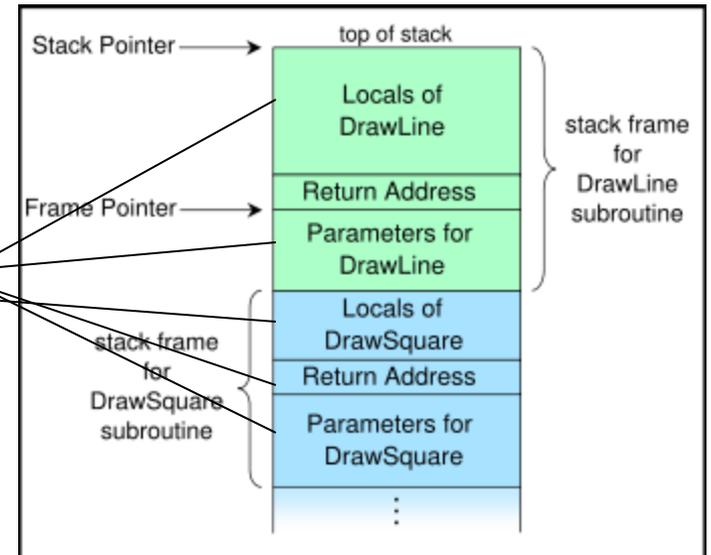
```
#include <string.h>
void foo (char *bar)
{
    char c[12];
    strcpy(c, bar);
}

int main (int argc, char **argv)
{
    foo(argv[1]);
    return(1);
}
```

- Lets assume the compiled program is called test
- Test is invoked from command line (next slide):
  - “> test hello” will work fine
  - “> test AAAAAAAAAAAAAA AAAAAAAAAA” (> 11 characters) may crash the program

# Background Info: The Call Stack

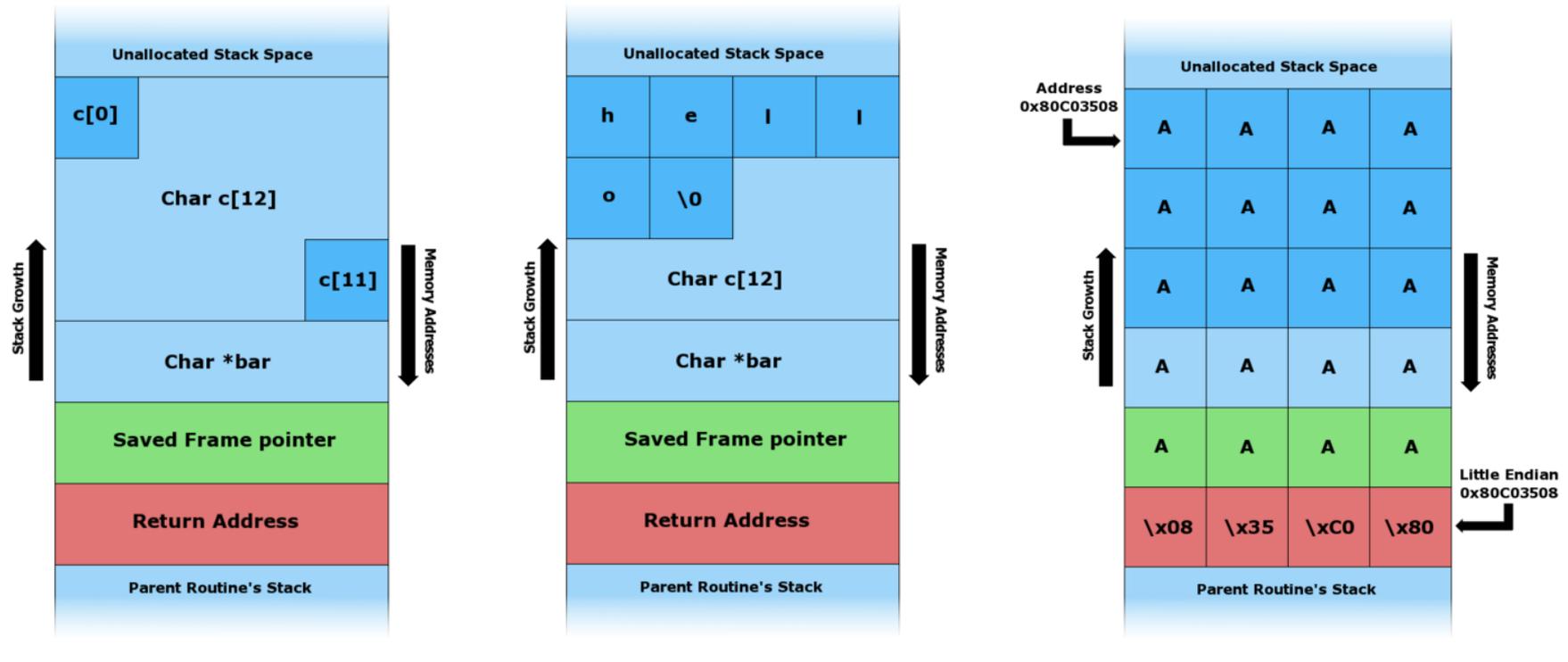
```
main()
{
    DrawSquare(1,1,4,4);
    ...
}
...
void DrawSquare(int lux, int luy, int rbx, int rby)
{
    int t1, t2;
    DrawLine(lux, luy, rbx, luy);
    DrawLine(lux, luy, lux, rby);
    ...
}
...
void DrawLine(int p1x, int p1y, int p2x, int p2y)
{
    int temp1;
    ...
}
```



# Background Info: The Call Stack

- Each stack frame contains a stack pointer to the top of the frame immediately below
  - ▣ The stack pointer is a mutable register
- The stack frame is the collection of all data on the stack associated with one subprogram call. The stack frame generally includes:
  - ▣ The return address
  - ▣ Argument variables passed on the stack
  - ▣ Local variables
- A frame pointer of a given invocation of a function is a copy of the stack pointer as it was before the function was invoked
- If a stack frame is corrupted, i.e. overwritten, arguments, variables and / or return address do change
- If the return address is manipulated, the program can crash, or malware can be executed (with the return address being the start address of the malware in memory)

# Example for a Stack Overflow



# Buffer Overflow Countermeasures

- ❑ Use a programming language that supports automatic bounds checking of buffers
  - ❑ Java or Python, but NOT C
- ❑ Use a language specific library module that implements info validation in the form of safe buffer handling
- ❑ Compilers can produce a warning when an unsafe function call is made, or can add code for buffer overflow detection
- ❑ An Operating System can enforce more stringent memory access control so that buffer overflows cannot infringe into the protected areas of the main memory

# Buffer Overflow Mitigation using Electric Fence / Boundary Checks

21

- Here each data object (i.e., array) is guarded by a boundary signature that is checked for its integrity every time that object is accessed
- If the signature has changed as shown below, the data object is deemed to be corrupted, and an alarm will be raised

Before	0xDA	0xEF	Array[0]	Array[1]	...	Array[n]	0xFF	0xED
Attack	0xAA	0xAA	0xAA	0xAA	...	0xAA	0xAA	0xAA

# Example Code

22

```
...
char boundary0 = 0xDA;
char boundary1 = 0xEF
char array[n];
char boundary2 = 0xFF;
char boundary3 = 0xED;
...
// Access array[] only if boundary is intact.
if ((boundary0 == 0xDA) && (boundary1 == 0xEF) && (boundary2 == 0xFF) && (boundary3 == 0xED))
{
    // Access array
    ...
}
else
{
    // Error handling
    ...
}
```

CT437

COMPUTER SECURITY AND FORENSIC COMPUTING

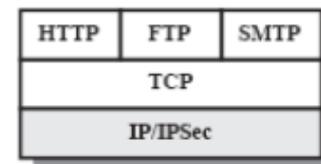
TRANSPORT LAYER SECURITY

Dr. Michael Schukat

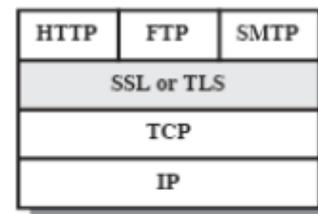


# Background

- The exponential growth of the internet in the 1990s resulted in a need for better security, thereby considering support of
  - ▣ ad-hoc and short-lived client/server connections
  - ▣ casual and untrained users
    - No awareness of risks and key concepts (confidentiality, integrity, authentication)
  - ▣ web browsers as the main vehicle for client / server communication
- The first attempt was Secure Socket Layer (SSL)
  - ▣ Introduced by Netscape in the 1990s
  - ▣ Embedded in web browsers / servers
  - ▣ Later became Internet standard known as TLS (Transport Layer Security)



(a) Network Level



(b) Transport Level

# TLS (Transport Layer Security)

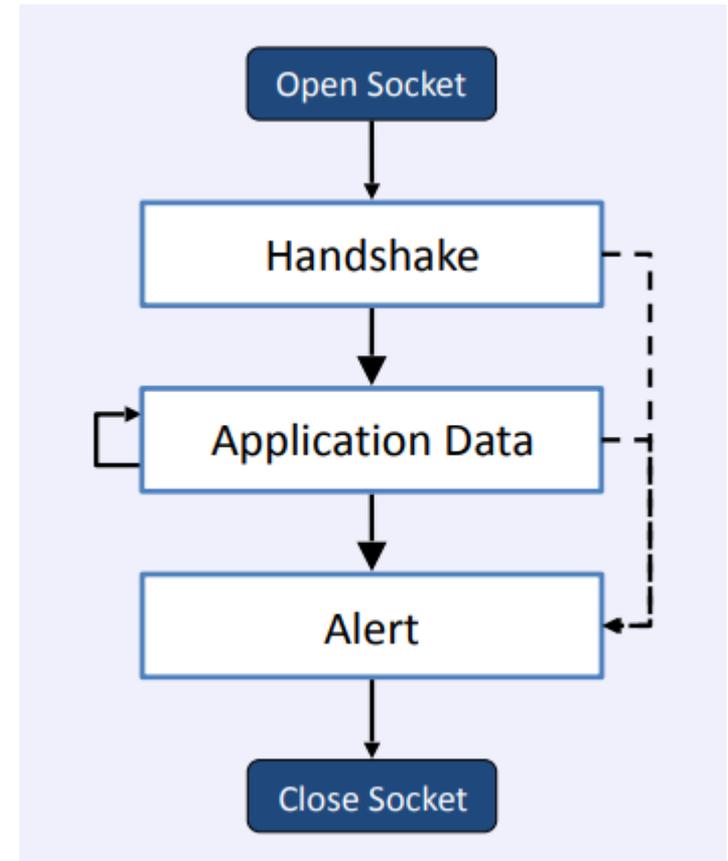
3

- This application layer protocol is widely used for applications such as email, instant messaging and VoIP
  - ▣ Mainly known for securing HTTP (i.e. HTTPS)
- TLS provides
  - ▣ privacy (confidentiality) of exchanged data
  - ▣ integrity of exchanged data
  - ▣ authentication of server (and optionally client) through the use of digital certificates
- Composed of two layers:
  - ▣ TLS handshake protocol (main focus)
  - ▣ TLS record protocol
- It operates on top of TCP, which in turn is gradually replaced by the QUIC (also called TCP/2) protocol

# Sequence of a TLS Session

4

- Handshake Protocol
  - ▣ Agree a cipher suite
  - ▣ Agree a master secret
  - ▣ Authentication using certificate(s)
- Record Protocol
  - ▣ Secure data communication
    - Symmetric key encryption
    - Data authentication
    - Often in combination with HTTP
  - ▣ Alerts
    - Graceful closure, or
    - Problem detected



# Website Protocol Support (Wikipedia)

5

- ❑ SSL 2.0 / 3.0 contain a number of security flaws
- ❑ Support for TLS versions 1.0 and 1.1 was widely deprecated by web sites around 2020
- ❑ TLS 1.3 was released as RFC 8446 in August 2018. It is a streamlined version of the earlier TLS 1.2 specification with some notable changes:
  - ❑ Streamlined handshake
  - ❑ Focus on elliptic curve cryptography using a reduced list of curves, RSA is not supported any more
  - ❑ Removing support for the MD5 and SHA-224 cryptographic hash functions
  - ❑ No more backwards compatibility beyond TLS 1.2
- ❑ As we'll see later, TLS 1.3 presents itself as 1.2 (well almost), this is apparently for compatibility reasons
- ❑ **Today, only TLS 1.2 and TLS 1.3 are in use, that's the focus of this lecture!**

Protocol version	Website support <sup>[72]</sup>	Security <sup>[72][73]</sup>
SSL 2.0	0.4%	Insecure
SSL 3.0	3.0%	Insecure <sup>[74]</sup>
TLS 1.0	43.8%	Deprecated <sup>[9][10][11]</sup>
TLS 1.1	47.8%	Deprecated <sup>[9][10][11]</sup>
TLS 1.2	99.6%	Depends on cipher <sup>[n 1]</sup> and client mitigations <sup>[n 2]</sup>
TLS 1.3	49.7%	Secure

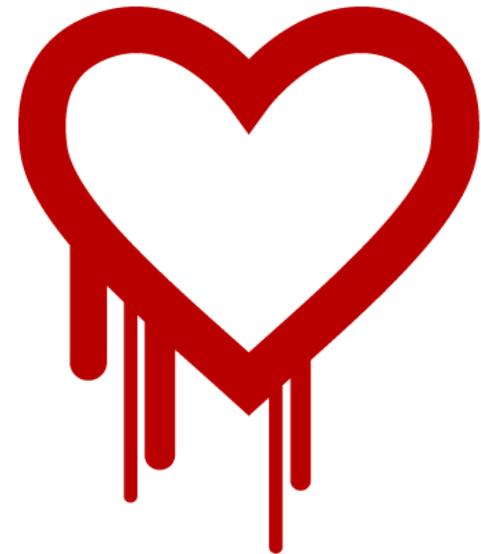
# Issues with Legacy TLS Versions: The Heartbleed Vulnerability in TLS 1.0 (2014)

6

## The Heartbleed Bug

The Heartbleed Bug is a serious vulnerability in the popular OpenSSL cryptographic software library. This weakness allows stealing the information protected, under normal conditions, by the SSL/TLS encryption used to secure the Internet. SSL/TLS provides communication security and privacy over the Internet for applications such as web, email, instant messaging (IM) and some virtual private networks (VPNs).

The Heartbleed bug allows anyone on the Internet to read the memory of the systems protected by the vulnerable versions of the OpenSSL software. This compromises the secret keys used to identify the service providers and to encrypt the traffic, the names and passwords of the users and the actual content. This allows attackers to eavesdrop on communications, steal data directly from the services and users and to impersonate services and users.



### What leaks in practice?

We have tested some of our own services from attacker's perspective. We attacked ourselves from outside, without leaving a trace. Without using any privileged information or credentials we were able to steal from ourselves the secret keys used for our X.509 certificates, user names and passwords, instant messages, emails and business critical documents and communication.

### How to stop the leak?

As long as the vulnerable version of OpenSSL is in use it can be abused. [Fixed OpenSSL](#) has been released and now it has to be deployed. Operating system vendors and distribution, appliance vendors, independent software vendors have to adopt the fix and notify their users. Service providers and users have to install the fix as it becomes available for the operating systems, networked appliances and software they use.

# Issues with Legacy TLS Versions: Apple 'goto fail;' Vulnerability in TLS 1.0 and TLS 1.1 (2014)

7

- ❑ Affected iOS and Mac OS X operation systems
- ❑ This vulnerability enabled MitM attacks on TLS connections

```
if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;

err = sslRawVerify(ctx,
                  ctx->peerPubKey,
                  dataToSign,
                  dataToSignLen,
                  signature,
                  signatureLen);
/* plaintext */
/* plaintext length */

if(err) {
    sslErrorLog("SSLDecodeSignedServerKeyExchange: sslRawVerify "
               "returned %d\n", (int)err);
    goto fail;
}

fail:
SSLFreeBuffer(&signedHashes);
SSLFreeBuffer(&hashCtx);
return err;
```

# TLS Record Protocol Characteristics

8

- The connection is private because a symmetric-key algorithm (i.e., AES) is used to encrypt the data transmitted
- The identity of the communicating parties is authenticated via digital certificates that are exchanged and validated during the initial handshake
  - ▣ This (server-side) authentication is required for the server and optional for the client (i.e. client-side authentication)
    - We focus on server-side authentications for now
- The connection is reliable, because each message transmitted includes a message integrity check using a message authentication code to prevent undetected loss or alteration of the data during transmission

# TLS Handshake Protocol Overview

9

- Secure (TLS) connection is initiated by client
  - Typically, via dedicated port, e.g. HTTP port 80 versus HTTPS port 443
- It uses public key cryptography to establish cipher settings and session-specific shared private keys with which further communication is encrypted using a symmetric cipher
  - Client and server agree on a **cipher suite** (a cipher and a hash function)
- The server also presents its digital certificate to the client for authentication
- To initiate the generation of session keys used for a secure connection, the client either:
  1. Encrypts a random number (PreMasterSecret) with the server's (RSA or EC) public key and sends the result to the server (only up to TLS 1.2)
    - Forward secrecy is not provided!
  2. Uses (Elliptic Curve) Diffie–Hellman key exchange (in TLS 1.2 and TLS 1.3)
    - This key may have the property of forward secrecy, but MitM attacks need to be mitigated

# Recall Forward Secrecy

10

- Consider an attacker who
  - ▣ intercepts and records all client / server messages, including the handshake
  - ▣ recovers the server's private key sometime in the future, using the public key in the server's digital certificate as a starting point
- In option 1 the *PreMasterSecret* can now be retrospectively recovered, session keys can be calculated, and all subsequent messages can be decrypted by the attacker
- However, the DH key negotiation in option 2 is based on other secret token not linked to the server's private key
  - ▣ Nonetheless the key exchange has to be protected to prevent a MitM attack as seen before

# Ephemeral Diffie-Hellman vs static Diffie-Hellman

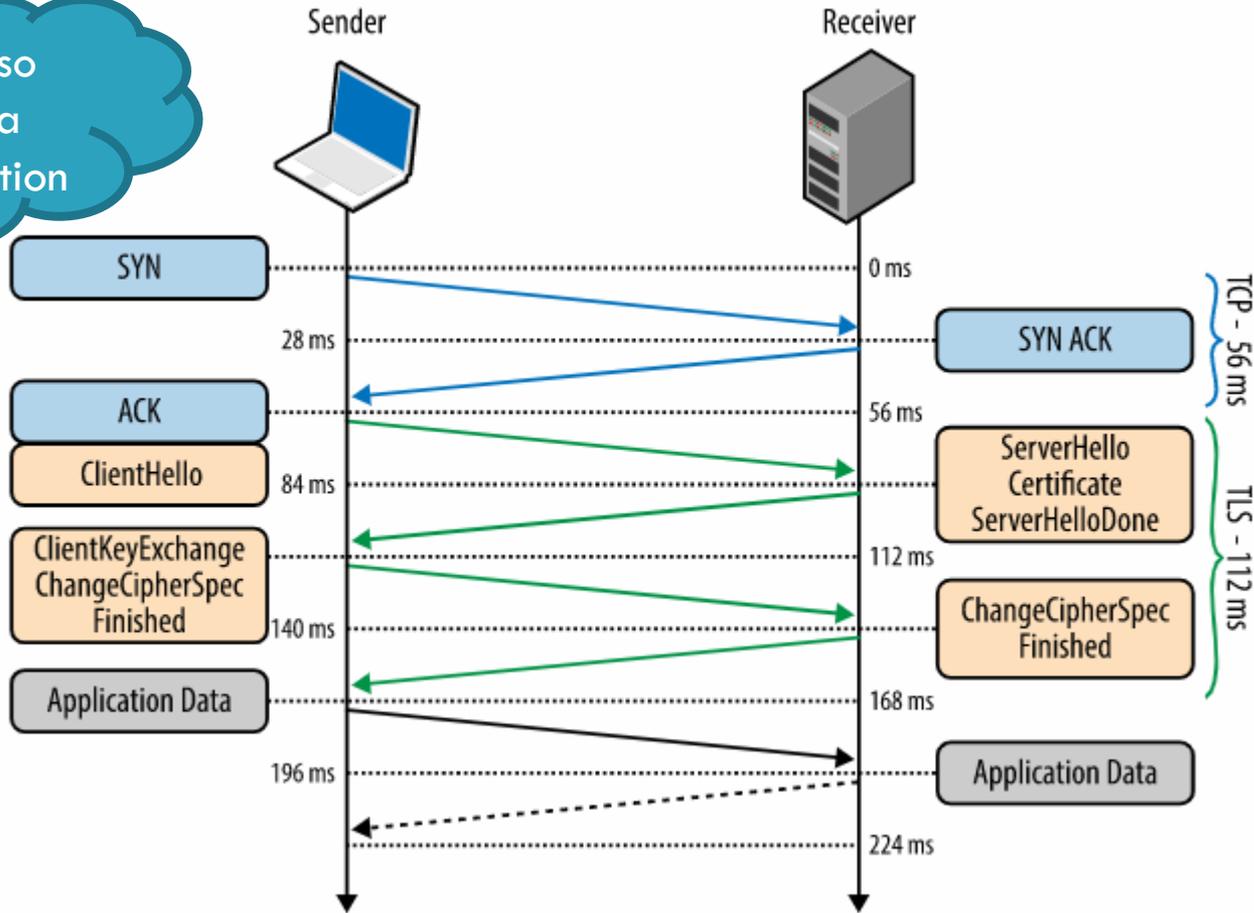
11

- ❑ Static Diffie-Hellman key exchange (in TLS 1.2 only)
  - ❑ Always use the same Diffie-Hellman private keys (this saves CPU cycles)
  - ❑ Each time the same parties do a DH key exchange, they end up with the same shared secret → only partial forward secrecy
- ❑ Ephemeral Diffie-Hellman key exchange (compulsory in TLS 1.3)
  - ❑ A temporary DH key is generated for every connection and thus the same key is never used twice
  - ❑ This enables forward secrecy, which means that if the long-term private key of the server gets leaked, past communication is still secure
- ❑ This distinction also holds for the Elliptic Curve DH variants
  - ❑ ECDHE (ephemeral, provides Forward Secrecy) and
  - ❑ ECDH (static)

# TLS Handshake Overview

12

TLS 1.3 also supports a faster variation



# In-Class Activity: Analysis of TLS Handshake

13

- Option 1:
  - ▣ Open Wireshark and start packet recording
  - ▣ In your browser open a HTTPS secured website you never visited before (e.g. fussball.de)
  - ▣ Stop packet recording and filter all TLS-related packets (Filter option 'tls')
- Option 2:
  - ▣ Load pcap file "revenue tls" (Blackboard file name "Example Wireshark TLS Handshake")
- Wireshark does a great job analysing the content of the packets

# TLS Handshake

14

- TCP connection establishment
  - ▣ SYN – SYN/ACK – ACK
- The *ClientHello* message
  - ▣ The client initiates the handshake by sending a (plaintext) “hello” message to the server
  - ▣ The message includes
    - the highest TLS version the client supports (1.2 or 1.3)
    - the cipher suites supported (i.e. what algorithms are available to client, see next slide),
    - a session identifier
      - Note that the session id is kept empty if the clients starts an entirely new session
    - a string of random bytes known as the “client random”

# Cipher Suite Naming Scheme

15

## □ Examples:

- ▣ TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256
- ▣ TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_GCM\_SHA384

## □ Here:

- ▣ TLS defines the protocol that this cipher suite is for
- ▣ ECDHE indicates the key exchange algorithm being used (Elliptic Curve Diffie-Hellman Ephemeral)
- ▣ RSA or ECDSA (Elliptic Curve Digital Signature Algorithm) authentication mechanism during the handshake
  - Remember the *ServerHello* message contains the server's public DH parameter signed with its private (RSA) key or signed via ECDSA
- ▣ AES cipher for symmetric data encryption
- ▣ 128-bit or 256-bit AES key size
- ▣ GCM type of encryption (Galois/Counter Mode, covered before)
- ▣ SHA256 / SHA384 hash function (HMAC) indicates the message authentication algorithm which is used to authenticate a message
  - 256-bit or 384-bit digest size

# Cipher Suite (Wireshark Screenshot)

16

```
▼ Cipher Suites (16 suites)
  Cipher Suite: Reserved (GREASE) (0x7a7a)
  Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
  Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
  Cipher Suite: TLS_CHACHA20_POLY1305_SHA256 (0x1303)
  Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
  Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
  Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
  Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
  Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xc0a9)
  Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xc0a8)
  Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
  Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
  Cipher Suite: TLS_RSA_WITH_AES_128_GCM_SHA256 (0x009c)
  Cipher Suite: TLS_RSA_WITH_AES_256_GCM_SHA384 (0x009d)
  Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
  Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
```

- Two bytes specify a cipher suite
- Suits have different levels of robustness
- See also for details

<https://ciphersuite.info/cs/>

# Cipher Suite (Wireshark Screenshot)

17

```
▼ Cipher Suites (16 suites)
  Cipher Suite: Reserved (GREASE) (0x7a7a)
  Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
  Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
  Cipher Suite: TLS_CHACHA20_POLY1305_SHA256 (0x1303)
  Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
  Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
  Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
  Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
  Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xc0a9)
  Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xc0a8)
  Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
  Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
  Cipher Suite: TLS_RSA_WITH_AES_128_GCM_SHA256 (0x009c)
  Cipher Suite: TLS_RSA_WITH_AES_256_GCM_SHA384 (0x009d)
  Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
  Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
```

- Two bytes specify a cipher suite
  - ▣ Suits have different levels of robustness
  - ▣ See also for details <https://ciphersuite.info/cs/>
- Cipher suits 0x1301, 0x1302 and 0x1303 do not describe the
  - ▣ server authentication mechanism (e.g., RSA)
  - ▣ key exchange algorithm (e.g., ECDHE)
- This is a simplification introduced with TLS 1.3 (more later!)

# TLS Handshake

18

## □ The *ServerHello* message

- ▣ In reply to the *ClientHello* message, the server sends a (plaintext) message containing
  - the server's digital certificate
  - a certificate chain that includes all intermediate certificates up to the root CA along the certification path
  - the server's chosen cipher suite,
  - its chosen session id (session resumption → later), and
  - the "server random," another random string of bytes that's generated by the server

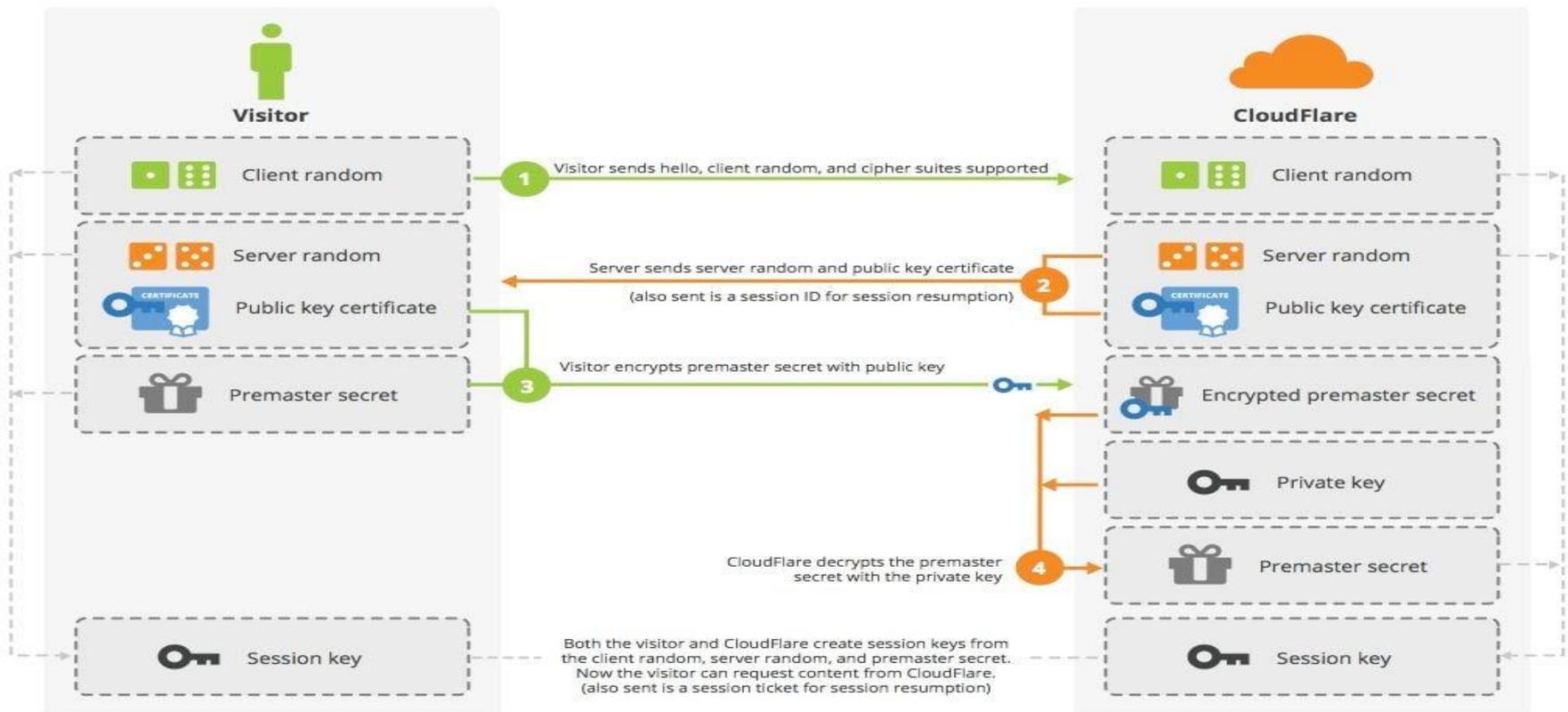
# TLS Handshake

19

- Authentication
  - ▣ The client verifies the server's digital certificate with the certificate authority that issued it using the intermediate certificates
  - ▣ This confirms that the public key is linked to the certificate owner, but does not confirm the authenticity of the server yet (as any threat actor could use the server's certificate in a spoofing attack)
- Key negotiation (next slides)
  - ▣ Option 1: RSA handshake (not supported anymore with TLS 1.3)
  - ▣ Option 2: DH handshake (ECDH to be exact)
- Change Cipher Spec (not shown in the diagrams in the following slides)
  - ▣ In due course both parties will send a ChangeCipherSpec message which is used to indicate that their subsequent messages will be sent encrypted using the negotiated key and algorithm
- Finished (not shown in the diagrams on the following slides)
  - ▣ This is an encrypted message (more later)

# Option 1 Overview: RSA Handshake

20



# Option 1: RSA Handshake

21

- Premaster secret generation
  - ▣ The client generates a random string of bytes, the "premaster secret"
  - ▣ The premaster secret is encrypted with the server's public key
- Premaster secret distribution 3 4
  - ▣ The client sends the encrypted secret to the server
  - ▣ The server decrypts the premaster secret
- Master secret creation
  - ▣ Both client and server generate a master secret (which is not the encryption key used), using
    - the client random,
    - the server random,
    - and the premaster secret

# Option 1: RSA Handshake

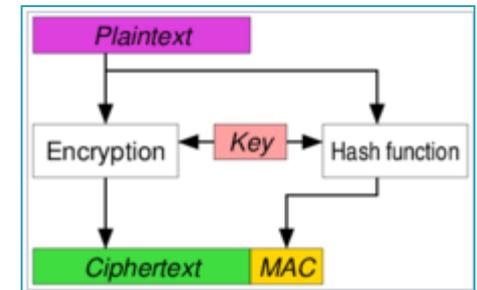
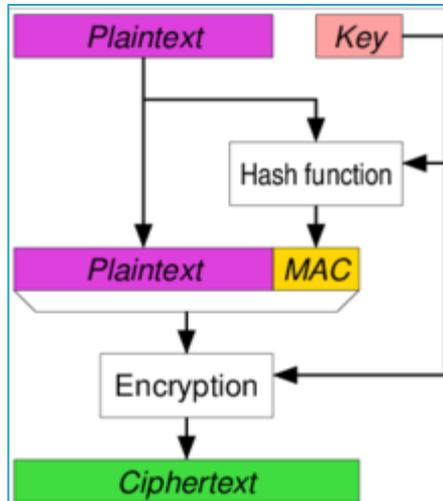
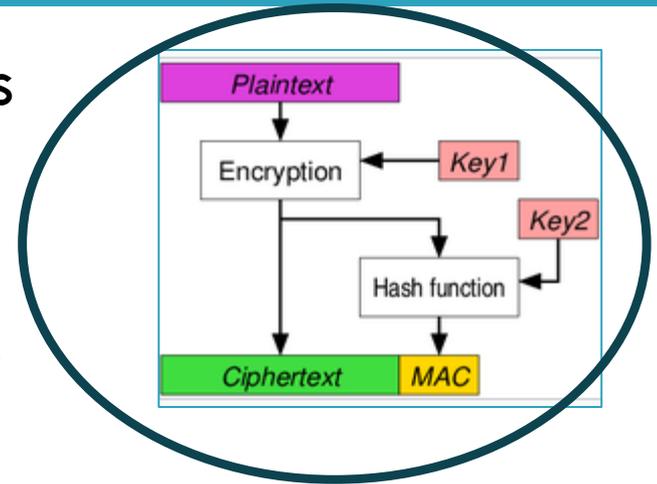
22

- Session keys generation
  - ▣ Using the master secret both client and server generate 4 session keys (see next slide):
    - Client-write symmetric encryption key
    - Server-write symmetric encryption key
    - Client-write MAC key (for client message authentication)
    - Server-write MAC key (for server message authentication)
- Client is ready
  - ▣ The client sends a *finished* message that is encrypted with the session key
- Server is ready
  - ▣ The server sends a *finished* message encrypted with the session key
    - This validates the authenticity of the server, i.e. the client has proof that the server is in possession of the private key linked to the server certificate
- Secure symmetric encryption can be provided
  - ▣ The handshake is completed, and communication continues using the session keys

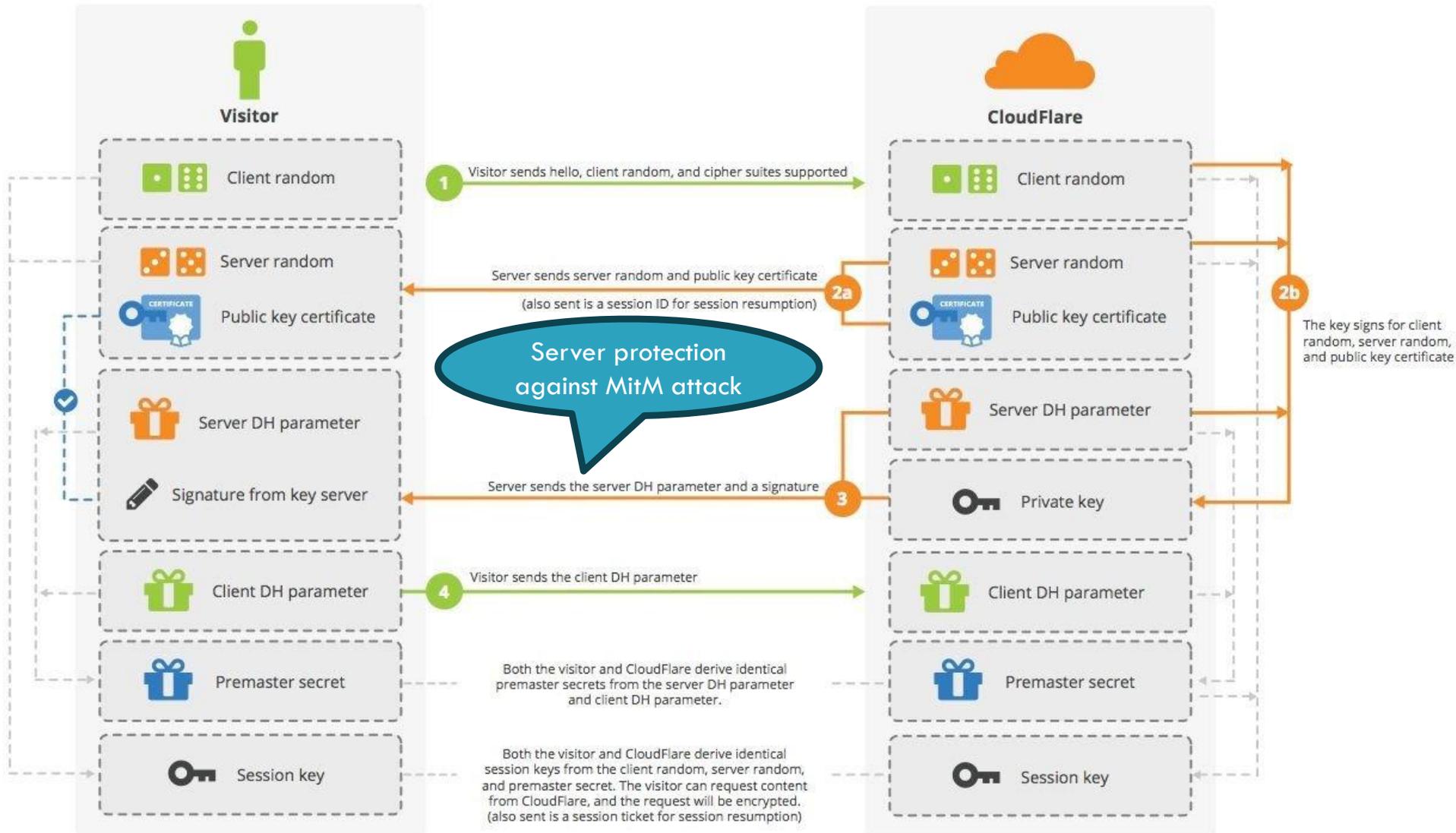
# Recall: Authenticated Encryption with Additional Data

23

- Links back to the use of hash functions (→ previous lecture):
  - ▣ Encrypt-then-MAC (EtM) → top right
  - ▣ Encrypt-and-MAC (E&M) → bottom right
  - ▣ MAC-then-Encrypt (MtE) → bottom left



# Option 2 Overview: DH Handshake



# Option 2: DH Handshake

25

- Server Key Exchange 3
  - This message contains either ECDH parameters (elliptic curve + primitive root + public ECDH parameter) or DH parameters (modulus, primitive root, public DH value) to be used by the client
  - The values are signed by using the private (RSA or EC) key of the server so that the client can verify (using corresponding public key in the certificate) that the parameter indeed came from the server it is talking to and not an attacker that impersonates the server
    - Note that in
      - TLS 1.2: DH, ephemeral DH (DHE), ECDH, or ECDHE can be used
      - TLS 1.3: only ECDHE is allowed
- Client Key Exchange 4
  - Contains the client's public parameters for the DH algorithm
  - Client parameters are **not signed** (as the client does not have a certificate)

# Option 2 Overview: DH Handshake

26

- Client and server calculate the **premaster secret**
  - ▣ Instead of the client generating the premaster secret and sending it to the server, as seen before, the client and server use the DH parameters they exchanged to calculate a matching premaster secret separately
- Master secret creation
  - ▣ The client and server calculate the master secret using the premaster secret, client random, and server random
- Session keys generation
  - ▣ Same as before
- Client is ready
  - ▣ Same as before
- Server is ready
  - ▣ Secure symmetric encryption achieved

# ClientHello (Wireshark Screenshot)

27

```
> Transmission Control Protocol, Src Port: 63377, Dst Port: 443, Seq: 1, Ack: 1, Len: 517
▼ Transport Layer Security
  ▼ TLv1.3 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: TLS 1.0 (0x0301)
    Length: 512
  ▼ Handshake Protocol: Client Hello
    Handshake Type: Client Hello (1)
    Length: 508
    Version: TLS 1.2 (0x0303)
    Random: 5628afe2a5afa352d8a3336c39307da39b13ec3d009e4c9f9ef622ae51df6e49
    Session ID Length: 32
    Session ID: 07c2d49a4554a1463c42de69738c7b645dbc5691c301e26bb3663df24c965f37
    Cipher Suites Length: 32
    > Cipher Suites (16 suites)
    Compression Methods Length: 1
    > Compression Methods (1 method)
    Extensions Length: 403
    > Extension: Reserved (GREASE) (len=0)
    > Extension: server_name (len=30)
    > Extension: extended_master_secret (len=0)
    > Extension: renegotiation_info (len=1)
    > Extension: supported_groups (len=10)
    > Extension: ec_point_formats (len=2)
    > Extension: session_ticket (len=0)
    > Extension: application_layer_protocol_negotiation (len=14)
    > Extension: status_request (len=5)
    > Extension: signature_algorithms (len=18)
    > Extension: signed_certificate_timestamp (len=0)
    > Extension: key_share (len=43)
    > Extension: psk_key_exchange_modes (len=2)
    > Extension: supported_versions (len=7)
    > Extension: compress_certificate (len=3)
    > Extension: application_settings (len=5)
    > Extension: Reserved (GREASE) (len=1)
    > Extension: padding (len=190)
```

- ❑ Highest TLS version supported
- ❑ 32-byte random structure (contains a 4-byte timestamp and a 28-byte random → next slide)
- ❑ Random 32-byte session id
- ❑ List of supported cryptographic algorithms
- ❑ List of supported data compression methods, obsolete with TLS 1.3
- ❑ List of extensions
- ❑ **Note that all is plaintext!**
- ❑ Version of the record protocol (still 1.0)

# Client Hello: 32-Byte Random Structure

28

## □ From RFC 5246 Section 7.4.1.2:

The ClientHello message includes a random structure, which is used later in the protocol.

```
struct {  
    uint32 gmt_unix_time;  
    opaque random_bytes[28];  
} Random;
```

gmt\_unix\_time

The current time and date in standard UNIX 32-bit format (seconds since the midnight starting Jan 1, 1970, UTC, ignoring leap seconds) according to the sender's internal clock. Clocks are not required to be set correctly by the basic TLS protocol; higher-level or application protocols may define additional requirements. Note that, for historical reasons, the data element is named using GMT, the predecessor of the current worldwide time base, UTC.

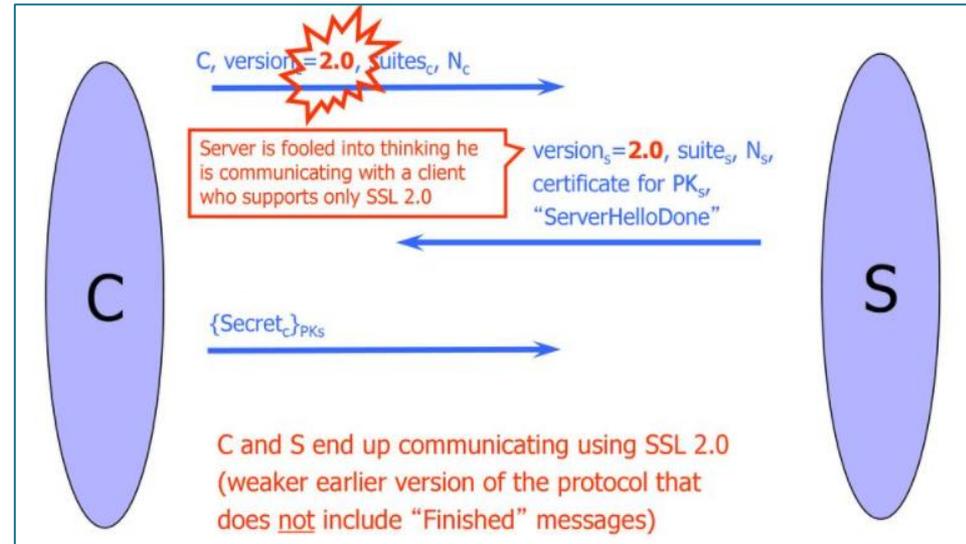
random\_bytes

28 bytes generated by a secure random number generator.

# The Version Rollback Attack

29

- This MitM attack targets SSL 3.0
- Here the attacker intercepts the plaintext *ClientHello* message, that includes the highest TLS version the client supports (i.e. SSL 3.0)



- The attacker changes the message content to “SSL 2.0”, thereby tricking both server and client to accept a weaker (i.e. flawed) protocol
  - ▣ The server assumes the client only understands SSL 2.0
  - ▣ The client assumes the server only understands SSL 2.0

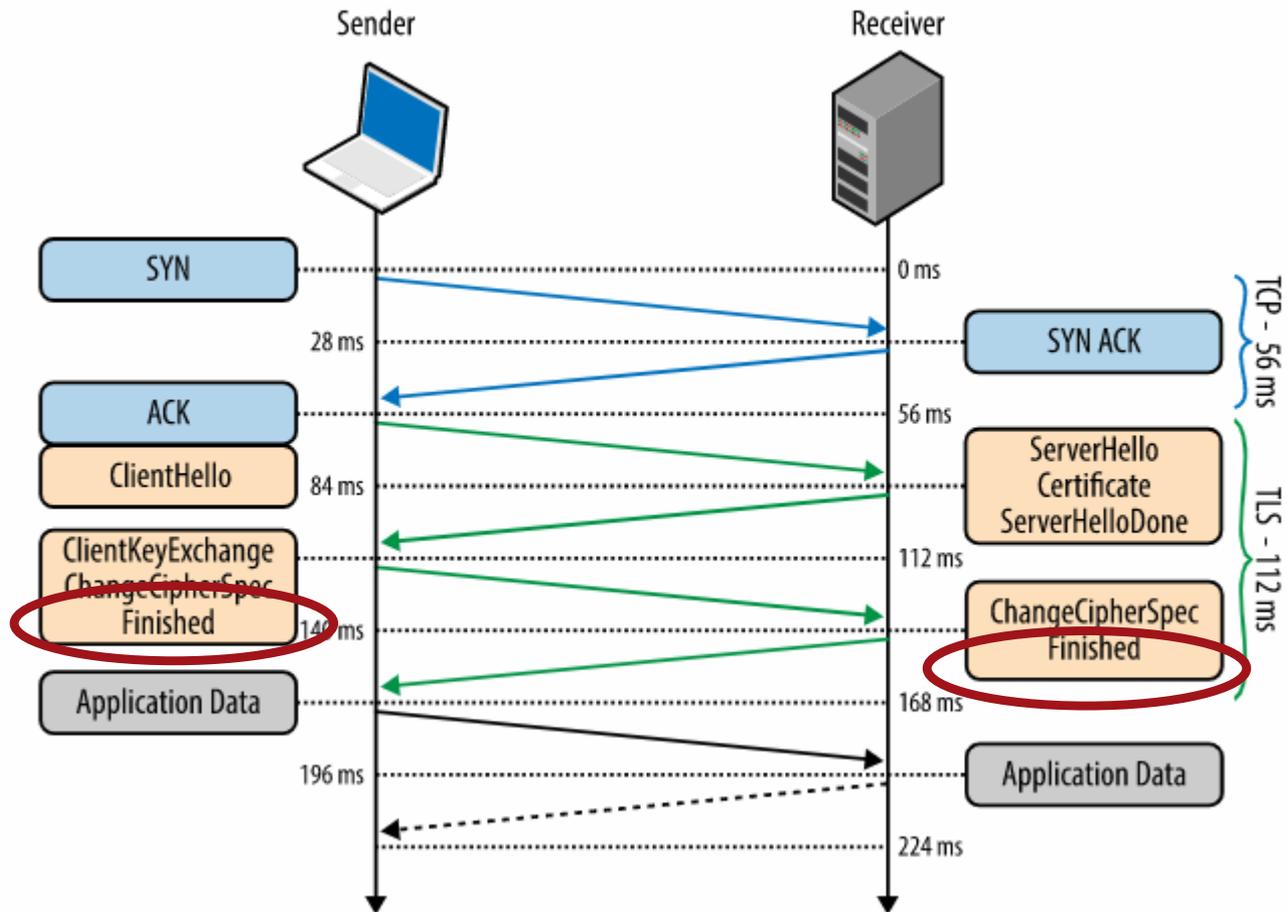
# TLS Protection against MitM Attacks

30

- MitM attacks cannot be mitigated, as *Client Hello* and *Server Hello* messages, as well as the client key exchange messages for DH key negotiation are sent as plaintext
- Instead, the *Finished* messages of both client and server contain the result of the HMAC of the negotiated cyphersuite, truncated to 12 bytes (therefore called a pseudo-random function (PRF)), of:
  - ▣ The master secret
  - ▣ A hash of all the previous handshake messages (from *ClientHello* up to but excluding the *Finished* message)
  - ▣ The finished-label string (“client finished” for client message and “server finished” for server message)
- Therefore, both sides can retrospectively validate the integrity of the handshake protocol
  - ▣ This includes all MitM attacks during the key exchange protocol (remember only the server value was signed)

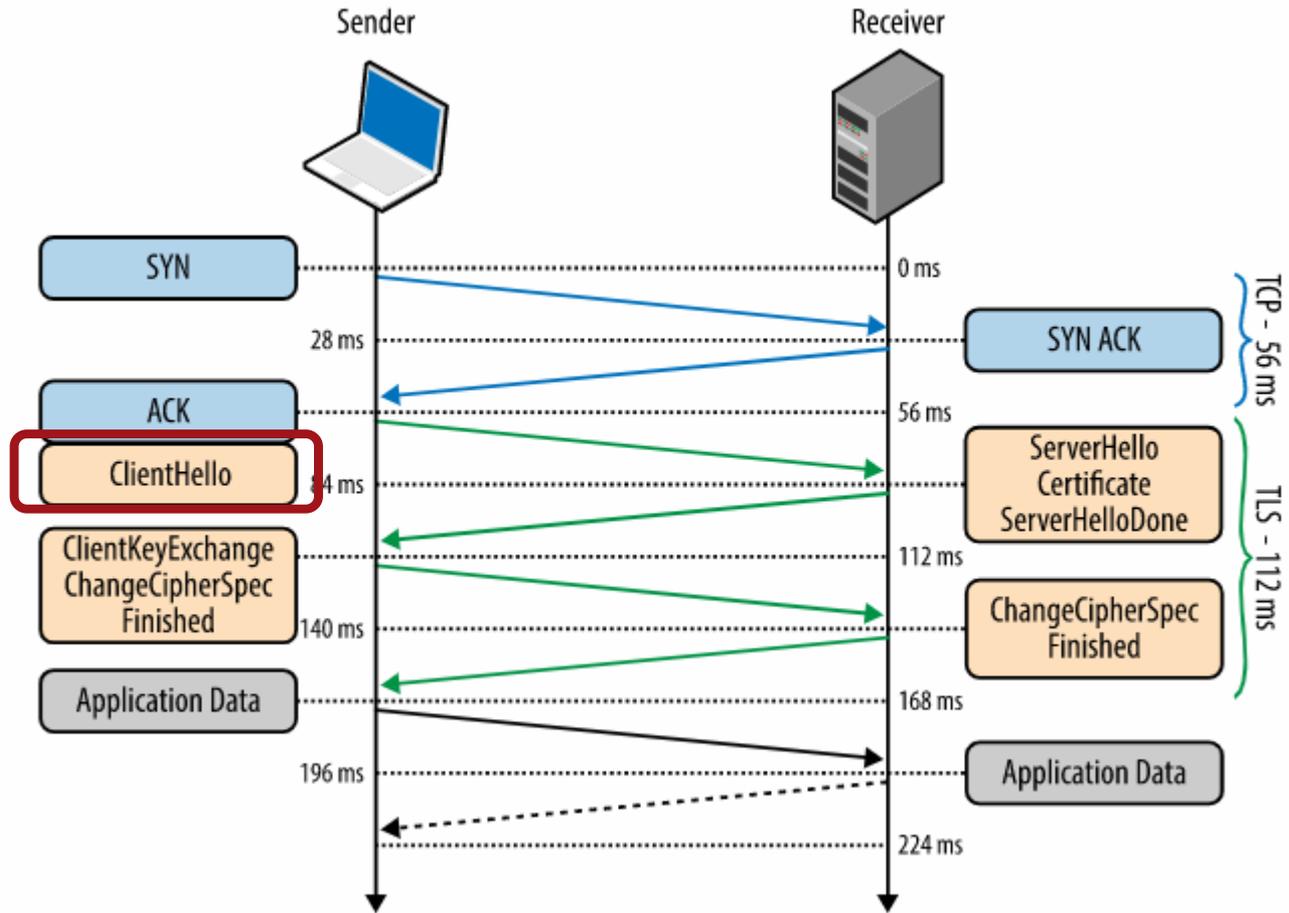
# TLS Protection against MitM Attacks

31



# TLS Handshake Extensions

32



# TLS Handshake Extensions in the Client Hello Message

33

```

  > Server Name Indication extension
      Server Name list length: 28
      Server Name Type: host_name (0)
      Server Name length: 25
      Server Name: lh3.googleusercontent.com
> Extension: extended_master_secret (len=0)
> Extension: renegotiation_info (len=1)
  > Extension: supported_groups (len=10)
      Type: supported_groups (10)
      Length: 10
      Supported Groups List Length: 8
      > Supported Groups (4 groups)
          Supported Group: Reserved (GREASE) (0x3a3a)
          Supported Group: x25519 (0x001d)
          Supported Group: secp256r1 (0x0017)
          Supported Group: secp384r1 (0x0018)
> Extension: ec_point_formats (len=2)
> Extension: session_ticket (len=0)
> Extension: application_layer_protocol_negotiation (len=14)
> Extension: status_request (len=5)
> Extension: signature_algorithms (len=18)
> Extension: signed_certificate_timestamp (len=0)
> Extension: key_share (len=43)
> Extension: psk_key_exchange_modes (len=2)
> Extension: supported_versions (len=7)
> Extension: compress_certificate (len=3)
> Extension: application_settings (len=5)
> Extension: Reserved (GREASE) (len=1)
> Extension: padding (len=190)
```

- These provide additional info to the server
- A few notable examples:
  - Supported elliptic curves
  - Server Name Indication
    - A client indicates which hostname it is attempting to connect to at the start of the handshake process
    - This allows a server to present one of multiple possible certificates on the same IP address and TCP port number and hence allows multiple secure (HTTPS) websites to be served by the same IP address without requiring all those sites to use the same certificate

# TLS Handshake Extensions in the Client Hello Message

34

```
Extensions Length: 403
> Extension: Reserved (GREASE) (len=0)
v Extension: server_name (len=30)
  Type: server_name (0)
  Length: 30
  > Server Name Indication extension
> Extension: extended_master_secret (len=0)
> Extension: renegotiation_info (len=1)
> Extension: supported_groups (len=10)
> Extension: ec_point_formats (len=2)
> Extension: session_ticket (len=0)
> Extension: application_layer_protocol_negotiation (len=14)
> Extension: status_request (len=5)
v Extension: signature_algorithms (len=18)
  Type: signature_algorithms (13)
  Length: 18
  Signature Hash Algorithms Length: 16
  v Signature Hash Algorithms (8 algorithms)
    > Signature Algorithm: ecdsa_secp256r1_sha256 (0x0403)
    > Signature Algorithm: rsa_pss_rsae_sha256 (0x0804)
    > Signature Algorithm: rsa_pkcs1_sha256 (0x0401)
    > Signature Algorithm: ecdsa_secp384r1_sha384 (0x0503)
    > Signature Algorithm: rsa_pss_rsae_sha384 (0x0805)
    > Signature Algorithm: rsa_pkcs1_sha384 (0x0501)
    > Signature Algorithm: rsa_pss_rsae_sha512 (0x0806)
    > Signature Algorithm: rsa_pkcs1_sha512 (0x0601)
  > Extension: signed_certificate_timestamp (len=0)
v Extension: key_share (len=43)
  Type: key_share (51)
  Length: 43
  v Key Share extension
    Client Key Share Length: 41
    > Key Share Entry: Group: Reserved (GREASE), Key Exchange length: 1
    > Key Share Entry: Group: x25519, Key Exchange length: 32
```

## □ Signature hash algorithms

- In TLS1.2 only, the client *MAY* include the *signatureAlgorithms* extension indicating what types of signatures it supports verifying
- This includes the signatures on the certificates in the server's chain
- This feature is dropped again in TLS 1.3

# ServerHello (Wireshark Screenshot)

35

```
▼ Transport Layer Security
  ▼ TLSv1.3 Record Layer: Handshake Protocol: Server Hello
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 122
  ▼ Handshake Protocol: Server Hello
    Handshake Type: Server Hello (2)
    Length: 118
    Version: TLS 1.2 (0x0303)
    Random: 77b42be2bd78b5c653da92f8624bf3ff90b742ba4ac632f67e6008b52aa4d00f
    Session ID Length: 32
    Session ID: 07c2d49a4554a1463c42de69738c7b645dbc5691c301e26bb3663df24e965f37
    Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
    Compression Method: null (0)
    Extensions Length: 46
  ▼ Extension: key_share (len=36)
    Type: key_share (51)
    Length: 36
    > Key Share extension
  ▼ Extension: supported_versions (len=2)
    Type: supported_versions (43)
    Length: 2
    Supported Version: TLS 1.3 (0x0304)
    [JA3S Fullstring: 771,4865,51-43]
    [JA3S: eb1d94daa7e0344597e756a1fb6e7054]
  ▼ TLSv1.3 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
    Content Type: Change Cipher Spec (20)
    Version: TLS 1.2 (0x0303)
    Length: 1
    Change Cipher Spec Message
```

- ❑ Highest TLS version supported
- ❑ Random 32 byte nonce (contains a timestamp)
- ❑ Client session id
- ❑ Chosen cipher suite
- ❑ List of supported data compression methods, obsolete with TLS 1.3
- ❑ List of extensions
- ❑ Again, all plaintext!

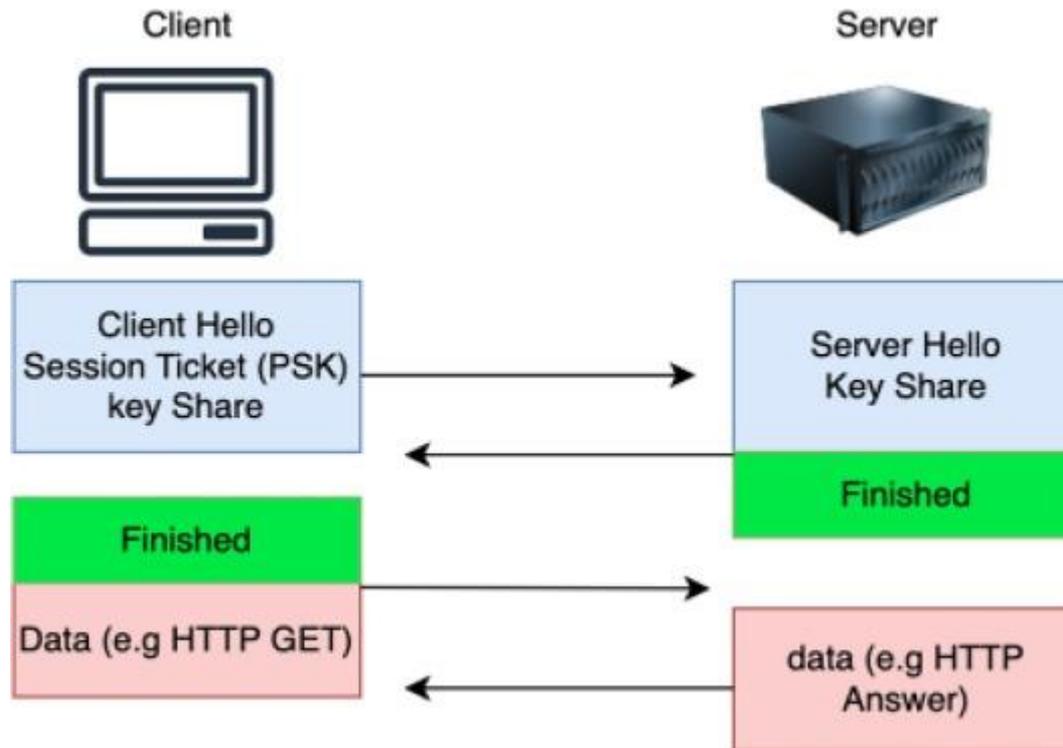
# TLS 1.2 Session Resumption

36

- Assume a client wants to reconnect to a server it has previously communicated with
- If the client still has the negotiated cipher suite and keys from the previous handshake cached, it can send the server the previously used session id in the *ClientHello* message
- If the server has cached all this data too, it can shorten the handshake
- However, it still requires a round trip to verify the session, which can introduce some latency
  - Otherwise, a full new session negotiation is required, which will generate a new session ID, and which will take longer
- If a browser requires multiple connections to the same host (e.g., when HTTP/1.x is used), it will often wait for the first TLS negotiation to complete before opening additional connections to the same server, such that they can be "resumed" and reuse the same session parameters
- On the other hand, caching the parameters of many client sessions over long periods of time does not scale and it rarely used

# TLS 1.2 Session Resumption

37

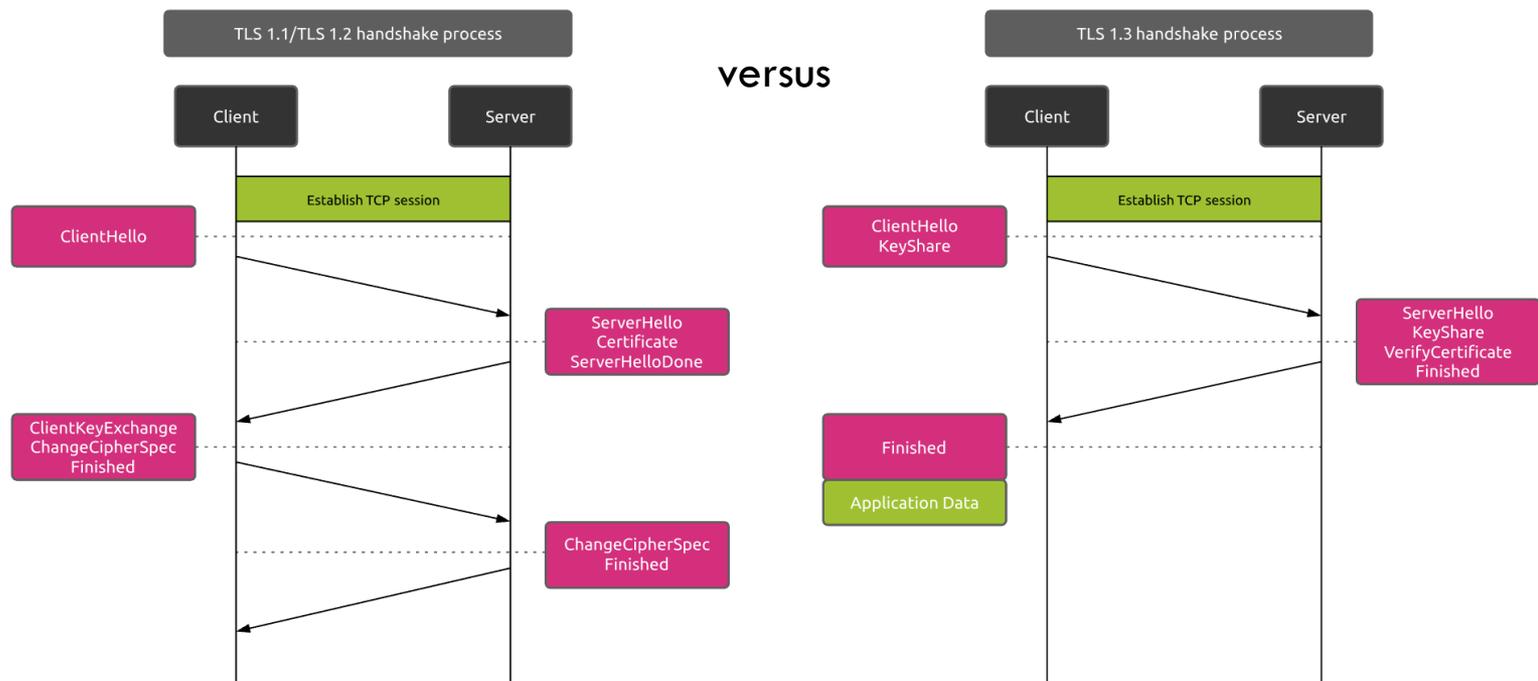


- Here all key negotiation steps are excluded

# TLS 1.3 and 1-Round Trip Time (1-RTT)

38

- Beside only supporting a streamlined ciphersuite for key negotiation (ECDHE only), TLS 1.3 also supports a new accelerated handshake process called 1-RTT



# The key\_share Extension

39

```
> Transmission Control Protocol, Src Port: 63377, Dst Port: 443, Seq: 1, Ack: 1, Len: 517
  > Transport Layer Security
    > TLv1.3 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 512
    > Handshake Protocol: Client Hello
      Handshake Type: Client Hello (1)
      Length: 508
      Version: TLS 1.2 (0x0303)
      Random: 5628afe2a5afa352d8a3336c39307da39b13ec3d009e4c9f9ef622ae51df6e49
      Session ID Length: 32
      Session ID: 07c2d49a4554a1463c42de69738c7b645dbc5691c301e26bb3663df24c965f37
      Cipher Suites Length: 32
    > Cipher Suites (16 suites)
      Compression Methods Length: 1
    > Compression Methods (1 method)
      Extensions Length: 403
    > Extension: Reserved (GREASE) (len=0)
    > Extension: server_name (len=30)
    > Extension: extended_master_secret (len=0)
    > Extension: renegotiation_info (len=1)
    > Extension: supported_groups (len=10)
    > Extension: ec_point_formats (len=2)
    > Extension: session_ticket (len=0)
    > Extension: application_layer_protocol_negotiation (len=14)
    > Extension: status_request (len=5)
    > Extension: signature_algorithms (len=18)
    > Extension: signed_certificate_timestamp (len=0)
    > Extension: key_share (len=43)
    > Extension: psk_key_exchange_modes (len=2)
    > Extension: supported_versions (len=7)
    > Extension: compress_certificate (len=3)
    > Extension: application_settings (len=5)
    > Extension: Reserved (GREASE) (len=1)
    > Extension: padding (len=190)
```

- ❑ This seems to suggest that the client is requesting a TLS 1.2 handshake
- ❑ A TLS 1.3 client hello looks superficially exactly like a TLS 1.2 handshake, right down to the version number
- ❑ If the server only understands TLS 1.2, it will just negotiate a TLS 1.2 handshake as before
- ❑ However, the new *ClientHello* extension *key\_share* indicates that the client understands version 1.3

# The key\_share Extension

40

```
Extensions Length: 403
> Extension: Reserved (GREASE) (len=0)
✓ Extension: server_name (len=30)
  Type: server_name (0)
  Length: 30
  > Server Name Indication extension
> Extension: extended_master_secret (len=0)
> Extension: renegotiation_info (len=1)
> Extension: supported_groups (len=10)
> Extension: ec_point_formats (len=2)
> Extension: session_ticket (len=0)
> Extension: application_layer_protocol_negotiation (len=14)
> Extension: status_request (len=5)
> Extension: signature_algorithms (len=18)
> Extension: signed_certificate_timestamp (len=0)
✓ Extension: key_share (len=43)
  Type: key_share (51)
  Length: 43
  ✓ Key Share extension
    Client Key Share Length: 41
    > Key Share Entry: Group: Reserved (GREASE), Key Exchange length: 1
    ✓ Key Share Entry: Group: x25519, Key Exchange length: 32
      Group: x25519 (29)
      Key Exchange Length: 32
      Key Exchange: e17fc347fe2e706a1b6bdb857a224161ca7e71b23e8868fdc
    > Extension: psk key exchange modes (len=2)
```

- ❑ In TLS 1.2, the *ClientKeyExchange* message is used to *kick-off the key exchange*
- ❑ This is now complemented by a method where the client presents the server with a ECDHE key exchange right at the start
- ❑ The idea is that the client just goes ahead and assumes that the server will select its preferred key exchange method and returns its ECDHE parameter
- ❑ If the server selects a different key exchange method, it will respond with a *RetryHelloRequest* message (not shown here) which restarts the handshake; this can be the result of either:
  - ▣ An ECDHE group that is not supported by the server
  - ▣ A server (security) policy that necessitate the use of different ECDH parameters than those proposed by the client
- ❑ In most cases the server will support the preferred key exchange method, so the handshake is shorter

# The *key\_share* Extension in both *ClientHello* (Left) and *ServerHello* (Right)

41

```
Extensions Length: 403
> Extension: Reserved (GREASE) (len=0)
✓ Extension: server_name (len=30)
  Type: server_name (0)
  Length: 30
  > Server Name Indication extension
> Extension: extended_master_secret (len=0)
> Extension: renegotiation_info (len=1)
> Extension: supported_groups (len=10)
> Extension: ec_point_formats (len=2)
> Extension: session_ticket (len=0)
> Extension: application_layer_protocol_negotiation (len=14)
> Extension: status_request (len=5)
> Extension: signature_algorithms (len=18)
> Extension: signed_certificate_timestamp (len=0)
✓ Extension: key_share (len=43)
  Type: key_share (51)
  Length: 43
  ✓ Key Share extension
    Client Key Share Length: 41
    > Key Share Entry: Group: Reserved (GREASE), Key Exchange length: 1
    ✓ Key Share Entry: Group: x25519, Key Exchange length: 32
      Group: x25519 (29)
      Key Exchange Length: 32
      Key Exchange: e17fc347fe2e706a1b6bdb857a224161ca7c71b23e8868fdc
  > Extension: psk key exchange modes (len=2)
```

```
Handshake Protocol: Server Hello
  Handshake Type: Server Hello (2)
  Length: 118
  Version: TLS 1.2 (0x0303)
  Random: 77b42be2bd78b5c653da92f8624bf3ff90b742ba4ac632f67e6008b5...
  Session ID Length: 32
  Session ID: 07c2d49a4554a1463c42de69738c7b645dbc5691c301e26bb366...
  Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
  Compression Method: null (0)
  Extensions Length: 46
  ✓ Extension: key_share (len=36)
    Type: key_share (51)
    Length: 36
    ✓ Key Share extension
      ✓ Key Share Entry: Group: x25519, Key Exchange length: 32
        Group: x25519 (29)
        Key Exchange Length: 32
        Key Exchange: ff210582cc0c2bcf10fd054cb945459fc82a325dd1...
    > Extension: supported_versions (len=2)
      Type: supported_versions (43)
      Length: 2
      Supported Version: TLS 1.3 (0x0304)
      [JA3S Fullstring: 771,4865,51-43]
      [JA3S: eb1d94daa7e0344597e756a1fb6e7054]
  TLSv1.3 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
```

- Note that in the server response all messages after the *ServerHello* message are already encrypted

# Cipher Suite (Wireshark Screenshot)

42

```
▼ Cipher Suites (16 suites)
  Cipher Suite: Reserved (GREASE) (0x7a7a)
  Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
  Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
  Cipher Suite: TLS_CHACHA20_POLY1305_SHA256 (0x1303)
  Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
  Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
  Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
  Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
  Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xc0a9)
  Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xc0aa)
  Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
  Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
  Cipher Suite: TLS_RSA_WITH_AES_128_GCM_SHA256 (0x009c)
  Cipher Suite: TLS_RSA_WITH_AES_256_GCM_SHA384 (0x009d)
  Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
  Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
```

- The highlighted cipher suites are used in TLS 1.3
- The negotiable bits are:
  - 128- or 256-bit AES in GCM mode, or
  - 256-bit ChaCha20 combined with POLY1305
    - ChaCha20 is a stream cipher
    - POLY1305 is a hash function, here used for authenticated encryption
  - SHA256 or SHA384 hashing
- The fixed elements are:
  - ECDHE using Curve25519
  - Message authentication using RSA or ECDSA
    - Depending if the server certificate contains a public RSA or EC key

# Review a TLS Handshake with OpenSSL

45

1. Init TLS connection:

```
openssl s_client -connect universityofgalway.ie:443
```

2. Review the output on screen

3. You may decode the server certificate via

<https://www.sslshopper.com/certificate-decoder.html>

# Review a TLS Handshake with OpenSSL

46

- Connection Status:
  - CONNECTED(00000003): Indicates that the connection to the server was successful.
- Certificate Verification:
  - depth=2, depth=1, depth=0: These lines show the verification process of the certificate chain. Each depth level represents a certificate in the chain, starting from the root CA (depth=2) to the server's certificate (depth=0).
  - verify return:1: Indicates that the certificate at each depth level was successfully verified.
- Certificate Chain:
  - Lists the certificates in the chain, including the subject (s:) and issuer (i:) details. The chain starts from the server's certificate and goes up to the root CA.
- Server Certificate:
  - The server's certificate is displayed in PEM format, including details like the subject and issuer.
- Peer Information:
  - No client certificate CA names sent: Indicates that no client certificate authority names were sent.
  - Peer signing digest: SHA256: Specifies the digest algorithm used for signing.
  - Peer signature type: RSA-PSS: Specifies the signature algorithm used.
  - Server Temp Key: X25519, 253 bits: Indicates the temporary key used for key exchange.

# The HTTPS Protocol

- HTTPS (Hypertext Transfer Protocol Secure) is syntactically identical to the HTTP protocol, but operates on top of TLS (rather than TCP)
  - ▣ TLS on the other hand operates on top of TCP
- It provides secure client / (web) server HTTP data communication, while also allowing a client (i.e. web browser) to authenticate the (web) server, as part of the TLS handshake
- The default HTTPS port is 443

# The Importance of Server-Side Authentication: Pharming Scams

48

## What is it?

- ❑ Pharming scams use domain spoofing (in which the domain appears authentic) to redirect users to copies of popular websites where personal data like usernames, passwords and financial information can be 'farmed' and collected for fraudulent use

## How can it be achieved – Simple Pharming!

- ❑ Copy a website 1:1 and present it to the victim using a slightly different domain name



# The Importance of Server-Side Authentication: Pharming Scams

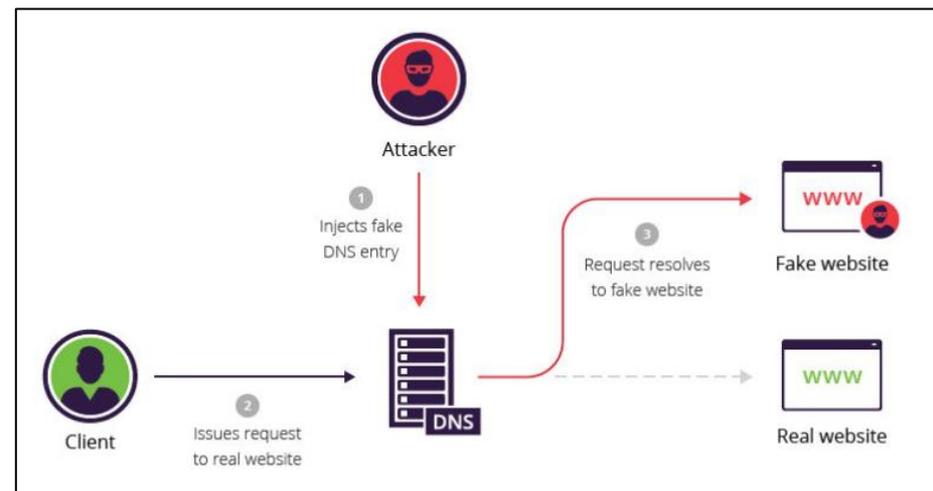
49

## What is it?

- ❑ Pharming scams use domain spoofing (in which the domain appears authentic) to redirect users to copies of popular websites where personal data like usernames, passwords and financial information can be 'farmed' and collected for fraudulent use

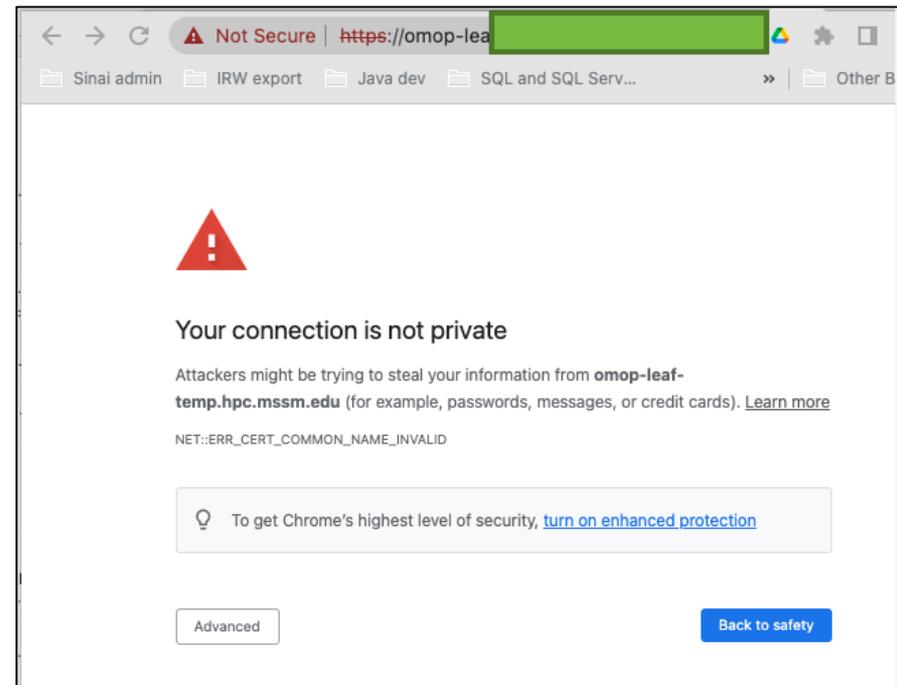
## How can it be achieved - DNS Spoofing!

- ❑ Similar to simple pharming, but also manipulate the DNS server to redirect DNS queries to the attacker's website, i.e. the same domain name is used
- ❑ Known as DNS poisoning, DNS cache poisoning or DNS spoofing



# Anti-Pharming Support in your Browser

- ❑ In DNS spoofing, the malicious server cannot support HTTPS or TLS, as it doesn't have the spoofed server's private key
  - ▣ It has its certificate though, but that's not enough to complete the TLS handshake
- ❑ All modern browsers pick up on this and abort the connection
- ❑ Also, users are warned if TCP rather than TLS is used (see image)



# Certificate Stapling

51

- In certificate stapling, the server appends all certificates in the path up to the root CA / RCA in a “Certificate” message, which is sent together with its *ServerHello* message to the client
- These stapled certificates are sent as
  - plaintext in TLS 1.2 (see Wireshark screenshot below)
  - ciphertext in TLS 1.3 (as all messages after the “Server Hello” message are already encrypted)

```
> Frame 12: 1072 bytes on wire (8576 bits), 1072 bytes captured (8576 bits) on interface \Device\NPF_{D65A8A53-7DBC-4AE2-93E1-1C9B99DCAC02}, id 0
> Ethernet II, Src: Sagemcom_5b:a3:57 (5c:b1:3e:5b:a3:57), Dst: IntelCor_a6:2e:6c (18:5e:0f:a6:2e:6c)
> Internet Protocol Version 4, Src: 13.79.243.64, Dst: 192.168.1.105
> Transmission Control Protocol, Src Port: 443, Dst Port: 56944, Seq: 2921, Ack: 518, Len: 1018
> [3 Reassembled TCP Segments (3938 bytes): #10(1460), #11(1460), #12(1018)]
< Transport Layer Security
  < TLSv1.2 Record Layer: Handshake Protocol: Multiple Handshake Messages
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 3933
  > Handshake Protocol: Server Hello
  < Handshake Protocol: Certificate
    Handshake Type: Certificate (11)
    Length: 3057
    Certificates Length: 3054
  < Certificates (3054 bytes)
    Certificate Length: 1838
  > Certificate: 3082072a30820612a00302010202100b103ee5deb9b4c931506d59591f7ecf300d06092a... (id-at-commonName=www.revenue.ie,id-at-organizationName=Office of the Revenue Commissioners,id-at-localityName=Dublin,id-at-countryName=IE,...
    Certificate Length: 1210
  > Certificate: 308204b63082039ea00302010202100c79a944b08c11952092615fe26b1d83300d06092a... (id-at-commonName=DigiCert SHA2 Extended Validation Server CA,id-at-organizationalUnitName=www.digicert.com,id-at-organizationName=DigiCert,...
  > Handshake Protocol: Certificate Status
  > Handshake Protocol: Server Key Exchange
  > Handshake Protocol: Server Hello Done
```



# OCSP Stapling

53

- Recall: The Online Certificate Status Protocol (OCSP) is a standard for checking the revocation status of X.509 digital certificates
  - ▣ An OCSP response is digitally signed and time-stamped by the CA (OCSP server) that confirmed the revocation status of a certificate
- In OCSP stapling
  - ▣ The client includes a "status\_request" extension in its *ClientHello* message
  - ▣ The server includes the OCSP response "Certificate Status" message in the *ServerHello* response
- This eliminates the need for a client to contact the CA, thereby improving overall performance
- However, the status of intermediate and root certificates is typically managed by separate OCSP checks

# The ServerHello OCSP Response

54

```
Handshake Protocol: Certificate Status
  Handshake Type: Certificate Status (22)
  Length: 475
  Certificate Status Type: OCSP (1)
  OCSP Response Length: 471
  OCSP Response
    responseStatus: successful (0)
    responseBytes
      ResponseType Id: 1.3.6.1.5.5.7.48.1.1 (id-pkix-ocsp-basic)
      BasicOCSPResponse
        tbsResponseData
          responderID: byKey (2)
            byKey: 3dd350a5d6a0adeef34a600a65d321d4f8f8d60f
            producedAt: Mar 10, 2023 23:06:29.000000000 GMT Standard Time
          responses: 1 item
            SingleResponse
              certID
                hashAlgorithm (SHA-1)
                  issuerNameHash: 49f4bd8a18bf760698c5de402d683b716ae4e686
                  issuerKeyHash: 3dd350a5d6a0adeef34a600a65d321d4f8f8d60f
                  serialNumber: 0x0b103ee5deb9b4c931506d59591f7ecf
                certStatus: good (0)
                thisUpdate: Mar 10, 2023 22:51:01.000000000 GMT Standard Time
                nextUpdate: Mar 17, 2023 22:06:01.000000000 GMT Standard Time
              signatureAlgorithm (sha256WithRSAEncryption)
                Padding: 0
                signature [...]: 6d31d2b16ded1b10b75ecd7d494facc5909f3954e781c4c6864815fcc
```

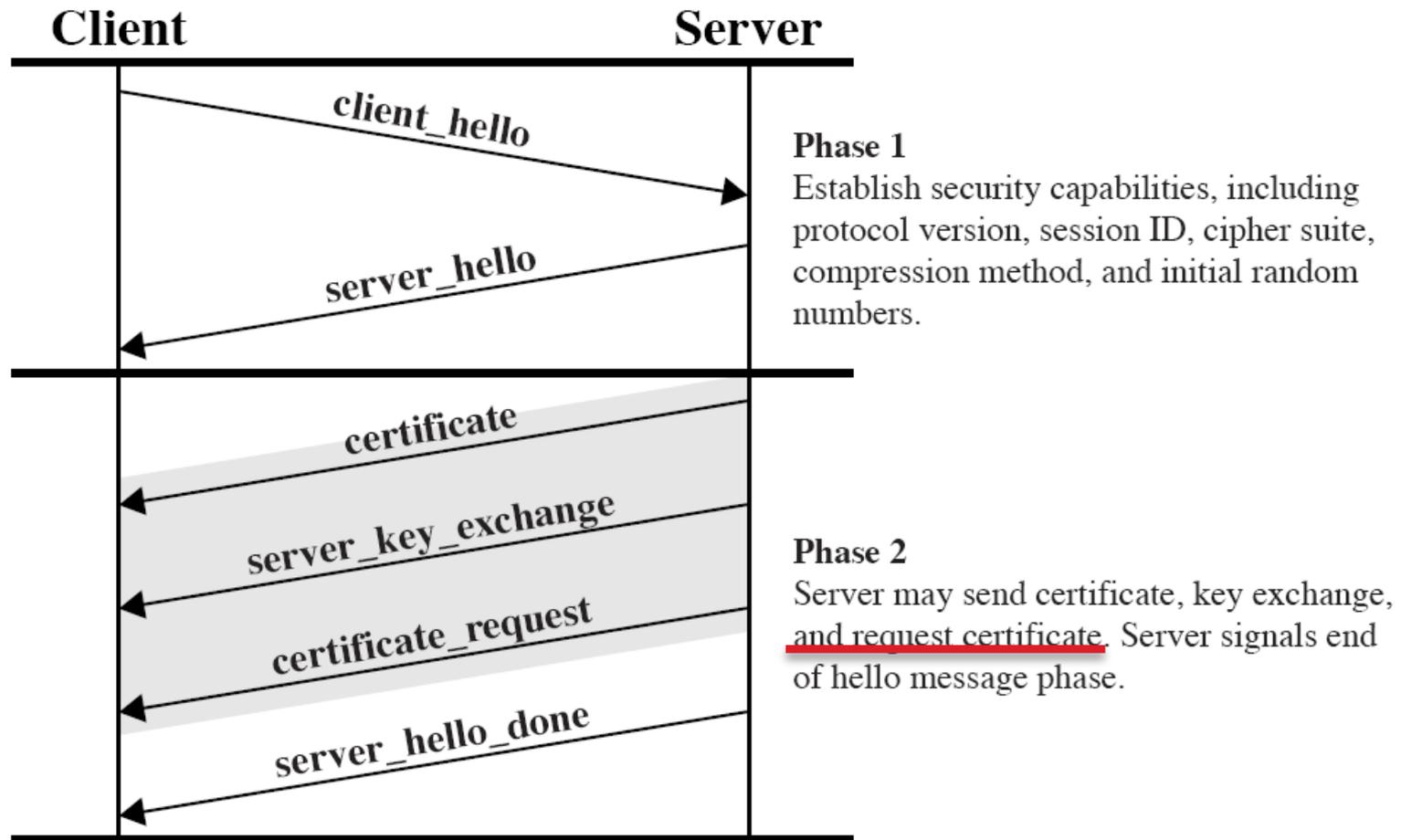
- Type (BasicResponse)
- ResponderID identifies the OCSP server via its DN issuer information, or its hashed public key (as shown here)
- CertId determines the cert that is being validated; using a hash algorithm (SHA-1) a hash of the issuer's DN, a hash of its public key, and the certificates serial number are provided
- certStatus (good)
- Validity period of OCSP response
- The entire message is digitally signed by the OCSP responder
- That's the signature

# Mutual Authentication (Server-Side and Client-Side Authentication)

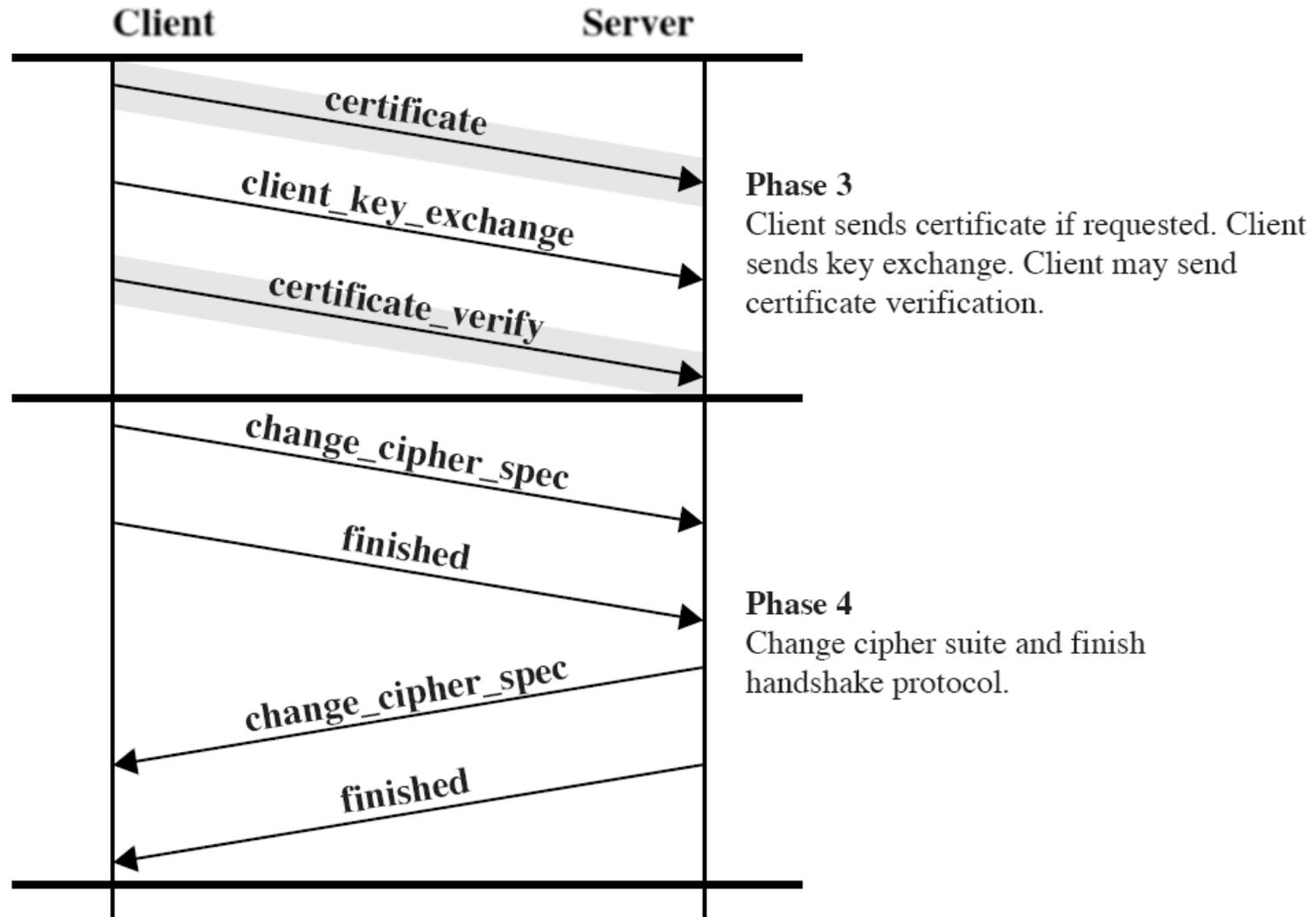
55

- Consider a scenario where both the server and the client need to mutually authenticate, e.g.
  - ▣ Server-to-server data communication
  - ▣ IoT sensor network communication
  - ▣ Online Revenue services where client (browser) needs to be authenticated too
- Mutual authentication is just an extension of the process as seen before with the difference that the client sends its certificate (chain) to the server too for verification

# Mutual Authentication I



# Mutual Authentication II



# In Summary

58

- ❑ TLS is the de-facto security protocol used in Internet data communication
- ❑ It went through a series of versions, and today only TLS 1.2 and TLS 1.3 are used
- ❑ TLS combines a lot of the foundation topics we've discussed in recent weeks
- ❑ Practically it is very hard to break TLS security, as the protocol went through various improvements over the years
- ❑ Therefore, from an attacker perspective, it is more promising to compromise a system by attacking either the client, the server, or the end user directly

# CT5191 NETWORK SECURITY & CRYPTOGRAPHY

## IPSEC

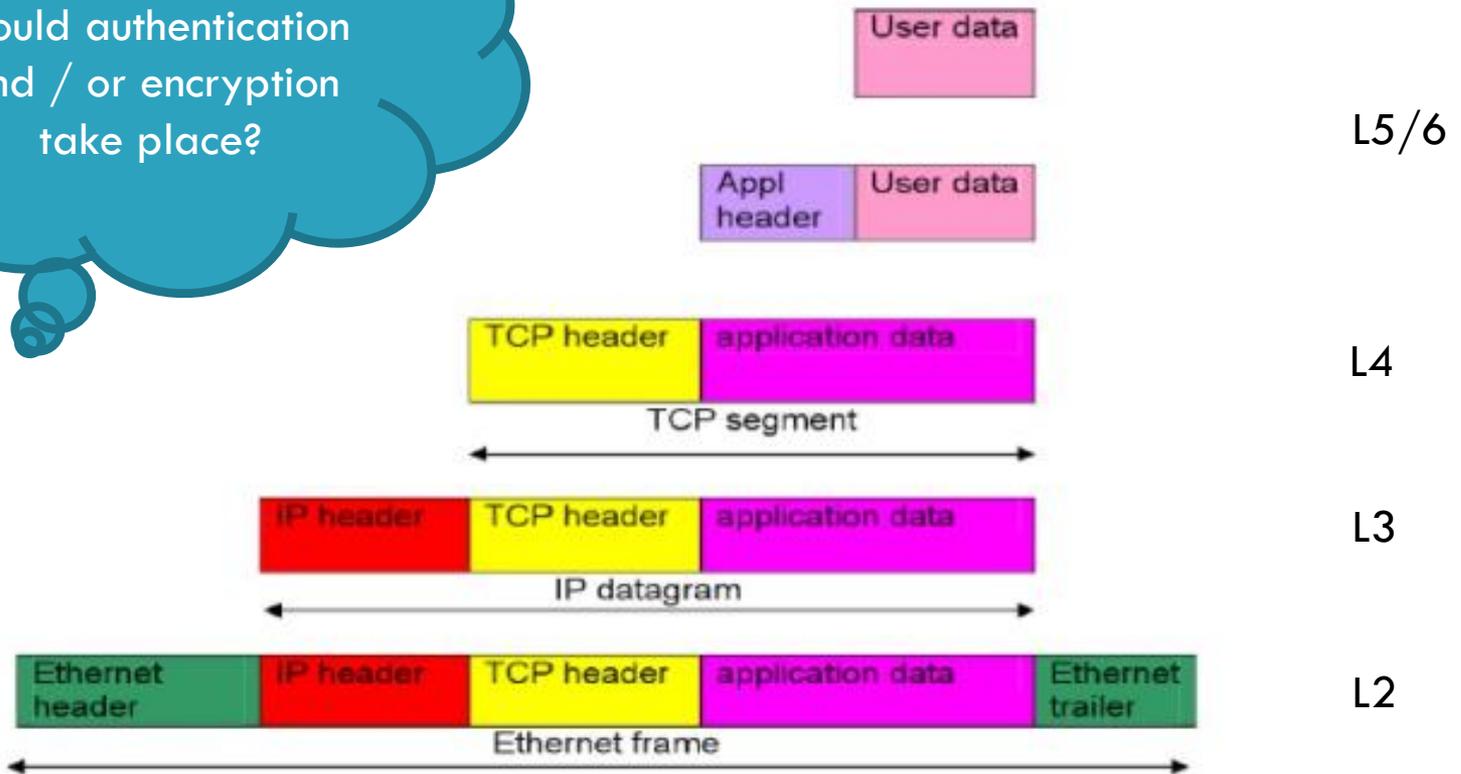
Dr. Michael Schukat



# Recap: TCP/ IP Header Hierarchy

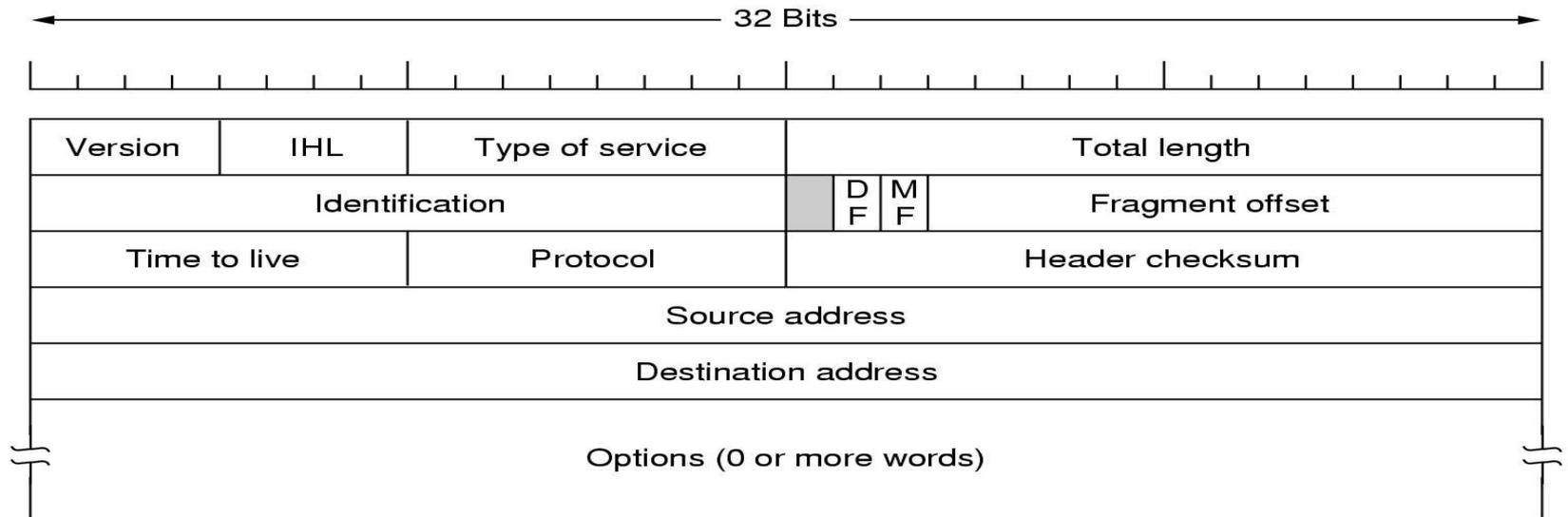
2

On what level(s)  
should authentication  
and / or encryption  
take place?



# Recap: Issues with the IP Protocol

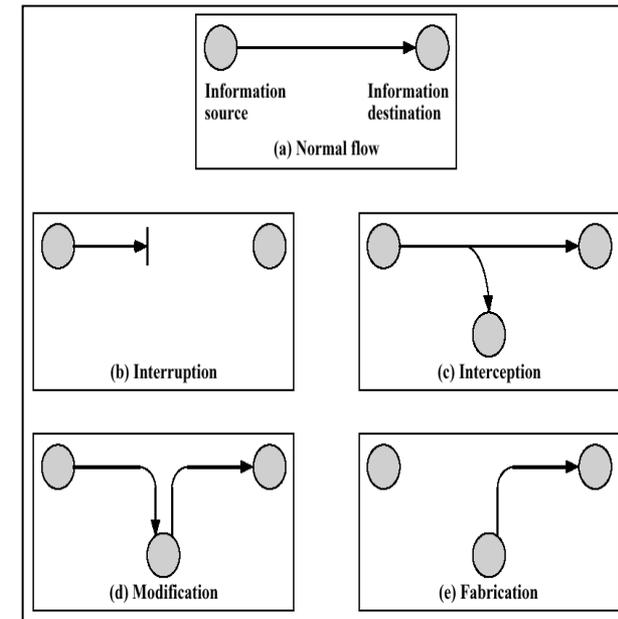
- ❑ IP payload is not encrypted (no confidentiality) and can be manipulated in transit
- ❑ IP header fields can be manipulated in transit (CRC can be adjusted on-the-fly → next slide)
  - ❑ IP addresses can be spoofed (no authentication)
- ❑ IP header has mutable fields that can change during datagram transport



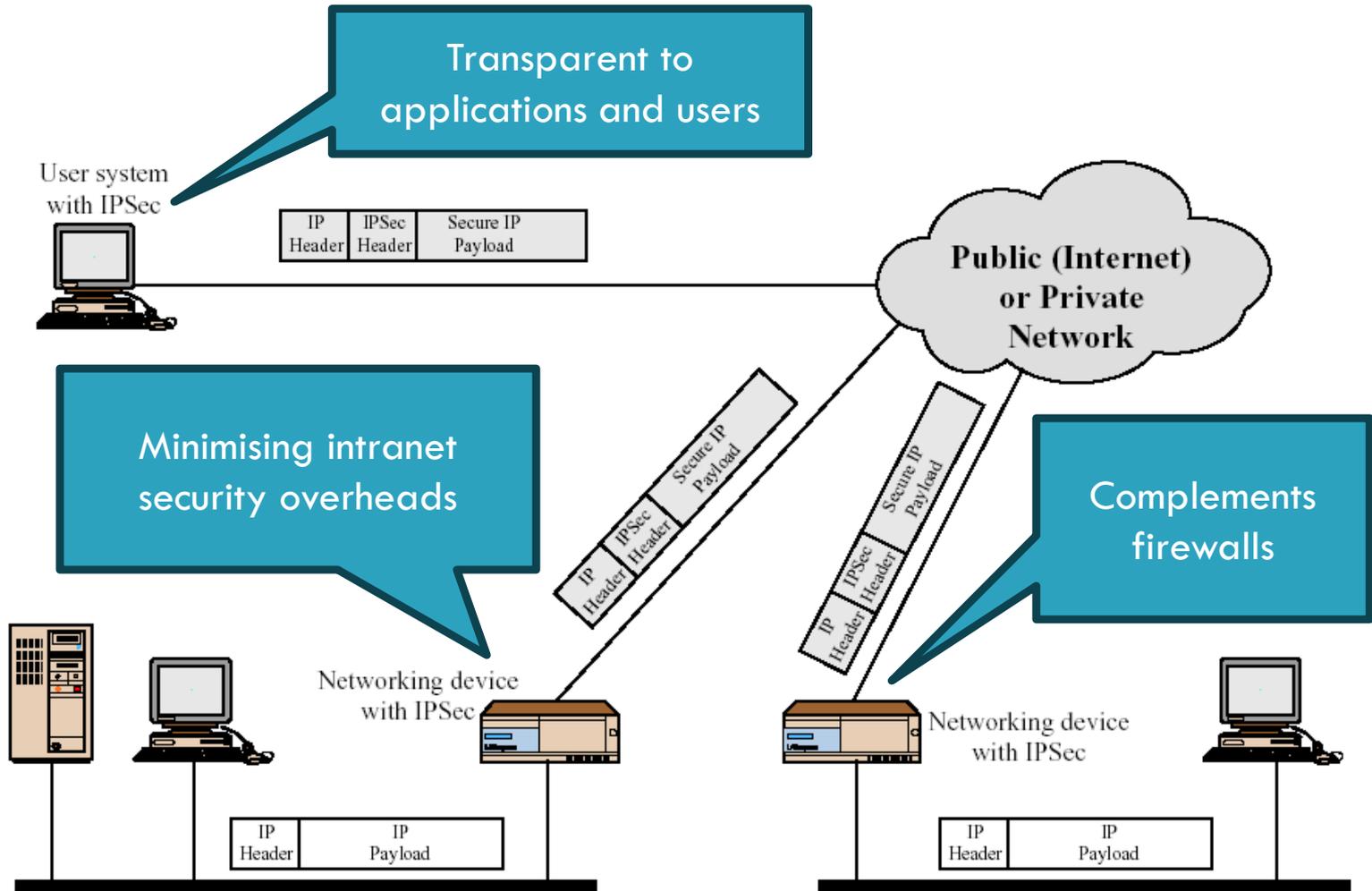
# IPsec Overview

4

- Protocol Standard to protect IP datagrams
- It provides:
  - ▣ Data origin authentication
    - → Protection against IP address spoofing
  - ▣ Connectionless data integrity authentication
    - → Protection against modification
  - ▣ Data content confidentiality
    - → Protection against interception
  - ▣ Anti-replay protection
    - Protection against replay attacks / modification
  - ▣ Limited traffic flow confidentiality
    - → Protection against interception
    - → (Limited) obfuscation of endpoint IP addresses



# Organisational Use of IPsec

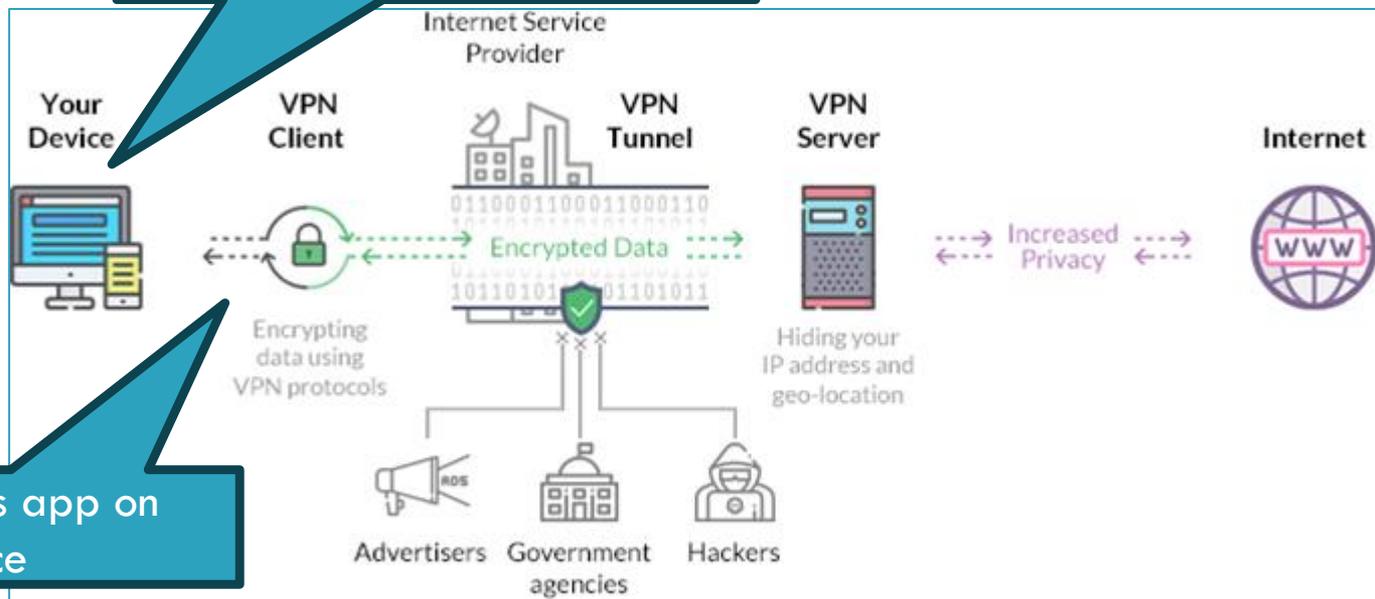


# IPsec Virtual Private Networks (VPN) for Individual Users

6

- VPN service providers enable their customers to protect their identity, as well as their data communication between their computer, across Wi-Fi / LAN and WAN to a trusted gateway (VPN server)

Transparent to applications and users



VPN client is app on device

# IPsec Services by Header Type

- ❑ IPsec is a network-layer security protocol that provides
  - ▣ IP payload encryption (for confidentiality) via **ESP (Encapsulating Security Payload)**
  - ▣ IP header and payload authentication via **AH (Authentication Header)**
  - ▣ Key management (not covered here)
- ❑ As an IP layer protocol extension, it provides secure Internet, LAN, and WAN communication

	AH	ESP (encryption only)	ESP (encryption plus authentication)
Access control	✓	✓	✓
Connectionless integrity	✓		✓
Data origin authentication	✓		✓
Rejection of replayed packets	✓	✓	✓
Confidentiality		✓	✓
Limited traffic flow confidentiality		✓	✓

# Security Associations (SA)

- Key concept for authentication and confidentiality for IP
- One-way relationship between sender and receiver
  - ▣ e.g. for a two-way secure peer relationship, two SAs (one for each host) are required
- A SA is uniquely identified by
  - ▣ Security parameter index (SPI)  
Unique identifier, which is carried in the IPsec AH and ESP headers
  - ▣ IP destination address
  - ▣ Security Protocol Identifier: indicates AH or ESP association

# SA and the Security Association Database (SAD)

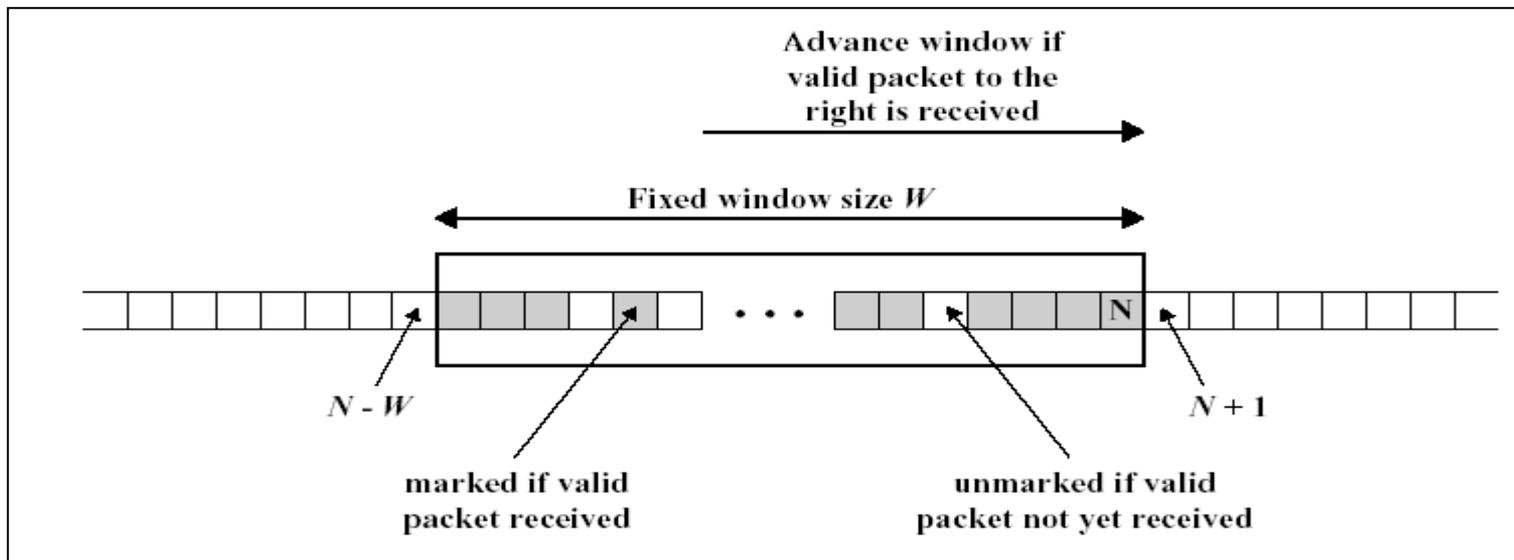
- The SAD contains the parameters associated with each SA, including
  - ▣ Sequence number counter:  
32-bit value for packet identification, which is part of AH or ESP header
  - ▣ Sequence Counter Overflow flag
  - ▣ Anti-replay window
    - Remark: The above 2 parameters are important to prevent replay attacks
  - ▣ AH information: Algorithm, key and key lifetime, etc.
  - ▣ ESP information: ditto
  - ▣ Lifetime of SA (and SPI)
  - ▣ IPSec protocol mode: Tunnel or transport mode

# Security Policy Database (SPD)

- Each point-to-point link (e.g. host-to-host) is associated with one or more SAs
- This association between links and SA(s) is stored in the SPD, using the following IP header fields (i.e. selectors) as keys:
  - ▣ Source / Destination IP address
  - ▣ Transport layer protocol
  - ▣ Source and destination ports
- For example, in order to process an outgoing IP packet,
  - ▣ its selectors are extracted and compared against the SPD entries
  - ▣ Zero or more SA references are returned, and their respective SA parameters are retrieved from the SAD
  - ▣ Subsequently each SA is processed
- In contrast, The SAs of incoming IPsec packets can be identified by their SPI

# The Anti-Replay Window

- A received protected package contains SA selectors, which allow to determine the required SA(s) in the SPD
- The SA entry within the SAD contains state information, e.g. parameters for replay window
- A protected package also contains a unique packet sequence number



# The Anti-Replay Window

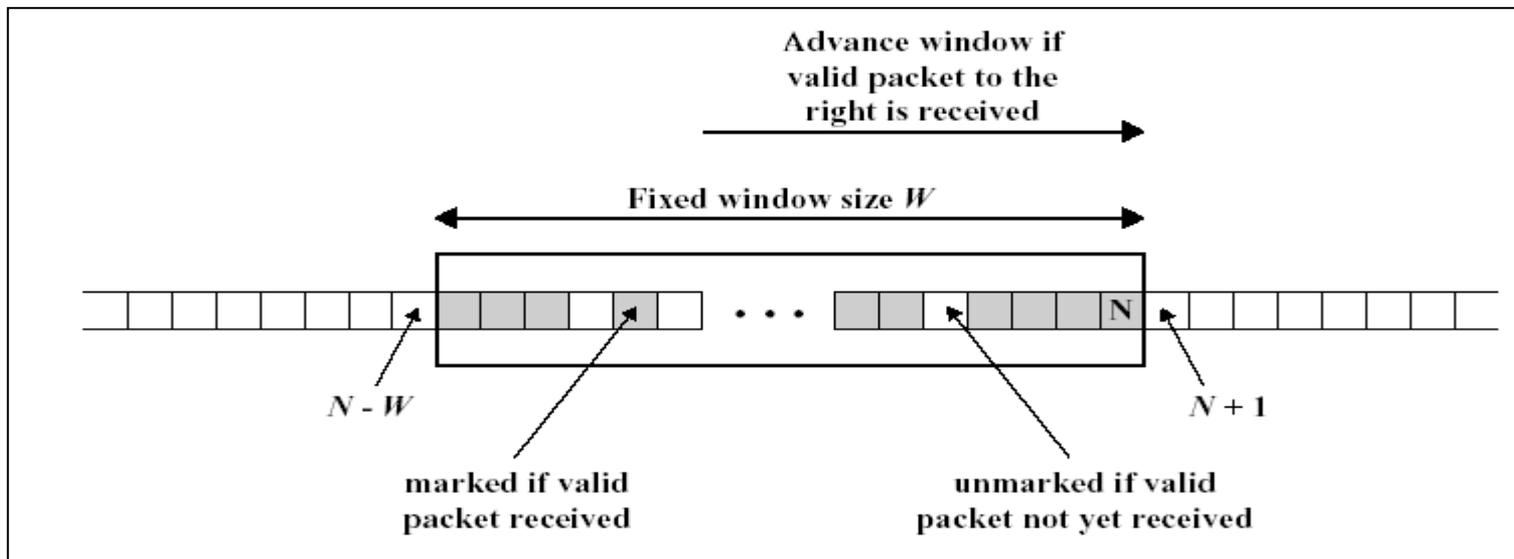
- Three scenarios for a received packet:
  - ▣ If packet falls within existing window, is new and is authenticated, the corresponding slot will be marked, and packet will be processed
  - ▣ If packet is right to the window, is new and is authenticated, the window will be advanced, slot will be marked, and packet will be processed
  - ▣ If packet is left to the window or authentication fails, it will be discarded, and an alarm will be raised
    - As this could be an indication for a replay attack

# Example

13

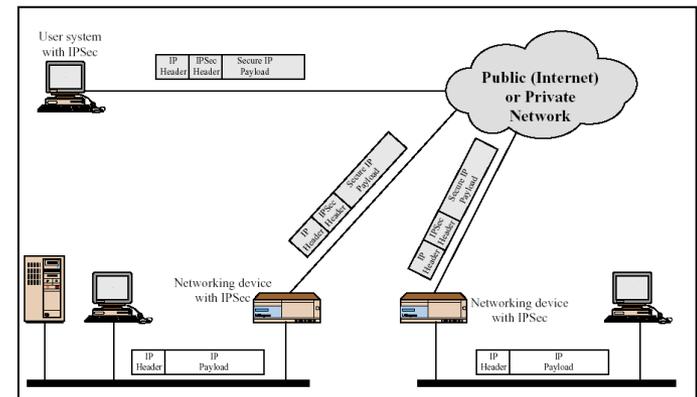
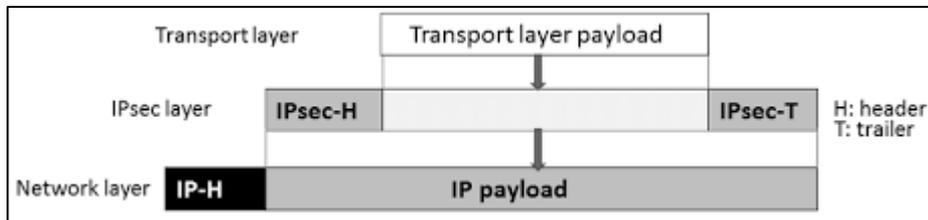
- Consider a receiver with  $W = 5$  and  $N = 33$
- Which of the following incoming (and authenticated) packets will be deemed as a replayed packet and discarded:
- 32, 29, 36, 38, 31, 35

M



# IPsec Transport Mode

- Here, the IP header is untouched, and only the payload can be encrypted (via ESP)
  - ▣ Therefore, the packet routing is kept intact
- Certain IP header fields (i.e. IP source / destination address) and the payload can be authenticated (via AH)
  - ▣ This prevents IP address spoofing, but also NAT, as network address translation invalidates the authenticator (see also NAT traversal)
- Also, transport and application layer are authenticated too, so they cannot be modified in any way in transit, for example by translating the port numbers, unless NAT traversal is used



# IPsec Tunnel Mode

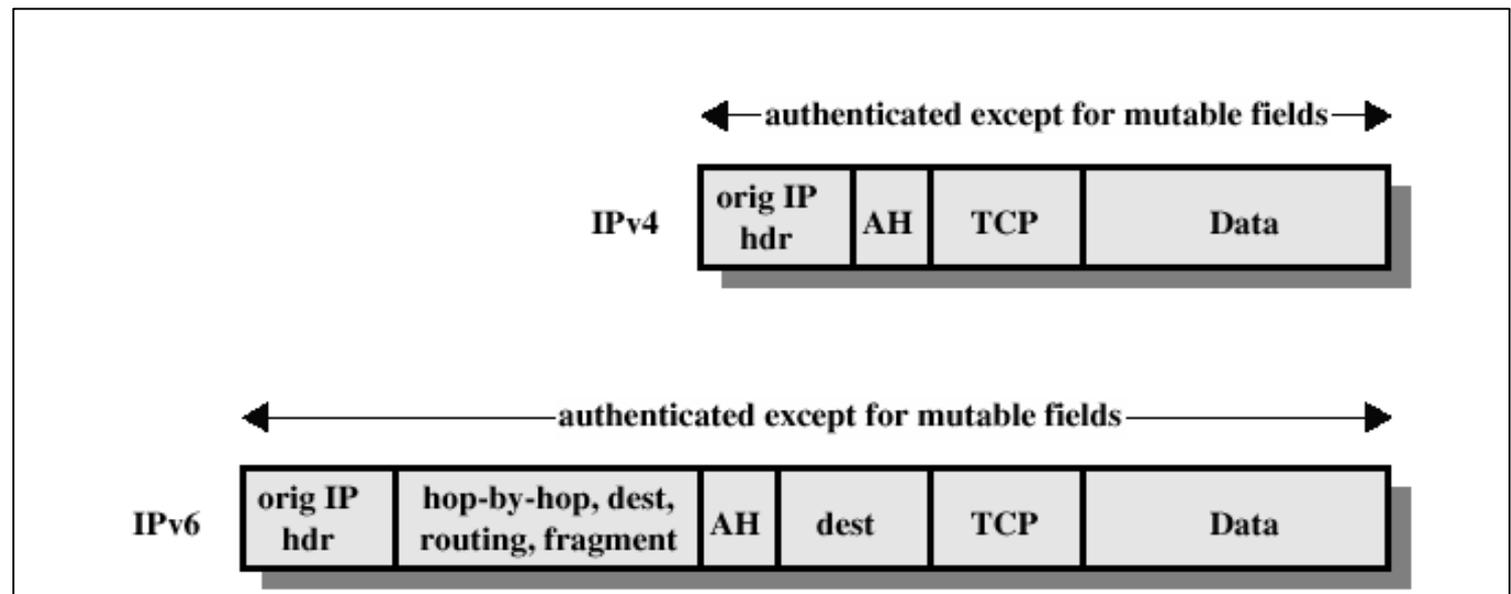
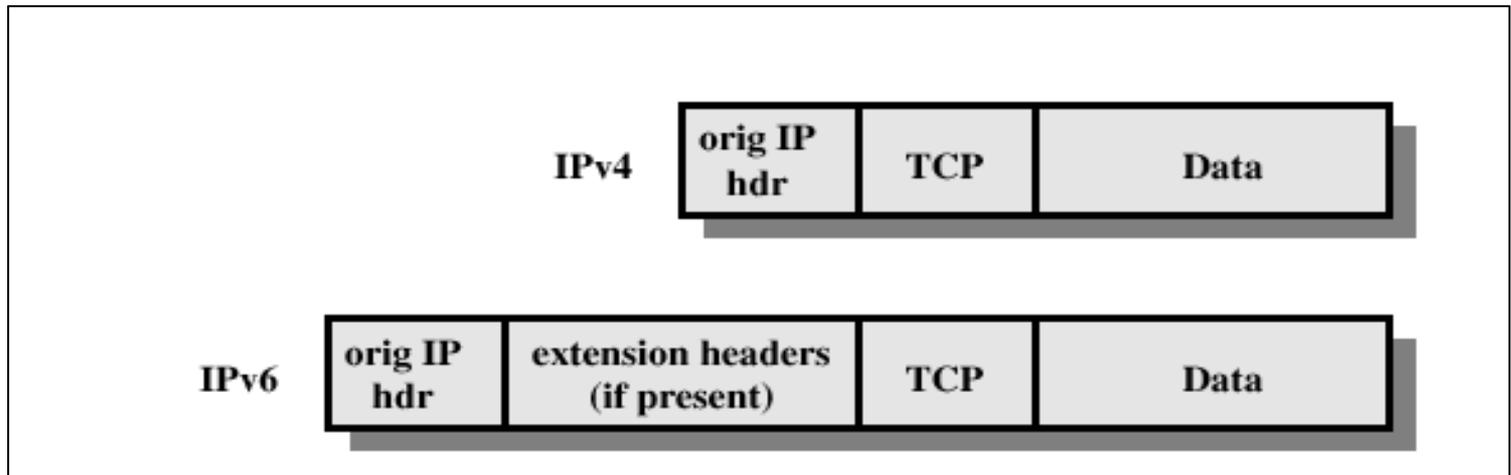
- ❑ Tunnel mode embeds an entire IP packet (as payload) into another (outer) IP packet
  - ▣ It secures the IP packet as a whole including its header(s)
- ❑ The IP datagram is delivered according to the outer IP header
- ❑ Typically for router-to-router or firewall-to-firewall VPN
  - ▣ Here IPsec is implemented in a security gateway (router/firewall) that secures all packets coming from within the intranet



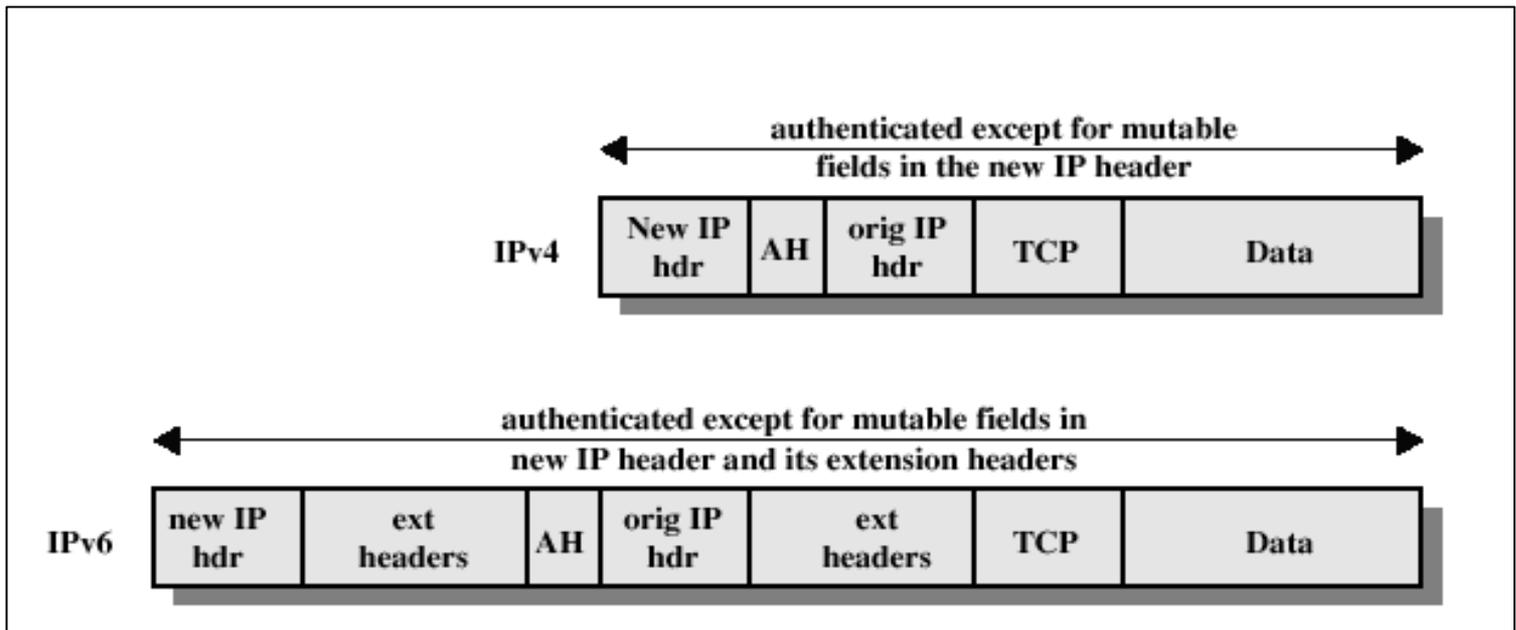
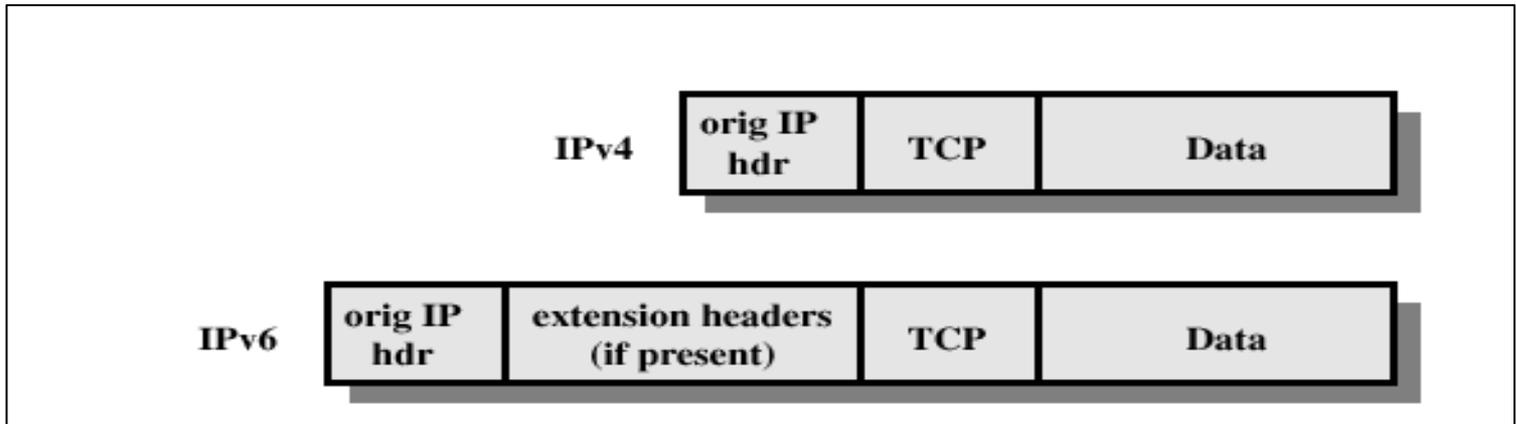
# Transport versus Tunnel Mode

	<b>Transport Mode SA</b>	<b>Tunnel Mode SA</b>
AH	Authenticates IP payload and selected portions of IP header and IPv6 extension headers.	Authenticates entire inner IP packet (inner header plus IP payload) plus selected portions of outer IP header and outer IPv6 extension headers.
ESP	Encrypts IP payload and any IPv6 extension headers following the ESP header.	Encrypts inner IP packet.
ESP with Authentication	Encrypts IP payload and any IPv6 extension headers following the ESP header. Authenticates IP payload but not IP header.	Encrypts inner IP packet. Authenticates inner IP packet.

# IPsec: AH in Transport Mode

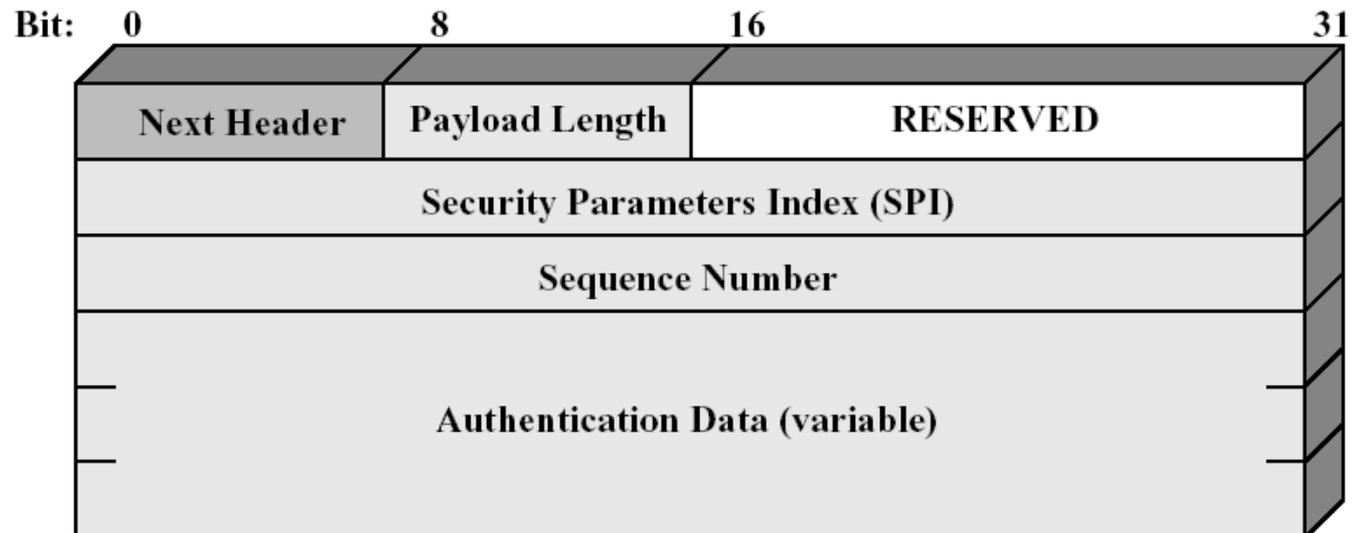


# IPsec: AH in Tunnel Mode



# The Authentication Header

- ❑ AH provides data integrity and authentication for IP packets
- ❑ AH prevents address spoofing and replay attacks
- ❑ Authentication Data is based on keyed hash function (→ later), so both parties share a secret key

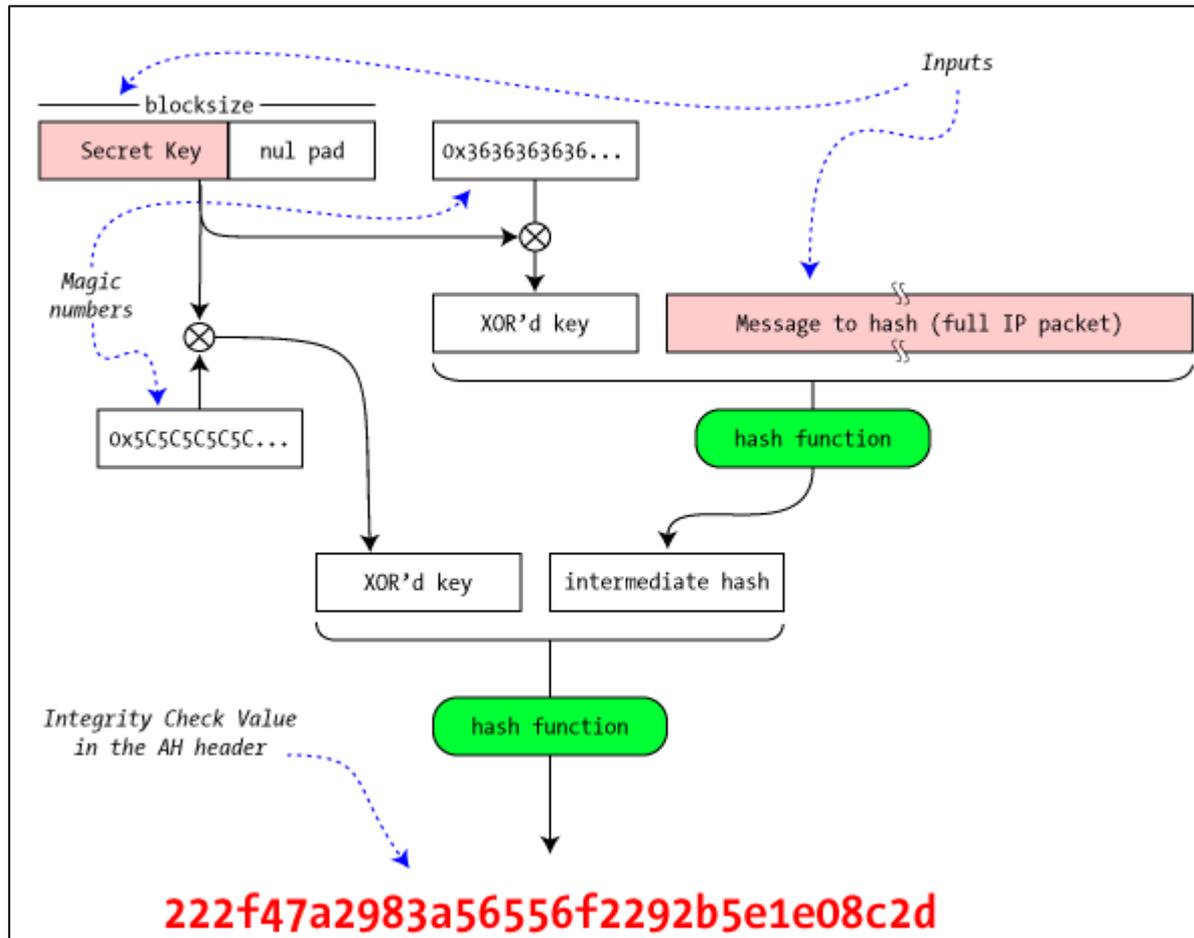


# AH Fields

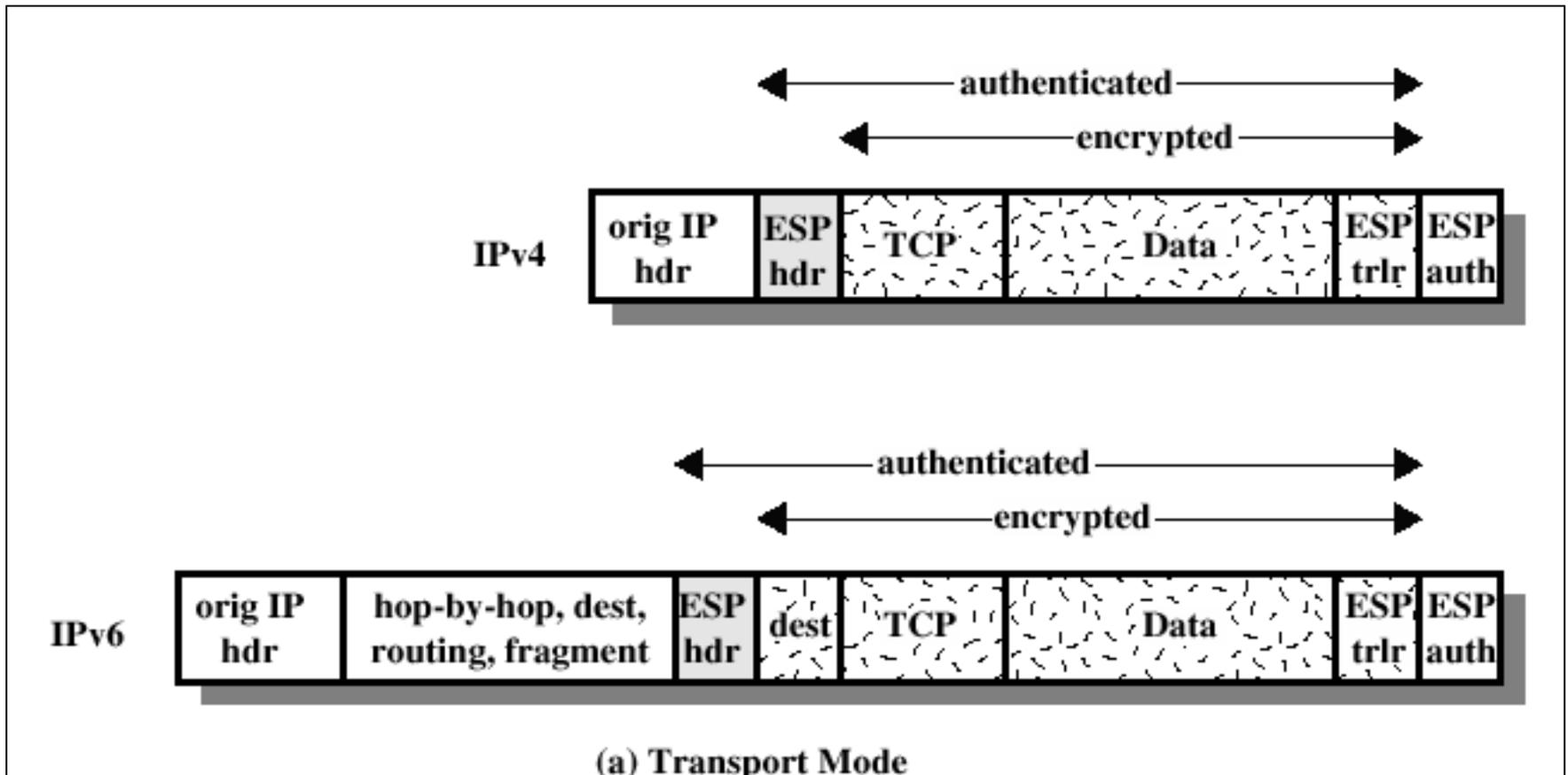
- ❑ **Next header** (8 bits): Identifies type of header following this header
- ❑ **Payload Length** (8 bits): Length of AH in 32-bit words minus 2
- ❑ **Reserved** (16 bits)
- ❑ **SPI** (32 bits): Identifies SA
- ❑ **Sequence Number** (32 bits): Unique incremented counter value
- ❑ **Authentication Data** (variable): Contains Integrity Check Value, i.e. the keyed hash value (next slide)

# FYI: ICV for AH Authentication

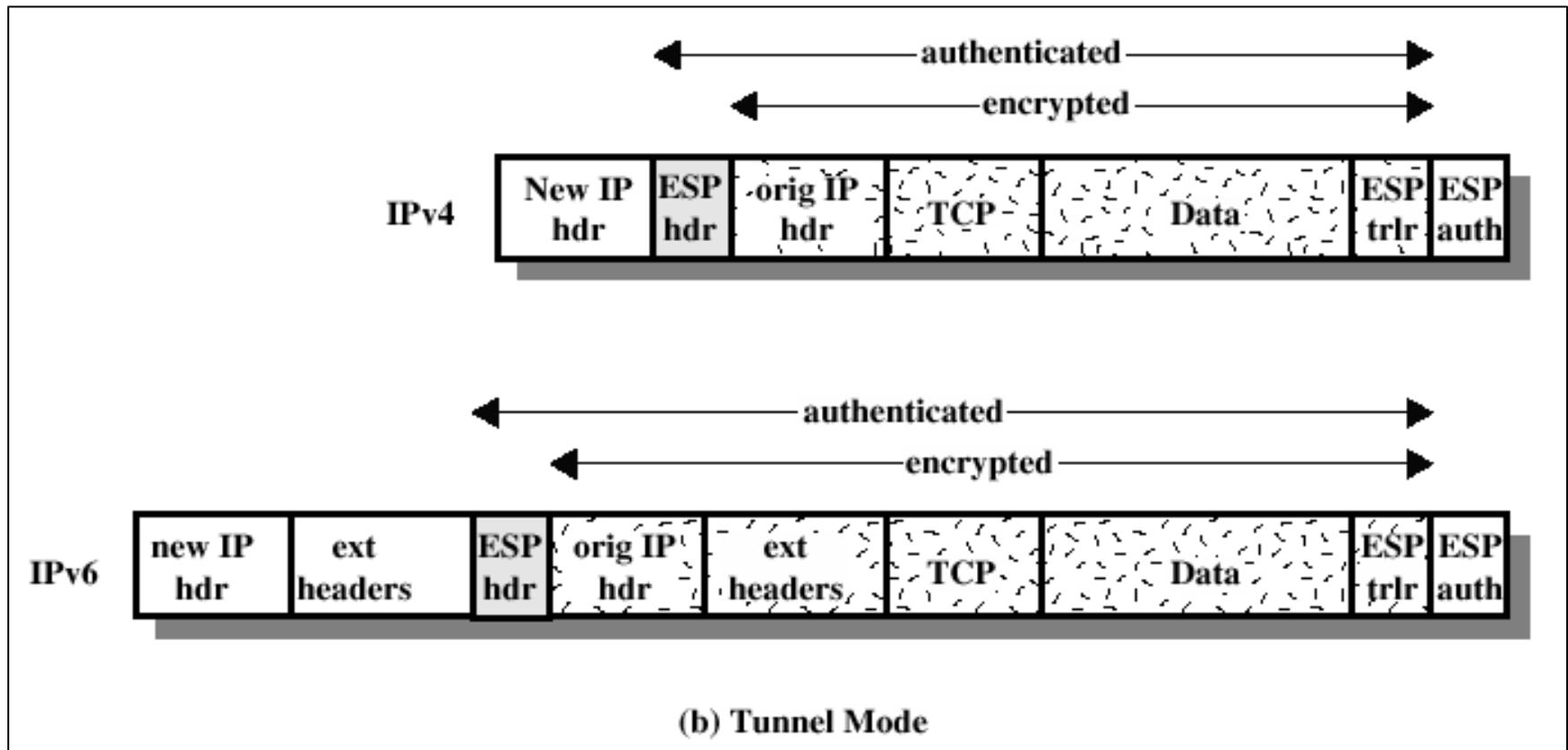
22



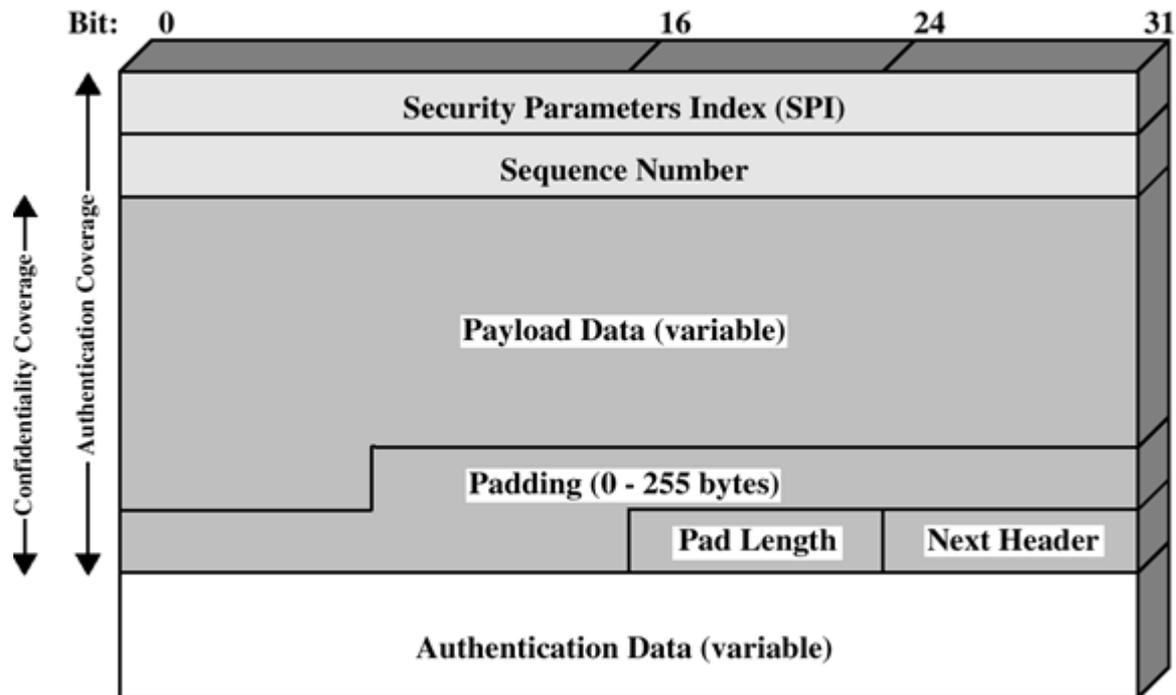
# ESP Encryption and (optional) Authentication in Transport Mode



# ESP Encryption and (optional) Authentication in Tunnel Mode



# The IPsec ESP Header

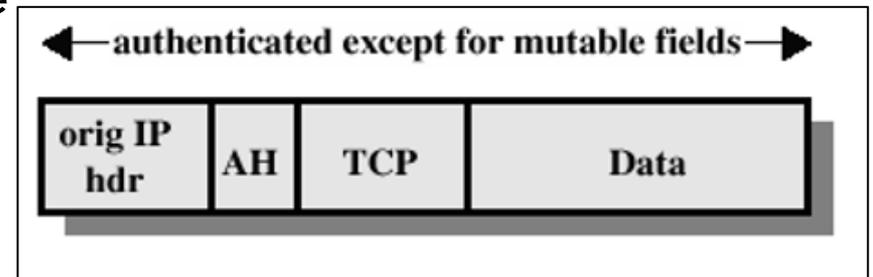
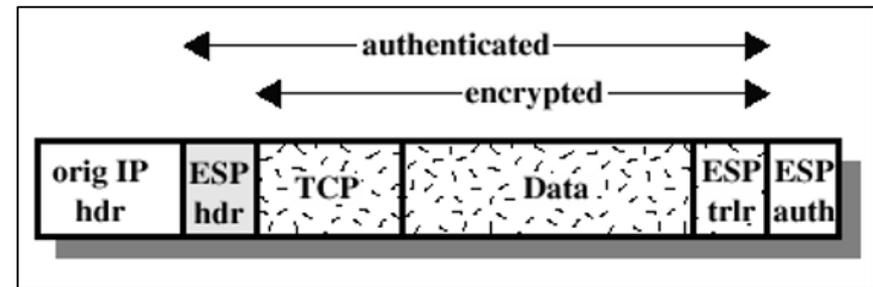


# ESP Header and Trailer

- ❑ ESP provides encryption using Triple-DES (obsolete by 2025) or AES, in CBC mode
- ❑ ESP header contains SPI and sequence number
- ❑ Hatched fields contain encoded payload
- ❑ ESP trailer contains
  - ▣ padding bytes
  - ▣ padding length
  - ▣ next header field
- ❑ Optional ESP auth trailer contains authentication data

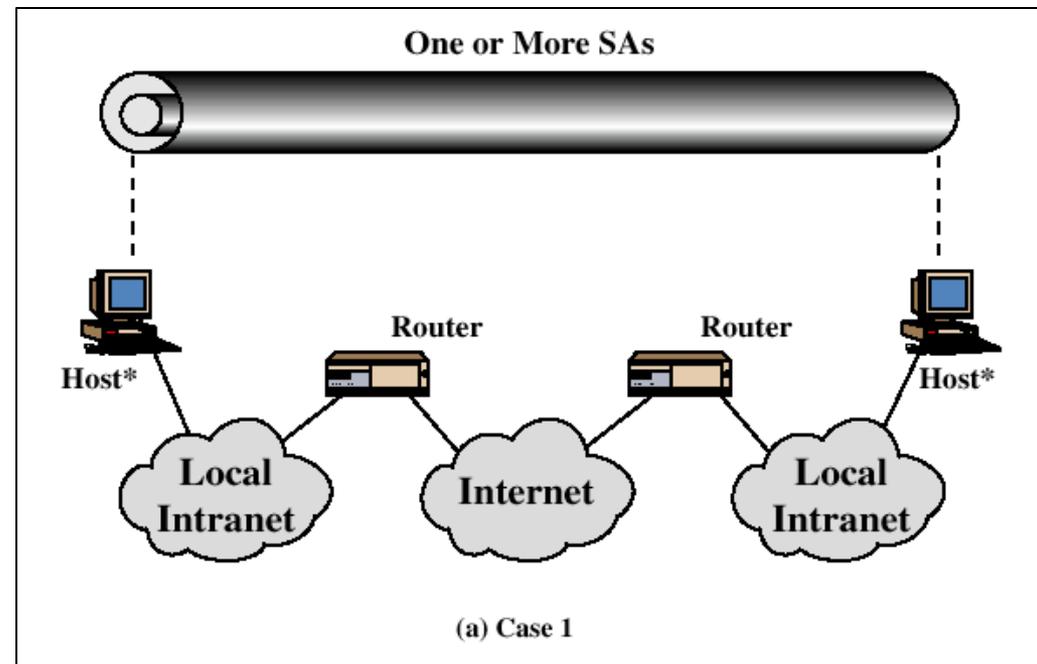
# Combining Security Associations

- SA are complementary and provide different scope in tunnel and transport mode
  - ESP+Auth (top) covers less fields than AH (bottom), as non-mutable fields of IP header are not covered
- Therefore, ESP and AH SA can be combined to provide more comprehensive encryption and authentication
- Likewise different SA can be applied at different locations, i.e. within different devices



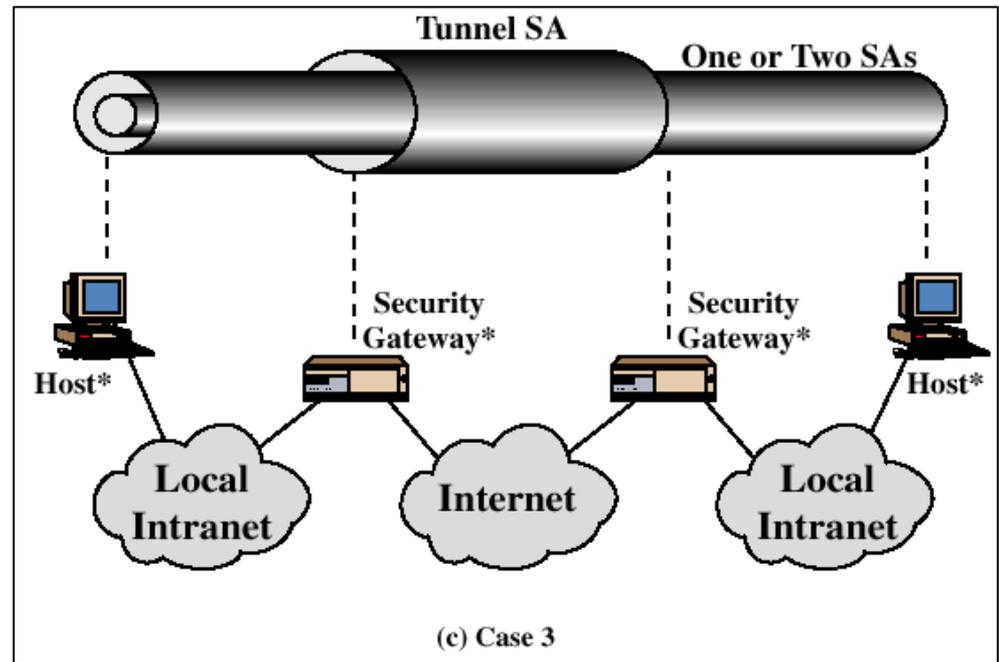
# Combining Security Associations

- AH-SA and ESP-SA bundled in transport mode, i.e. ESP-SA inside an AH-SA



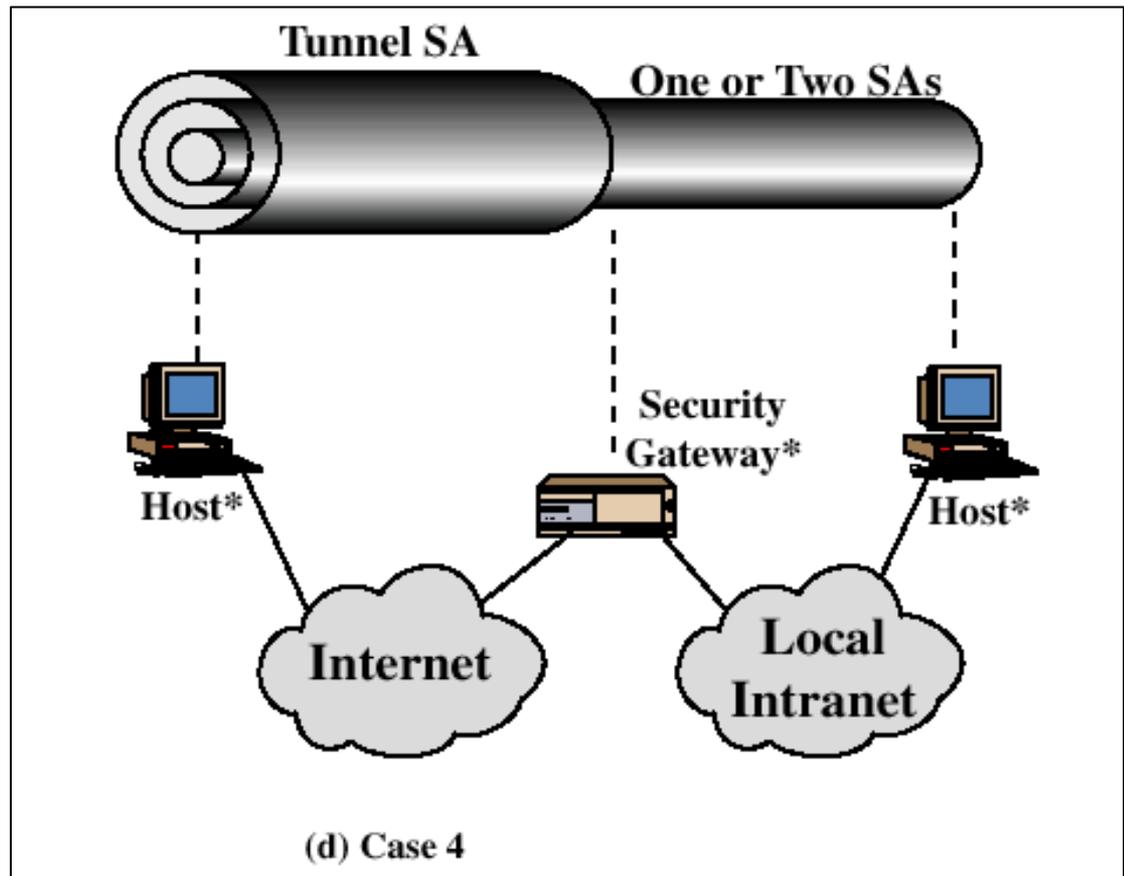
# Combining Security Associations

- VPN tunnel with added end-to-end security
- The gateway-to-gateway tunnel provides confidentiality and/or authentication
- Individual users can add any additional IPsec service to meet their needs



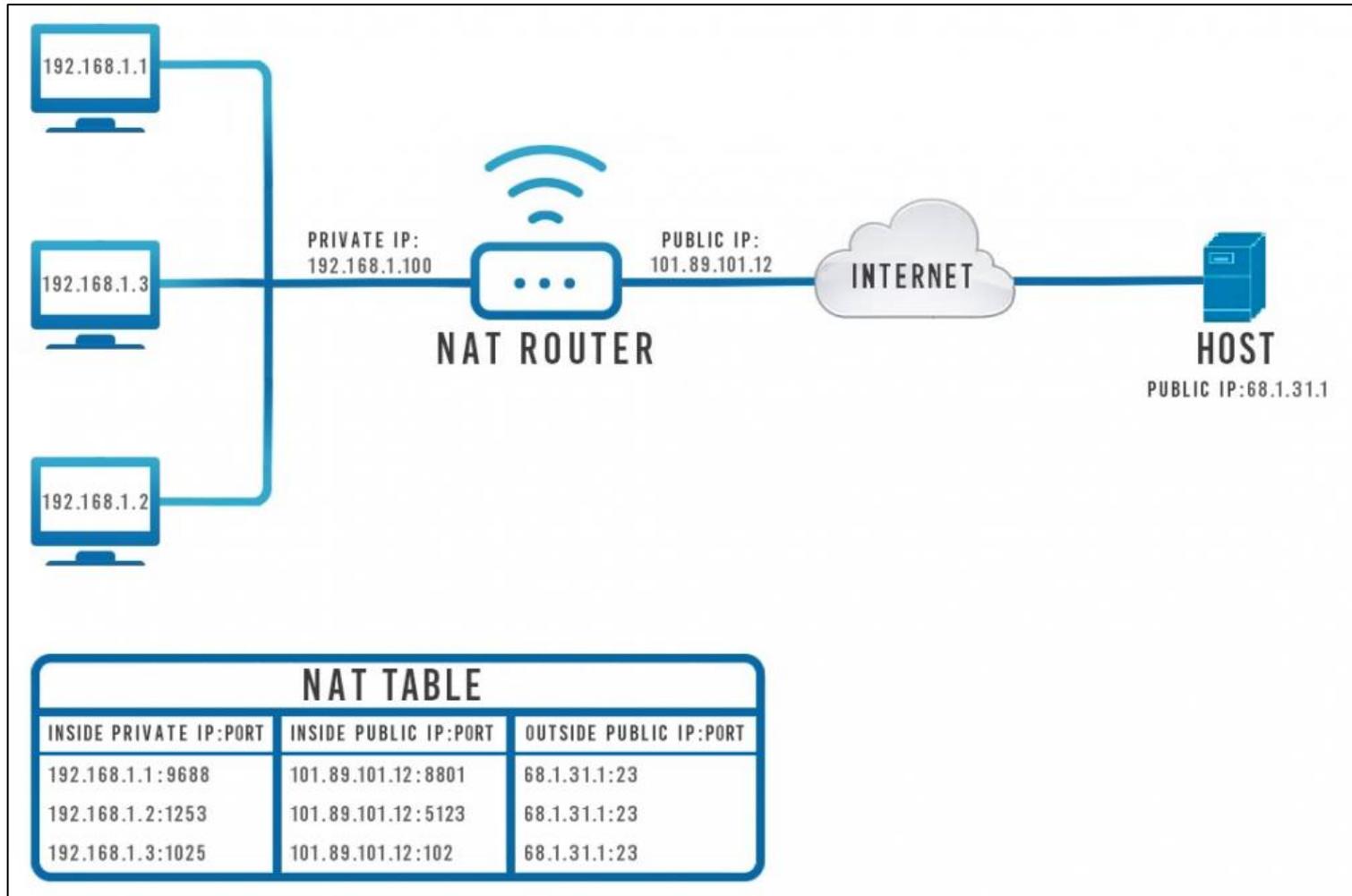
# Combining Security Associations

- Remote host connection to server using tunnelling



# Recap Network Address Translation (NAT)

31



# IPsec and NAT

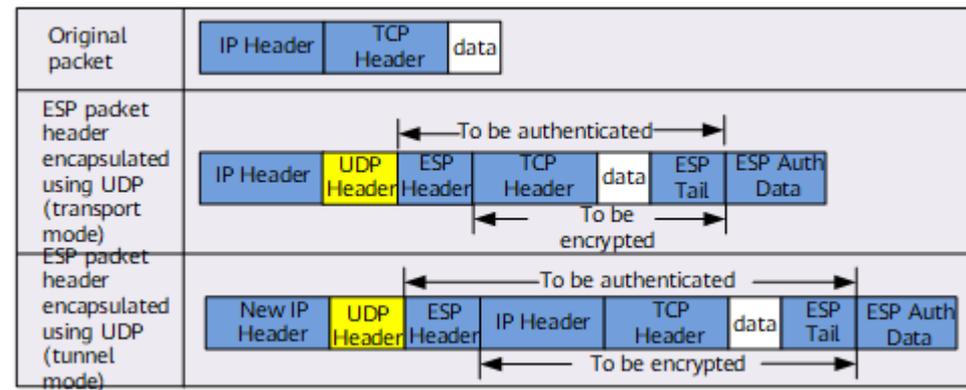
32

- With NAT a single public IP address can be shared by multiple endpoints (e.g. Wi-Fi network)
- This requires the NAT router (i.e., access point) to change the sender's
  - ▣ IP address in the IP header
  - ▣ port number in the transport layer header (UDP or TCP)
- If IPsec is installed on a client, this causes problems:
  - ▣ In AH, the datagram authentication by the receiver will fail
  - ▣ In ESP, the NAT router cannot change the encrypted port number

# IPsec and NAT Traversal

33

- ❑ NAT traversal is a technique that allows IPsec ESP to work with a NAT router
- ❑ It adds a UDP header and a special payload to the IPsec packet, which makes it look like a normal UDP packet to the NAT router
- ❑ The router can then perform the address translation on the UDP header, without affecting the IPsec payload
- ❑ The IPsec receiver endpoint can then remove the UDP header and process the IPsec packet normally



# Summary

- Network security (i.e., data encryption and / or authentication) is important for obvious reasons
- The layered structure of the TCP/IP stack allows positioning the extra security layer in different levels
- Each of these options has its advantages and disadvantages / limitations, for example with regard to
  - ▣ the portions of a packet that can be secured
  - ▣ Compatibility with network routing, NAT, etc.
- IPsec provides one possible option with encryption / authentication taking place on network layer

CT437  
COMPUTER SECURITY AND  
FORENSIC COMPUTING

Dr. Michael Schukat



# About me

2

- Professional Background:
  - ▣ M.Sc. Computer Science
  - ▣ Dr. rer. nat. (Computer Science)
  - ▣ (Senior) Lecturer in the School of Computer Science at NUI Galway
  - ▣ Senior Embedded Systems Design Engineer (Ireland)
  - ▣ Embedded Systems Design Engineer (Germany)
  - ▣ Junior Lecturer and Researcher (Germany)
- Research Interests:
  - ▣ Many, including cybersecurity
- Contact:
  - ▣ [michael.schukat@universityofgalway.ie](mailto:michael.schukat@universityofgalway.ie)
  - ▣ Office CSB3002



# My recent Publications in (AI-supported) Cybersecurity

3

- ❑ Detecting Ransomware Encryption with File Signatures and Machine Learning Models (2023)
- ❑ A Security Enhancement of the Precision Time Protocol Using a Trusted Supervisor Node (2022)
- ❑ The Application of Reinforcement Learning to the Fliplt Security Game (2022)
- ❑ Precision Time Protocol Attack Strategies and their Resistance to existing Security Extensions (2022)
- ❑ New Framework for adaptive and agile Honeypots (2020)

# Your Lab Tutor

4

- Timothy Hanley: 2<sup>nd</sup> year PhD student



**Timothy Hanley**  · 1st  
PhD Candidate at University of Galway, SFI Centre for Research Training in Digitally-Enhanced Reality (d-real)  
Kilkenny, County Kilkenny, Ireland · [Contact info](#)

 Xperi Inc.  
 University of Galway



# Cybersecurity versus Computer Security

- **Cybersecurity** is the practice of protecting systems, networks, and programs from digital attacks. These cyberattacks are usually aimed at accessing, changing, or destroying sensitive information; extorting money from users; or interrupting normal business processes

Source: Cisco

- **Computer Security** is the historically older term coined at a time when the focus was on individual stand-alone computers rather than entire systems

# What is Computer Forensics?

- Computer forensics is a branch of digital forensic science pertaining to evidence found in computers and digital storage media

The goal of computer forensics is to examine digital media in a forensically sound manner with the aim of identifying, preserving, recovering, analysing and presenting facts and opinions about the digital information

Source: [Wikipedia](#)

7

# Some Housekeeping ...

# Disclaimer

- ❑ Please adhere to the **ACM Code of Ethics and Professional Conduct!**
- ❑ See Canvas



ACM Code of Ethics and Professional Conduct

## ACM Code of Ethics and Professional Conduct

### Preamble

Computing professionals' actions change the world. To act responsibly, they should reflect upon the wider impacts of their work, consistently supporting the public good. The ACM Code of Ethics and Professional Conduct ("the Code") expresses the conscience of the profession.

The Code is designed to inspire and guide the ethical conduct of all computing professionals, including current and aspiring practitioners, instructors, students, influencers, and anyone who uses computing technology in an impactful way. Additionally, the Code serves as a basis for remediation when violations occur. The Code includes principles formulated as statements of responsibility, based on the understanding that the public good is always the primary consideration. Each principle is supplemented by guidelines, which provide explanations to assist computing professionals in understanding and applying the principle.

Section 1 outlines fundamental ethical principles that form the basis for the remainder of the Code. Section 2 addresses additional, more specific considerations of professional

# Use of Canvas

9

- Announcements
  - **Main communication mechanism, urgent messages may be circulated by email**
- Syllabus
  - Contains module outline, breakdown of marks, etc.
- Modules
  - Compulsory and optional reading materials
- Assessment
- Quizzes
  - In-class quizzes
  - End-of term student feedback questionnaire
- Discussion Forum
  - Mainly used for assignment-related questions
- Quickly attendance (used later for every lecture)
- Virtual Classroom
  - Possibly used for virtual labs

# Lecture Organisation / Breakdown of Marks

10

- 2 hours of lectures per week
  - ▣ Wednesday 10:00 – 11:00 in Tyndall Theatre
  - ▣ Wednesday 13:00 – 14:00 in ENG-2002
- 2 hours of labs per week (from week 3, tbc)
- There will be a continuous assessment (CA) component worth 30% consisting of
  - ▣ 2 assignments
  - ▣ in-class quizzes
  - ▣ lab worksheets
- The exact CA structure will be shared with you in coming days
- The summer exam has a weight of 70%
  - ▣ See Canvas for 2022/23 summer exam
- I'll be also using **Mentimeter** or Vevox for in-class feedback

# In-Class Quizzes

11

- Canvas MCQs, during the lectures
- Open book, addressing content covered during the current or previous week
  - ▣ I will provide you with details beforehand
- Typically, 5 randomised questions out of a pool of 20+ questions
- One question is presented at a time, there is no backtracking allowed
- 5 minutes duration

# Flipped Learning

12

- In some lectures we'll apply the concept of **flipped learning**:
  - ▣ You'll be notified via Canvas and study the learning materials prior to the weekly lectures
  - ▣ If you have specific questions about content, please let me know in good time, so that I can incorporate them into my lecture slots that week

# Assignment Content Overview

13

- The assignments will require you to do the following:
  1. Software development / benchmarking in C using the OpenSSL library
  2. Installation and demonstration of ethical hacking tool (i.e., Metasploit)
    - Extensive use of VM or container
- **Because of various campus restrictions you need to use your own computer / laptop for the assignments**

# Some important Ethical Hacking / Penetration Testing Tools

- Kali Linux  
An Advanced Penetration Testing Linux distribution used for Penetration Testing, Ethical Hacking and network security assessments
- Metasploit  
A software platform for developing, testing, and executing exploits
- Shodan  
Shodan is a search engine for Internet-connected devices  
<https://www.youtube.com/watch?v=Db5TPYTgy9c>

# Learning Materials and Textbooks

15

- Weekly presentations
- There's no single primary textbook, but William Stallings's
  - ▣ Cryptography and Network Security
  - ▣ Data & Computer Communicationsprovide a good overview
- I'll provide you with links to additional sources, e.g.
  - ▣ articles
  - ▣ eBooks
  - ▣ source codeas we go along

# Main Learning Outcomes

On successful completion of this module you will:

1. Have a knowledge of fundamental cybersecurity principles, including confidentiality, integrity, and availability (CIA triad), as well as an understanding of threats and attack techniques by threat actors
2. Have a solid understanding of modern cryptographic algorithms, modern cryptographic network protocols, and their applications
3. Synthesize cryptographic concepts into algorithms / frameworks to address a given cybersecurity problem
4. Be able to conduct simple information / computer system security assessments using ethical hacking / pen-testing strategies and tools
5. Proficient in the use of cryptographic libraries (i.e., OpenSSL)