INFO3616: PRINCIPLES OF SECURITY AND SECURITY ENG.

- CAESAR CIPHER
- VIGENERE CIPHER
- VIGENERE CIPHER
- SYMMETRIC CRYPTOGRAPHY
  - DESIGN GOALS
  - ONE-TIME PAD (OTP)
  - STREAM CIPHERS
  - BLOCK CIPHERS
    - GENERAL STRUCTURE
    - S-BOXES AND P-BOXES
    - FEISTEL CIPHER
    - DATA ENCRYPTION STANDARD (DES)
    - 2- and 3-DES
      - 2-DES (Double DES)
      - 3-DES (Triple DES)
    - ADVANCED ENCRYPTION STANDARD (AES)
  - BLOCK CIPHER MODES
    - ELECTRONIC CODE BOOK MODE (ECB MODE)
    - CIPHER BLOCK CHAINING (CBC MODE)
    - COUNTER MODE (CTR MODE)

WEEK 5: ASYMMETRIC CRYPTOGRAPHY

- INTRODUCTION
  - TRAPDOOR FUNCTIONS
    - DISCRETE LOGARITHM PROBLEM (DIFFIE-HELMAN)
    - RSA PROBLEM
  - COPRIME NUMBERS
- THE RSA PROBLEM

WEEK 6: HASHES AND MESSAGE AUTHENTICATION CODES

WEEK 7: AUTHENTICATION & KEY DISTRIBUTION

- THE PROBLEM OF KEY DISTRIBUTION
  - OPTIONS FOR KEY DISTRIBUTION
- SYMMETRIC KEY DISTRIBUTION
  - KEY DISTRIBUTION CENTRES
    - NEEDHAM-SCHROEDER PROTOCOL
- PUBLIC KEY DISTRIBUTION
  - PUBLIC KEY INFRASTRUCTURES (PKIs)
    - CERTIFICATES
      - OBTAINING CERTIFICATES
      - USING CERTIFICATES
      - X.509 CERTIFICATE
    - ROOT STORES & MULTIPLE CERTIFICATION AUTHORITIES
    - CERTIFICATE REVOCATION
      - CERTIFICATE REVOCATION LISTS (CRLs)
      - ONLINE CERTIFICATE STATUS PROTOCOL (OCSP)
- AUTHENTICATION
  - CORROBORATIVE EVIDENCE
    - POSSESSION
    - INHERENCE
    - KNOWLEDGE

MULTI-FACTOR AUTHENTICATION
PROCESS & INVOLVEMENT OF TWO PARTIES
AUTHENTICATION PROTOCOLS

WEEK 8: NETWORK SECURITY – PROTOCOLS

THE TCP/IP MODEL
APPLICATION LAYER
SOCKETS
TRANSPORT LAYER
UDP (USER DATAGRAM PROTOCOL)
TCP (TRANSMISSION CONTROL PROTOCOL)
NETWORK/INTERNET LAYER
LINK LAYER
PROTOCOLS BY LAYER
APPLICATION LAYER: OAUTH AND OPENID
OAUTH (OPEN AUTHORIZATION)
OPENID
TRANSPORT LAYER: TLS
INTERNET LAYER: IPSec

WEEK 9: NETWORK SECURITY – FIREWALLS

PRINCIPLES
CONFIGURATION

WEEK 10: SOFTWARE SECURITY

WEEK 11: WEB SECURITY

INTRODUCTION TO THE WEB
URL: Uniform Resource Locator
DOMAIN NAMES
HTTP AND HTTPS

WEEK 12: GUEST LECTURE

WEEK 13: REVIEW

ASSESSMENTS

ASSESSMENT	DATE OPEN	DATE CLOSE	WEIGHTING
Assignment 1	Week 3	Week 5	10%
Quiz 1	Week 6 Monday	Week 6 Sunday	5%
Assignment 2	Week 6	Week 8	10%
Assignment 3	Week 9	Week 11	10%
Quiz 2	Week 12 Monday	Week 12 Sunday	5%
Final Exam			60%

WEEK 1: INTRODUCTION

WHAT IS SECURITY ENGINEERING?

Security engineering means conceiving, designing, and implementing hardware and software that produces only the expected answers, *even when confronted with malice, error, or mischance*.

This is different from simple fault tolerance:

- ➢ Safety Engineering: Safety engineering is the engineering of systems in such a way as to minimise risk and harm when they fail. It involves hazard identification, risk analysis, and fail-safe design. For example, ensuring that a nuclear power plant shuts down safely in the case of an accident is delegated to safety engineering.
- ➢ Reliability Engineering:Reliability engineering is about minimising system failures in the first place so that a system can perform its intended function consistently over time. For example, ensuring that a server has 99.9% uptime is delegated to reliability engineering.

A secure system tolerates the activities of the opponent. Hence, it was famously analogised to "programming Satan's computer" as you must assume the presence of a hostile opponent.

SECURITY FRAMEWORK

1. **Policy**
    a. What do we want to achieve? What does it mean for this particular system to be "secure"?
    b. Examples:
        i. *Customer data must be stored only within our data centres.*
        ii. *Only authorized employees must be able to access the personal information of clients.*
    c. Note that policies do not define HOW to achieve or implement these goals, just WHAT we want to achieve.

2. **Mechanisms**
    a. What machinery do we have to implement the policies?
    b. Examples:
        i. Encryption for data confidentiality
        ii. Tamper-resistant hardware for physical security
        iii. Cryptographic hashes for data integrity

3. **Assurance**
    a. How much can you rely on each mechanism? How well do the mechanisms work together?
    b. Examples:
        i. How much time is required based on current computing capacities to break RSA encryption?
        ii. How much time is required to brute-force a 10-character password?

4. **Incentives**
    a. What motives do people have for protecting or attacking the system?
    b. Understanding incentives allows us to understand the degree of security we need.

SECURITY GOALS

Security goals are used to define the security policies.

- ➢ Confidentiality/secrecy
  - Limiting who can access data in plaintext form.
- ➢ Data/Message Integrity
  - Verifying whether a message has been modified in transit or whether data has been changed since the last honest write.
  - It is much much harder to achieve tamper-resistance for data. It is easier and more important to identify *whether* something has been tampered with.
- ➢ Authenticity
  - Verifying the origin/originator of a message or data item.
  - Effective implementation usually implies integrity.
  - In cryptographic protocols, the term also implies freshness, as we need proof that the other party is active in the process and it isn't someone else replaying recorded or intercepted messages.
  - Not the same as *authentication* which is the process of identifying a party we are communicating with.
- ➢ Authorisation
  - Verifying whether an entity is allowed to execute some action or access some data.
  - Often makes use of authentication but does not imply it. e.g. When you enter a cinema you only need to show authorisation, not who you are.
- ➢ Accountability
  - Verifying which entity is responsible for a certain action.
  - Implies authorisation and access control, may imply authentication.
  - Closely related to non-repudiation.
  - Needs some form of logging to be effectively implemented, and needs protection for the logs.
- ➢ Non-repudiation
  - The entity that has performed an action cannot successfully deny responsibility.
  - Needs secure timestamping of the action.
- ➢ Deniability
- ➢ Availability
  - A system's capability of providing its services when needed, even during an attack, error, or mishap.
  - May trade a lower level of security for higher availability when the availability of communication channels is more important than having confidentiality on them.
- ➢ Privacy
  - An entity's right to determine what information relating to themselves they want to release or hide.
  - Not the same as anonymity.

SECURITY ATTACKS

A security attack is any action that compromises the security of information owned by an organisation.

1. **Passive attacks** relate to eavesdropping on, or monitoring of, transmissions.
2. **Active attacks** involve some modification of the data stream or the creation of a false data stream.
   - 2.1. Replay
   - 2.2. Masquerade
   - 2.3. Modification of messages
   - 2.4. Denial of service

The **attack vector** is the specific method or pathway which a hacker uses to gain unauthorised access to a computer or network system. e.g.

- Phishing
- Malware
- Man-in-the-Middle or Person-in-the-Middle attacks

The **attack surface** consists of the reachable and exploitable vulnerabilities in a system.

1. Network attack surface
2. Software attack surface
3. Human attack surface

WEEK 2: PSYCHOLOGY & USABILITY

USABILITY OF SECURITY

"Build systems that accommodate for users instead of blaming them."

Usability: how easy and intuitive it is for users to interact with a software application.

- ➢ For example, the idea of "smart cards" is not very "usable":
  - The user can forget them at home
  - What happens if not all devices support them?
  - What happens if the card breaks?
  - etc.
- ➢ Compare this to passwords: users find them easier because the model matches the user's intuition (to enter a house you need a key).
- ➢ It is easy to build something secure, but it may not work for your users → not "usable".

PRINCIPLE OF PSYCHOLOGICAL ACCEPTABILITY

This refers to the principle of **psychological acceptability.**

- ➢ *A security mechanism should not make a resource more difficult to access than if the mechanism were not present.*

This is easier said than done as various users have various computing levels. Principles cannot be applied in a vacuum – must consider the end user's ability, knowledge, and mental model.

- ➢ For example: acceptability of passwords.
  - Psychological acceptability demands a low threshold for users to use the mechanism correctly - this can conflict with the goal of higher security.
  - e.g. Administrators understand the need for strong passwords while users generally do not.
  - Even though passwords are the best-known mechanism for usable security (because users understand them intuitively), users often choose weak passwords because it is difficult to memorise long, random strings.

- Another example: patching.
    - When a software patch is released to fix a security vulnerability, existing software often breaks.
    - Alert fatigue – solved by clustering updates
    - Users just ignoring the update notification – solved by forced update
- Security configurations
    - The software must be configured correctly for it to properly provide security.
    - But what if users don't actually set up or use the privacy settings?
    - Solution: make the default settings the most secure ones.
        - But this isn't always the case, as some software companies do not have this as an incentive.

CATEGORIES OF HUMAN ERRORS

- Manual skill failures
    - e.g. Typo squatters: typo in the domain name
- Following the wrong rule
    - e.g. Blindly trusting HTTPS URLs because it is "secure"
- Cognitive reasons
    - Fail to make the right decision e.g. Accidentally clicking a link in a phishing email
    - People do not understand the problem, or pretend they do, or do not want to understand the problem.

BIASES IN THE HUMAN MIND

People don't make rational decisions all the time.

- **Prospect theory and loss aversion:** People dislike losing. We feel losses more keenly than wins e.g. A $100 loss hurts more than the pleasure of receiving a $100 gain.
    - Hence, "Don't miss out, sale ends soon!" is more persuasive than "Get it now on sale!"
    - And hence, "Your PayPal account has been frozen, and you need to click here to unlock it" is a persuasive phishing email, because the end user feels like doing something about it.
- **Risk misperception:** If we like an activity, we tend to judge its benefits to be high and its risk to be low. Conversely, if we dislike the activity, we judge it as low-benefit and high-risk.
- **Anchoring effect:** We base a judgement on an initial guess and then adjust it if necessary.
    - e.g. A product for $100 seems expensive but a product for $200 that is 50% off seems reasonable
- **Availability heuristic:** Most of our decisions are based on what is the easiest to recall.
    - 'Spray and pray' traffic ticket SMS scams by referencing common and memorable traffic violations, prompting recipients to recall their own driving experiences and pay fines without verifying the ticket's authenticity.
- **Present bias:** People give stronger weight to payoffs that are closer to the present time when considering trade-offs between two future moments.
    - Causes people to decline security updates, ignore privacy policies, ignore cookie notifications etc.
- **Clustering illusion:** Our tendency to see patterns in random data.
    - One such example is in phishing attacks, where scammers send out emails that appear to come from legitimate sources, such as a bank or a social media platform, and try to convince the recipient to click on a link or provide sensitive information.
    - The scammer doesn't need to do a *perfect* job – just doing the basics, creating a fake logo and graphics, and using some basic personal information that they have obtained about the victim is enough.

- **Confirmation bias:** We tend to accept evidence that confirms our beliefs and preconceptions. Inversely, it can be very hard to convince them that something is wrong.
    - e.g. Once a major data breach happens, another set of attackers will try and leverage that by calling up victims asking them for information to “reset their account”, for example.
- **Zero-risk bias:** People focus on absolute, information-theoretical security, even if they are nearly impossible to deploy.
    - Security solutions that achieve very good, but not perfect, security are ignored, even if they reduce *overall* risk.
    - A counterexample is RSA, which is dependent on the infeasibility of finding the two divisors of a prime number. It is not *theoretically perfect* but it’s good enough.

DECISION MAKING

CLASSES OF TECHNIQUES TO INFLUENCE PEOPLE

- **Reciprocity:** People feel the need to return favors.
- **Commitment and consistency:** People can suffer cognitive dissonance if they feel they are being inconsistent.
- **Social proof/validation:** People want the approval of others. Most people comply with something when they see others do it as well.
- **Like bias:** People tend to comply with requests coming from people they like.
- **Respect to authority:** People are deferential to authority figures.
- **Scarcity:** We are afraid of missing out.

USER CONDITIONING

User conditioning refers to getting users habituated to react to certain situations in a specific way (knee-jerk reaction). This results in the user not stopping to think when a security decision has to be made but reacts automatically, for example automatically clicking “Accept All” on a cookie dialogue.

USER EDUCATION

Attempts to educate users have been met with mixed success.

- Training provides better results than just telling employees (e.g. distributing company policy).
- But training and education doesn’t work all of the time, because
    - The employees’ jobs aren’t security, and is not in their interest
    - Security language is fundamentally different from user language
    - Too hard to learn – high overhead
- User education helps a little, but by far not sufficiently.

USABILITY AND SECURITY OF PASSWORDS

- **Concern 1:** Will I be able to enter the password correctly if the password is too long?
- **Concern 2:** Will the user remember the password, or will they have to either write it down or choose one that’s easy for the attacker to guess?
- **Concern 3:** Will the user disclose their password to a third party on accident, on purpose, or as a result of deception?

WEEK 3: ACCESS CONTROL

OVERVIEW

Access control is not a *security goal*, but it is actually a policy. The way we enforce access control is through mechanisms.

- ➢ **Access control:** We want to control access to resources and assets.
    - **Authentication:** Process of acquiring evidence of the identity of a party
    - **Authorization:** The allocation of a privilege to a party.

Access control:

- Is specified in the *policy*
- Counters the *incentives* of the attacker who wants to access resource/asset
- Is implemented by one or more *mechanisms.*
    - Which gives us a certain level of assurance.

DAC AND MAC

Objects (files, processes, ...) are owned by subjects (people, groups, processes, ...).

Two ends of the spectrum:

Discretionary Access Control (DAC)

- ➢ It is the responsibility of the owner to give access control to other subjects.
- ➢ High level of flexibility
- ➢ Downside: humans can make mistakes.
    - We can forget to remove access, give the wrong access to the wrong subject, etc.
    - Not everyone is IT-savvy.

Mandatory Access Control (MAC)

- ➢ An external entity (security policy administrator) defines access.
- ➢ "Security kernel" checks via security attributes whether a subject is allowed to interact with an object.
- ➢ High level of rigour.

e.g. Google Drive

- DAC because you can share files with certain people with certain edit access.
- MAC because:
    - Every file you upload is automatically given access to the system administrator and yourself.
    - Now, every file you share has an automatic 6 month expiry date for the sharing.

OPERATING SYSTEM ACCESS CONTROL

ACCESS CONTROL LISTS (ACLs)

Each object has an ACL which contains a list of subjects mapped to their allowed operations.

- Found in all OS
- Also found on social networking sites (e.g. who can view your post)

User	Accounting Data
Sam	rw
Alice	rw
Bob	r

Access control list (ACL)

UNIX OPERATING SYSTEM SECURITY

USERS

Each user has a User ID (UID) and a Group ID (GID).

- A user can belong to the `sudo` group (superuser)
- To see the list of users on the system: `/etc/passwd`
  - One line for each user
  - `sam:x:1000:1000:Sam Jones,,,:/home/sam:/usr/bin/zsh`
  - `username:is_password_set:uid:gid:user_info:home_directory:shell`
- To see the list of groups: `/etc/group`
  - One line for each group
  - `sudo:x:27:sam,pnappa`
  - `group_name:group_password:gid:user_list`

FILE PERMISSIONS

- Owner ID: the user that owns the file
- Group ID: the group that 'relates to' the file (default is just the owner themselves)
- Owner R/W/X: what the owner can do
- Group R/W/X: what the group can do
- Other R/W/X: what everyone else can do
- e.g. `-rw-r--r-- 1 sam sam 24K Jun 12 10:20 email.png`
  - The first bit tells you what sort of file you are dealing with (d: directory, l: symbolic link)
  - The second, third, and fourth bits are for the owner (RWX)
  - The fifth, sixth, and seventh bits are for the group
  - The eighth, ninth, and tenth bits are for everyone else
  - The owner, group and others can all read. Only the owner (sam) can write.

Files also have some special bits in UNIX.

- **Set-user-ID bit (SUID)**
  - File can be executed as the owner
  - Useful for running specific programs with additional permissions without being a privileged user
  - e.g. `mount`
  - The fourth bit will be `S` if this is the case
- **Set-group-ID bit (SGID)**
  - On file: like SUID, but as the group
  - On directories: new files/directories are created with group of parent directory
- **Sticky bit**
  - Only the owner can modify the file
  - Used in `/tmp`

Processes inherit permissions from the running user (UID and GID)

- **Real**: UID and GID
  - The user/group that called the processes
- **Effective:** UID (eUID) and GID (eGID)
  - What the process is running as (SUID/SGID affects this)
- **Saved:** UID and GID
  - Used so that the kernel knows what to switch back to

Root is the all-powerful user.

- UID and GID of 0
- The `sudo` command lets you run a command as root
- You can become root using `sudo_su`

MIDDLEWARE ACCESS CONTROL

Middleware is a resource that is shared by multiple applications. For example:

- One database, multiple applications connect to it
- Siri – multiple applications can access voice assistant

There needs to be some definition of what each application can access in the middleware.

DATABASE ACCESS CONTROLS

DBMSes with multi-user, multi-process support have their own access control, usually *role-based access control:*

- ➢ Database administrator
  - Assigns roles to all users
- ➢ Database managers
  - Can do “DROP TABLE” etc.
- ➢ Database users
  - Can only read and query tables

But, this can lead to an overkill in complexity (super fine-controlled access control).

Keep in mind that the actual data is still on the disk or in RAM so you still need to protect the files at the operating system level.

BROWSER ACCESS CONTROLS

One browser allows you to access multiple web applications e.g. one tab you have your bank account open, and one you have malware – can the malware access your bank?

No, because each tab is run in a **sandbox** to prevent it from altering the host system.

- Main access control is the “same-origin policy” – JavaScript is only allowed to run if it comes from the same source??

OTHER ACCESS CONTROL

SANDBOXING

You run each application in a virtualised environment so that it cannot interfere with the surrounding environment.

- Java applet runs in the browser with its own …
- Google Chrome: tabs run in their own process

Problem: the sandbox is hard to secure.

Virtualisation

- Use case: Virtual Machines (VMs) emulate a computer system
  - **Old** technology: around the 1960s!
  - Allows to run software even if the real environment it was written for is not available
  - OS running in VM sees only its 'own' hardware

- Varying degrees of virtualisation:
  - Full emulation of hardware (full virtualisation even across hardware architecture)
  - Para-virtualisation (guest OS must support being virtualised)
  - Containers and jails are actually a form of light-weight virtualisation

Virtualisation

- Hypervisor manages and runs VMs: hardware support in ring -1
  - In full and para-virtualisation
  - Examples: Virtualbox, VMWare, Xen, QEMU, …

- Not designed for access control, but the typically high effort required to break out
  - Raises the bar for attackers
  - Note that execution in a VM is detectable: the VM does not perfectly imitate a real environment
  - Malware can detect it is inside one and act differently
  - Vulnerabilities exist in the hypervisors and can be exploited: always keep up-to-date, and do not use as a single defence

Example: Stuxnet

Example: Docker

- Initial use case: continuous integration and delivery (CI/CD)
- Abstraction over containers
  - Supports several OSes via their native jail implementations
  - Builds images out of apps and supporting libraries (!)
  - Fast to spin up and delete, easy on resources
  - Management systems (e.g. Kubernetes) available
- Sandboxes the app in the container
  - App cannot communicate with app in another container
  - Docker daemon is broker but runs as root
  - Means any user who can start/stop containers can interact with the daemon: attack surface
- While not designed for access control, containers provide an extra degree of separation and raise the bar for attacks

HARDWARE ACCESS CONTROL

MEMORY PROTECTION

Each program should get some memory segment to run in, and other programs should not interfere with that memory space.

- ➢ CPUs support the OS in keeping processes' memory allocations apart
  - Memory is addressed by two registers – a segment register that points to a segment of memory and an address register that points to a location within that segment
  - Segmentation fault: SIGSEGV is raised if a process tries to access memory outside its address space

PRIVILEGED EXECUTION

- ➢ CPUs have layered modes (rings) to support OS
- ➢ Rings separated with different privilege classes

Modern Intel CPUs have nine rings: 0-3 for normal code, under which is a further set of rings 0-3 Virtual Machine Manager (VMM) root made for hypervisor, and System Management Mode (SMM) for BIOS

Ring 0: OS
Ring 1-2: Device drivers
Ring 3: Runs the program, except for system calls.

Kernel and hardware

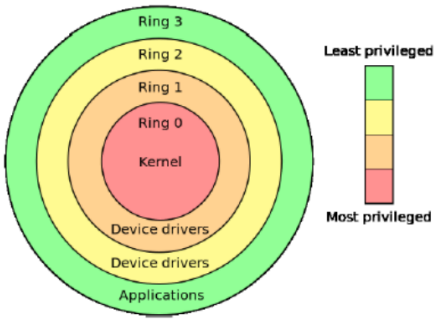


Figure: x86 rings (Wikipedia)

- Code runs in one of the available CPU modes, called *rings*.
- CPUs enforce access control from higher privileged rings to lower
- Process running in a higher ring cannot directly interact with process in lower

- **Principle of least privilege:** all code should run only with minimal required privilege, never more.
- Number of ways to communicate with the kernel should be limited to the required minimum, well-controlled

SECURITY POLICY MODELS

MULTILEVEL SECURITY (MLS) POLICY

MLS is a foundational form of access control

- Idea: information must be handled at different levels of classification
  - Top Secret, Secret, Confidential, etc.
- Different people can access different levels of information

How can we build a system to implement this?

SECURITY POLICY MODELS

When analyzing a system's security, one often follows this order:

1. Analyze incentives and create a *threat model* (a model that describes the attacker's motivations and capabilities)
2. Analyze security policy; the policy describes the goals to be achieved (relatively verbose)
3. Analyze security mechanisms to implement the security policy

A Security Policy Model is a highly precise and succinct statement about the protection properties of the system

- Can be highly formal
- Drives engineering

BELL-LAPADULA MODEL

Addresses confidentiality.

Two main MAC properties:

1. **No read up** aka *Simple Security*: no principal may read data at a higher security level
2. **No write down** aka $*$*-property*: no principal may write data to a lower security level

a. Example: user at low security level writes a program to exfiltrate data, Waits for admin at high security level to accidentally execute it
b. Or could be does accidentally

One DAC property: Bell-LaPadula also has one **discretionary form of access control**, in case someone needs access to just a few files from a higher security level. Here, an external entity defines the access of subjects to objects in a matrix.

Innovations

- ➢ Lends itself well to formal analysis
- ➢ Combines DAC and MAC

Criticisms

- ➢ One Trusted Principal – how to secure that?
- ➢ Does not consider covert channels
  - e.g. screenshots

BIBA MODEL

Addresses integrity (prevents malicious change).

A reverse of Bell-LaPadula:

- ➢ **Read up:** principal may only read data at own level or higher
- ➢ **Write down:** principal may only write data at own level or lower
- ➢ **Invocation property:** principal cannot obtain any access to objects at higher level (only own and lower)

- e.g. Aircraft in-flight entertainment system can read data from avionics (e.g., airspeed), but cannot affect it
- e.g. Odometer can display mileage, but not change it

Pure Bell-LaPadula or Biba implementations are rare.

WEEK 4: SYMMETRIC CRYPTOGRAPHY

Content
1. **Motivation:** Why we need cryptography at all a. The goal of cryptography b. Kerckhoff's principle sets the philosophy: *security comes from the key, not secrecy of the algorithm.* 2. **Threats:** What we're defending against a. Attacks (ciphertext-only, known plaintext, chosen plaintext, chosen ciphertext). 3. **Principles** a. Core design goals: confusion + diffusion. 4. **Algorithms** a. One-time pad (conceptual baseline for perfect secrecy). b. Stream ciphers (practical OTP approximation). c. Block ciphers (DES, AES) → Feistel structure, S-boxes. d. Block cipher modes (ECB, CBC, CTR, etc.). e. Pseudorandom number generators (essential for key streams).

INTRO TO CRYPTOGRAPHY

Textbook definition: The science of securely transmitting data, spatially and temporally

- Spatially: over a physical distance
- Temporally: data stays secure a long time into the future

In this unit, we are mostly concerned with cryptography's application to:

- ➢ Confidentiality
  - Encryption (symmetric and asymmetric)
- ➢ Integrity
  - Symmetric and asymmetric ways to achieve this
  - Message Authentication Codes and Signatures
  - Hash functions
- ➢ Authentication
  - Protocols
  -

We define the components of a **cryptosystem**

- **Plaintext**: non-encrypted, unprotected data
  - Symbols from alphabet $R = \{r_1, r_2, \dots, r_n\}$
- **Encryption**: encode plaintext using **cipher** function and **key** k_e
  - $Enc_{k_e}(p) = c$, with Enc being a cipher function
- **Ciphertext**: the outcome of encryption c
  - Symbols from alphabet $S = \{s_1, s_2, \dots, s_m\}$
- **Decryption**: decode ciphertext using **decipher** function and **key** k_d
  - $Dec_{k_d}(c) = p$, with Dec being the inverse cipher function
  - Yields plaintext again

- **Symmetric cryptography**: key for encryption and decryption is the same ($k_e = k_d$), or at least $k_d \leftarrow k_e$ (can derive k_d from k_e)
  - Also known as **shared-key cryptography**
  - Based on algorithms to shuffle symbols around and map them to others

- **Asymmetric cryptography:** keys for encryption and decryption are different ($k_e \neq k_d$), and **infeasible** to compute k_d from k_e
  - Also known as **public key cryptography**
  - Based on the hardness of mathematical problems

KERKHOFF'S PRINCIPLE

The security of a given cryptographic mechanism must depend only on the security of the cryptographic keys used, but never on the mechanism or anything else, except the keys, being a secret

- ➢ You should be able to disclose the mechanisms for the cryptography publicly without worry that anyone will overcome it unless they have access to the keys themselves.
- ➢ This allows the weaknesses of a mechanism to be found faster (open-source argument).

ATTACKS ON CRYPTOSYSTEMS

A cryptographic system is compromised:

- if an adversary can obtain plaintexts for certain ciphertexts, or
- adversary can obtain decryption key k_d when given k_e

Cryptanalysis: decrypting without knowledge of the key

Several forms of attacks:

- ➢ Ciphertext-only: attacker only sees the ciphertexts. This is how you crack the Caesar cipher, Enigma in WWII, etc.
- ➢ Known plaintext: attackers may see some pairs of plaintexts and corresponding ciphertexts, but do not have control over which pairs they can get.
- ➢ Chosen plaintext: attacker may choose a plaintext, query the crypto system, and see the corresponding ciphertext (standard model for asymmetric cryptography)
- ➢ Chosen ciphertext: attacker may choose ciphertext, query the crypto system, and see the decrypted plaintext (oracle attacks)

CLASSICAL CIPHERS

CAESAR CIPHER

The simplest substitution cipher

The Caesar cipher involves replacing each letter of the alphabet with the letter standing three places further down the alphabet.

```
plain:  meet me after the toga party
cipher: PHHW PH DIWHU WKH WRJD SDUWB
```

If we assign a = 0, and z = 25, then for each plaintext letter *p*, the ciphertext letter *C* can be expressed as:

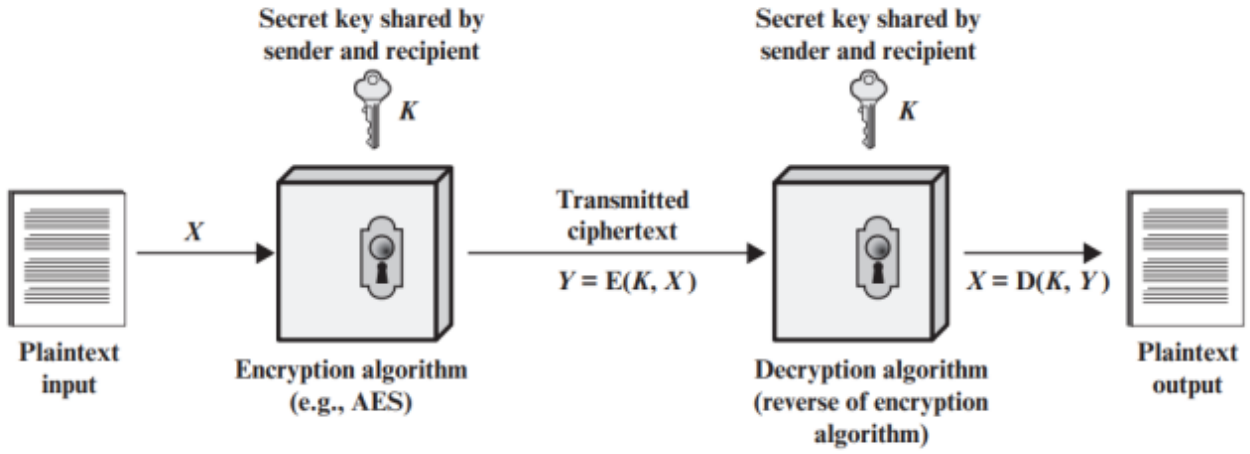
$$C = E(3, p) = (p + 3) \bmod 26$$

VIGENERE CIPHER

The best known polyalphabetic cipher

In essence, each plain-text character is encrypted with a different Caesar cipher, depending on the corresponding key character.

SYMMETRIC CRYPTOGRAPHY



DESIGN GOALS

- ➢ Diffusion: blur the statistically significant characteristics in plaintext
  - Every ciphertext symbol must depend on many plaintext symbols
  - On average, a bit flip in plaintext should change half the ciphertext bits
- ➢ Confusion: blur the dependency between ciphertext and key
  - Each ciphertext symbol must depend on several symbols in the key

ONE-TIME PAD (OTP)

OTP is the *perfect cipher,* which means that given a ciphertext, any possible plaintext is equally likely along with the correct one. This is the **theoretical gold standard.**

- **Encryption:** `ciphertext = plaintext XOR key.`
- **Decryption:** `plaintext = ciphertext XOR key`

- ➢ OTP *o* is a random sequence of bits with |o| >= |p| i.e. at least as long as the plaintext
- ➢ Ciphertext: c = o XOR p
- ➢ Reproduce the plaintext by applying the XOR function again
- ➢ OTP is perfectly secure if and only if:
  - o is truly random
  - |o| >= p
  - Never used twice

Why is OTP secure? Because there is **no relationship between the plaintext and the ciphertext.**

e.g. Alice wants to send the plaintext “hello” to Bob.

- She chooses a one-time pad using a RNG of length five and converts it into binary:
  - e.g. 73 92 40 122 121 → 01001001 01011100 00101000 01111010 01111001
- She then converts the plaintext into ASCII and then into binary:
  - 01101000 01100101 01101100 01101100 01101111
- She computes the XOR of these two strings, *c*
  - 10000100 11100101 00010000 01011000 010110
- She sends this ciphertext to Bob.

- Any eavesdropper cannot decrypt this ciphertext because each possibility is equally possible for each bit.
- Bob, knowing the one-time pad, can compute the XOR of the ciphertext to reveal the original plaintext.

Why is this not used?

➢ Conditions on *o* are usually too hard to achieve, and hence OTP is impractical
  - You need to have an infinitely long, random sequence of numbers for *o*
  - And then you need to share this key securely
  - The problem is with the practicality of key distribution. If Alice and Bob already share a long, secret, random key, they could just use that to communicate securely instead of exchanging OTP-secured messages. That's why OTP is mostly of theoretical and historical interest (though some intelligence agencies did use it with physical codebooks).

STREAM CIPHERS

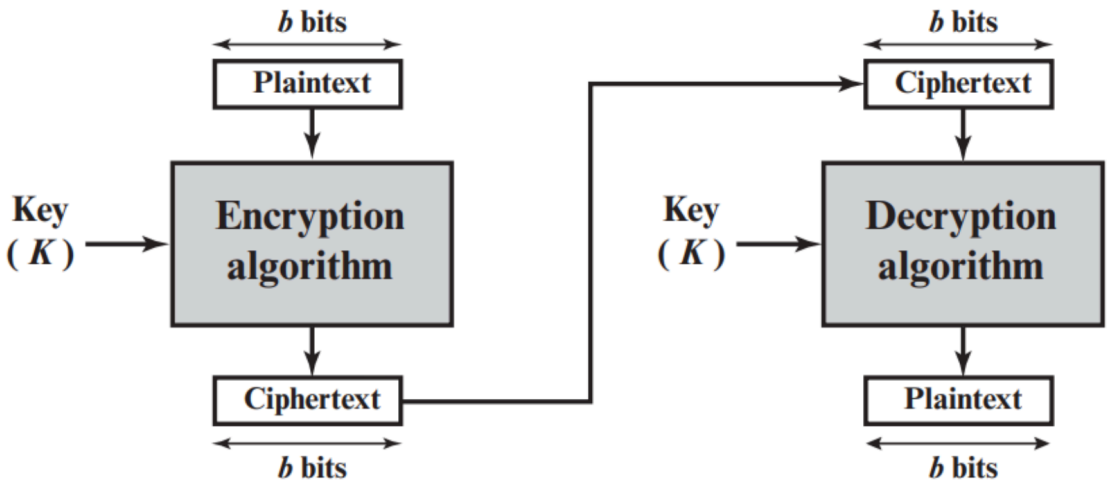
These are an approximation of OTP with pseudorandom key streams.

- Netflix

E.g. salsa20

BLOCK CIPHERS

GENERAL STRUCTURE



- Substitution implemented via Substitution box (S-boxes)
- Permutation implemented via Permutation box (P-boxes)
- Rounds arranged in so-called 'networks'

S-BOXES AND P-BOXES

A substitution box (S-box) is essentially a look-up table which maps a block of bits to an output block of bits.

- It can be shown that S-boxes achieve the property of *confusion*, where changing one bit in the input flips (on average) half of all output bits

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

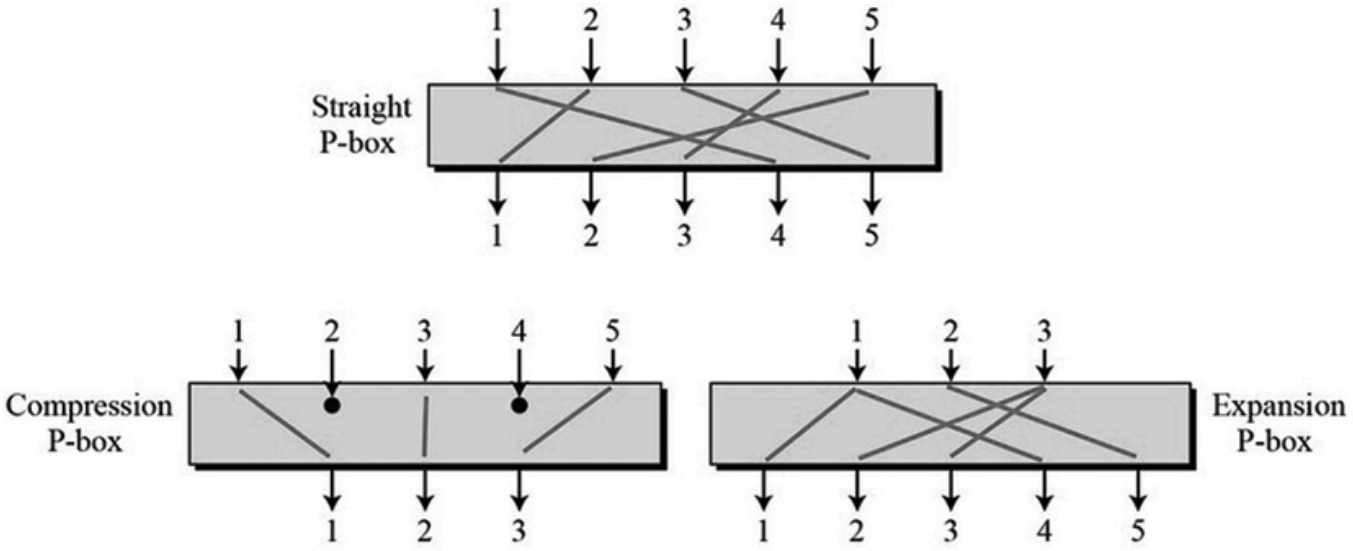
Use like this:

- input string = $(b_0, b_1, b_2, b_3, b_4, b_5)$
- b_0 and b_5 give row; (b_1, b_2, b_3, b_4) give column
- Interpret the number in that cell as binary

Example:

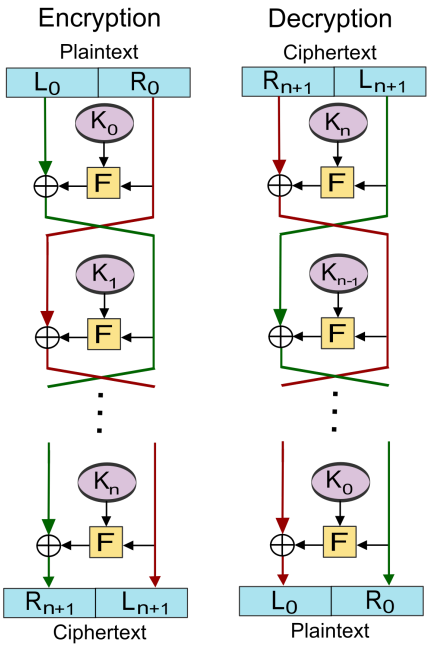
- Input = $(1, 1, 0, 1, 1, 1) \rightarrow (110111)_2$
- Row: $(11)_2 = (3)_{10}$; Column: $(1011)_2 = (11)_{10}$
- Output: $(14)_{10} = (1110)_2 \rightarrow \mathbf{1110}$

A P-box is a permutation of all the bits: it takes the outputs of all the S-boxes of one round, permutes the bits, and feeds them into the S-boxes of the next round. A good P-box has the property that the output bits of any S-box are distributed to as many S-box inputs as possible → *diffusion*.



FEISTEL CIPHER

https://www.youtube.com/watch?v=FGhj3CGxl8I&t=29s



1. Take the right half of the plaintext bits R and input it into some hash function F which has key K_0.
2. XOR the result F(R, K_0) with L to produce L_0⊕F(R, K_0). This becomes the new RHS, and R becomes the new LHS.
3. Repeat. The next RHS is R ⊕ F(L⊕F(R, K_0), K_1), and the next LHS is L⊕F(R, K_0)
4. Repeat for as many blocks as required.
5. The final ciphertext is the result of swapping the LHS and RHS one last time.
6. For two blocks, this means LHS = R ⊕ F(L⊕F(R, K_0), K_1), RHS = L⊕F(R, K_0)

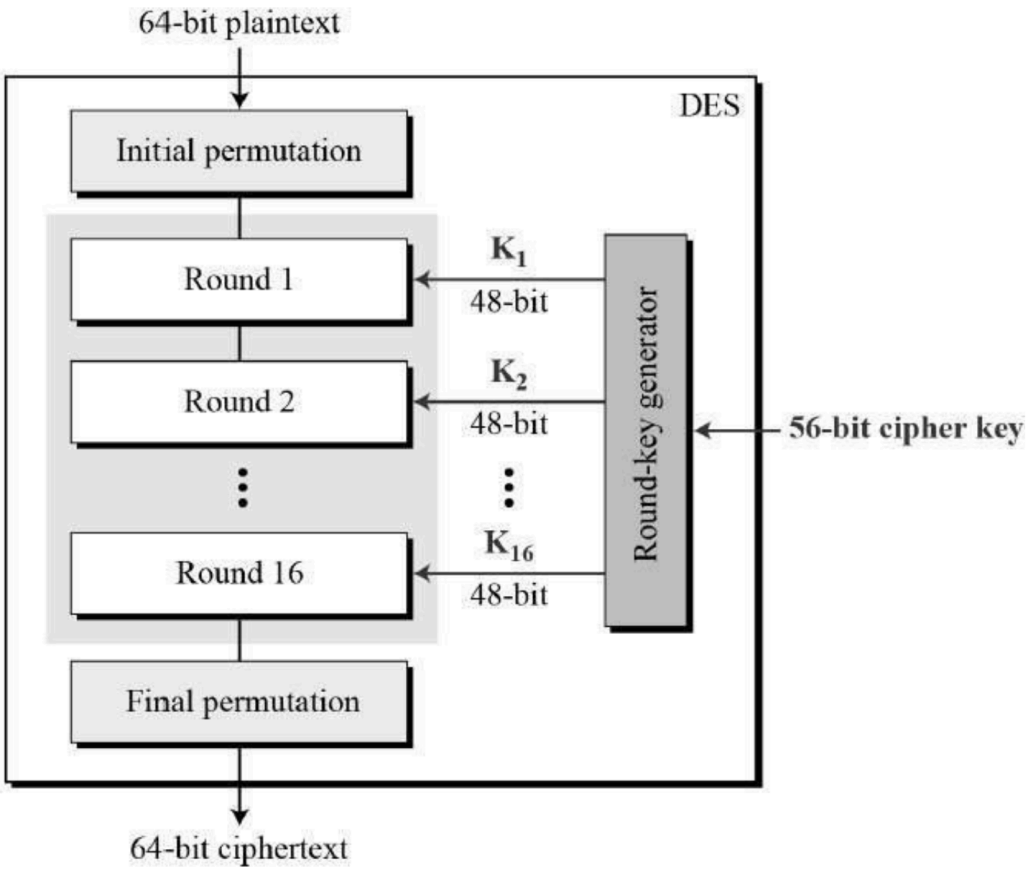
The beauty of the Feistel Cipher is that to decrypt the message, you just do the exact same thing in reverse. Imagine the network mirrored vertically.

1. Take the right half of the plaintext bits and input it into some hash function F which has key K_1 = F(L⊕F(R, K_0), K_1)
2. XOR this result with LHS: R ⊕ F(L⊕F(R, K_0), K_1) ⊕ F(L⊕F(R, K_0), K_1) = R
3. R becomes the new RHS and L⊕F(R, K_0), K_1 becomes the new LHS.
4. Then, the next RHS is L and the next LHS is R.
5. Then you swap the two halves, returning the original text!

DATA ENCRYPTION STANDARD (DES)

DES uses Feistel Cipher

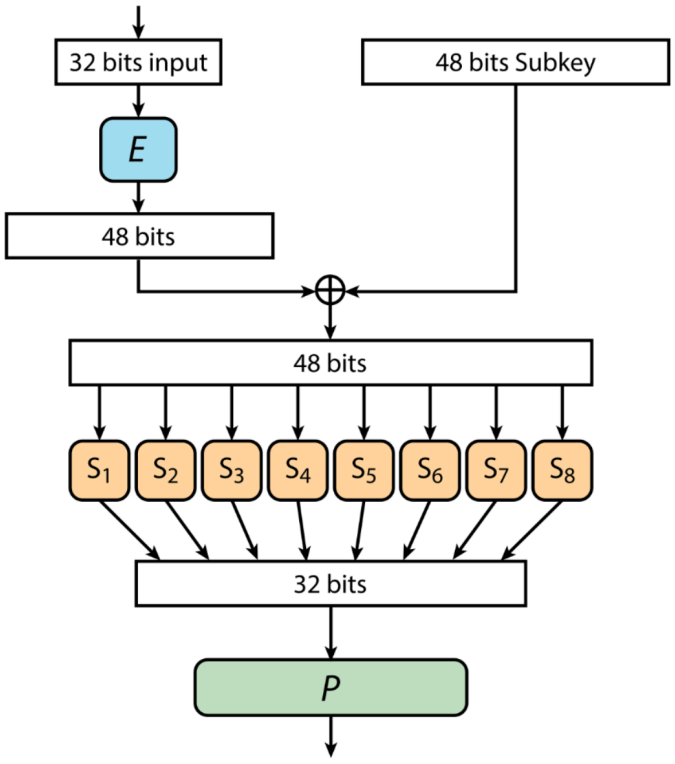
Now we have all the pieces needed to understand DES: the Feistel cipher and P- and S-boxes.



This is DES. It takes in fixed inputs of 64 bits, as well as a main cipher key of 56 bits.

1. Each “round” here is one round of the Feistel cipher
2. The key for each round is a 48-bit “round” key (or subkey) which is created from the main cipher key

The question that we have not answered is: what function do we use for *F*? In DES, that function is defined using S- and P- boxes as:



The reason we do not use DES anymore is because the key length (56 bits) was too short.

- Supercomputers can decrypt this using brute force in 1 hour
- Compare this to AES which has a key length of 128, which can only be brute forced in 10^17 years.

2- and 3-DES

They temporarily created 2- and 3-DES in between the time they discovered DES was not secure and the time AES was standardised.

2-DES (Double DES)

Instead of encrypting once with DES, you encrypt twice with two different keys:

$$C = E_{K_2}(E_{K_1}(P))$$

where P = plaintext and C = ciphertext.

You would think the key size is now 56 + 56 = 112 bits, HOWEVER due to meet-in-the-middle attacks, this is not actually true.

1. Suppose an attacker knows a plaintext–ciphertext pair (P, C).
2. They compute:
   - For all K_1, encrypt P once: $X = E_{K_1}(P)$. Store results in a table. (about 2^{56} operations).
   - For all K_2, decrypt C once: $Y = D_{K_2}(C)$. Compare with the table. (another 2^{56} operations).
3. When $X = Y$, they've likely found the correct K_1, K_2.

This attack requires **about** $2^{56} + 2^{56} = 2^{57}$ **operations**, not 2^{112}.

👉 That means **2DES only offers ~57-bit security** — barely better than DES itself.

3-DES (Triple DES)

There are two main variants of 3-DES: one with 3 keys and one with 2 keys. However, the general structure is the same:

ENCRYPTION → DECRYPTION → ENCRYPTION

The reason why they chose EDE is because if all three keys were the same, this would just reduce down to normal DES. This made it *backward-compatible* with hardware and systems that only supported DES.

1. 3 keys

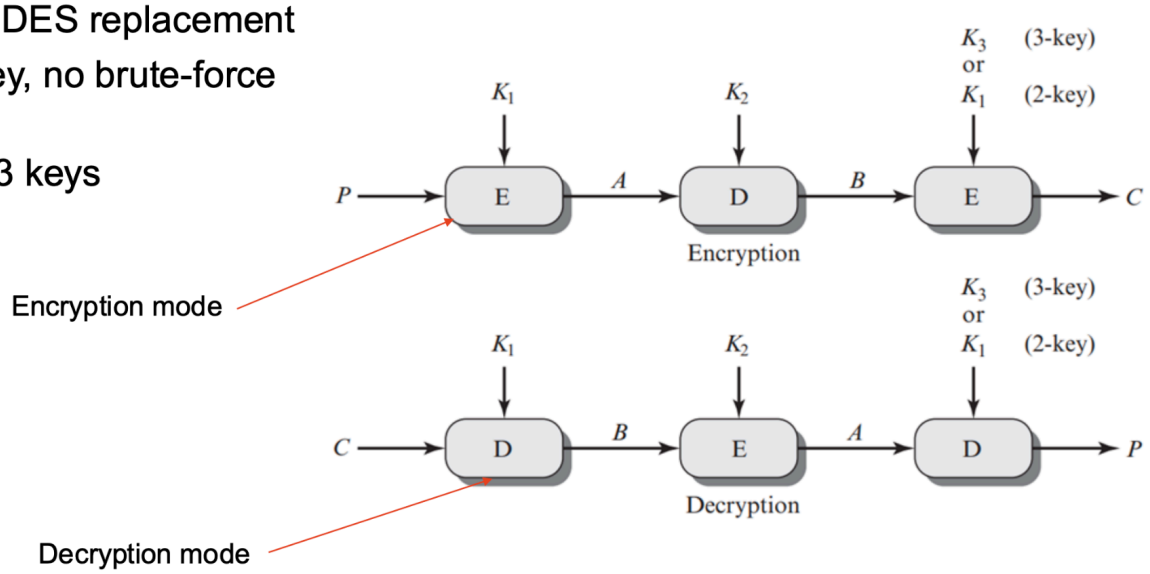
$$C = E_{K_3}(D_{K_2}(E_{K_1}(P)))$$

2. 2 keys

$$C = E_{K_1}(D_{K_2}(E_{K_1}(P)))$$

2-key 3DES was cheaper to implement as you only had to manage two keys, and it still gave sufficient security for the time-being.

- Purpose: DES replacement
- 168-bit key, no brute-force attack
- Has 2 or 3 keys



Ultimately, 3DES was replaced with AES because 3DES is slow (3 x DES operations per block) and AES is more secure with longer key lengths.

ADVANCED ENCRYPTION STANDARD (AES)

*AES uses Substitution-Permutation Networks (SPNs) with the **Rijndael cipher***

- Standard AES takes in a block size of 128 bits = 16 bytes.
  - AES can also take in 192 bits (24 bytes) or 256 bits (32 bytes).
- The 16 bytes are arranged in a 4x4 grid
- Uses S-Boxes and P-boxes

It performs arithmetic in the finite field (Galois field) of *GF(2^8)* elements.

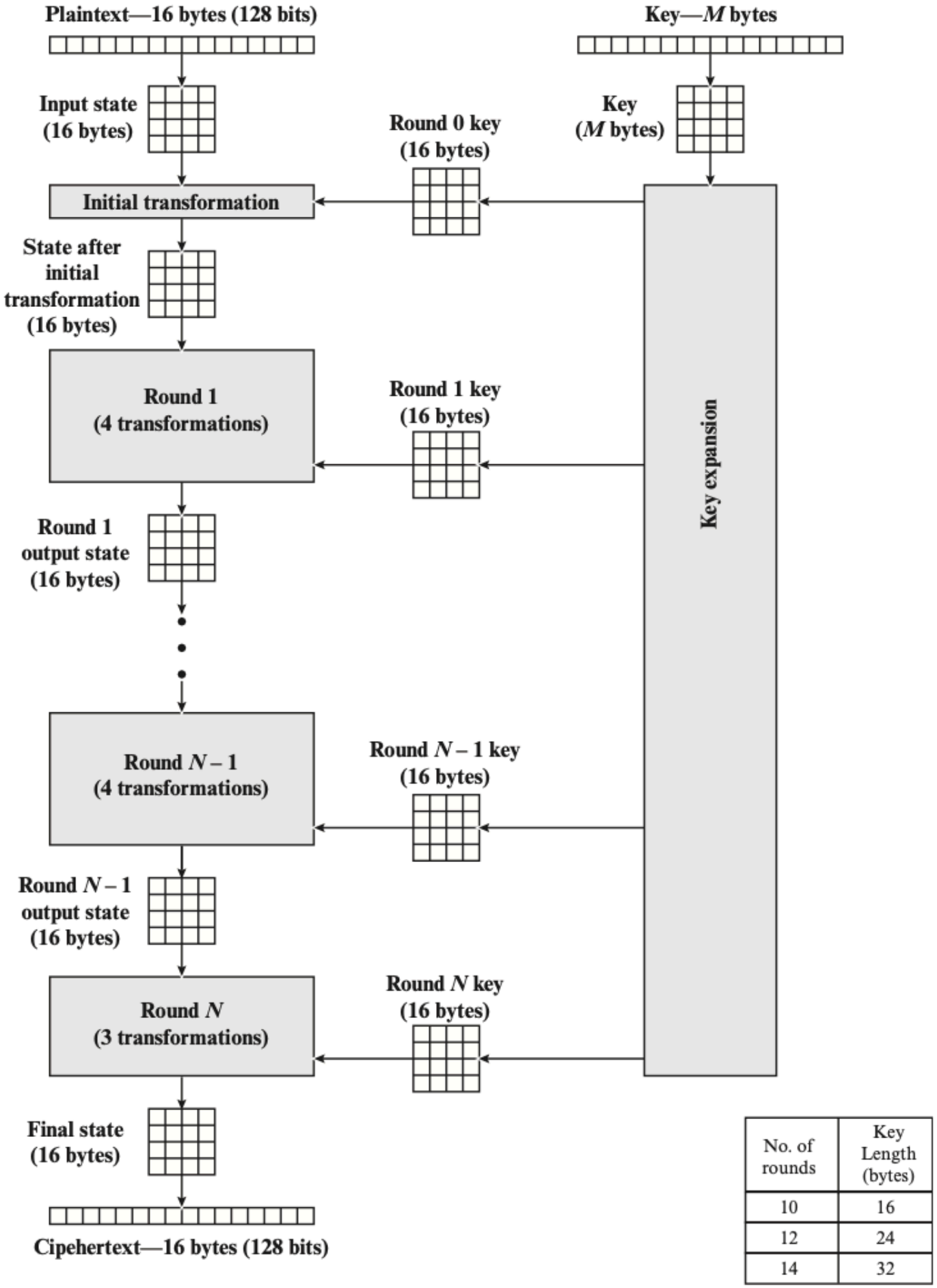
➢ A finite field has a *finite* number of elements. In this case, the finite field contains all the elements from 0000 0000 to 1111 1111 = 2^8 elements = total number of possibilities in a byte.

➢ The reason this is important is because it means that whatever operations you perform in the field, you will always output something *also in the field.* You will never produce something outside of the 128 bits, which is important in the AES algorithm.

- Multiplication of two bytes is calculated as the multiplication of two bytes in a finite field
- Addition of two bytes is defined as the bitwise XOR operation

You begin with a key of length 128.

- This key is expanded into an array of 44 32-bit words using the key expansion algorithm.



No. of rounds	Key Length (bytes)
10	16
12	24
14	32

Figure 6.1 AES Encryption Process

There are 4 main stages in the AES algorithm:

1. **Substitute bytes:** Uses an S-box to perform a byte-by-byte substitution of the block
2. **ShiftRows:** A simple permutation of each row
3. **MixColumns:** A substitution that makes use of multiplication arithmetic over GF(2^8)
4. **AddRoundKey:** A bitwise XOR of the current block with a portion of the expanded key

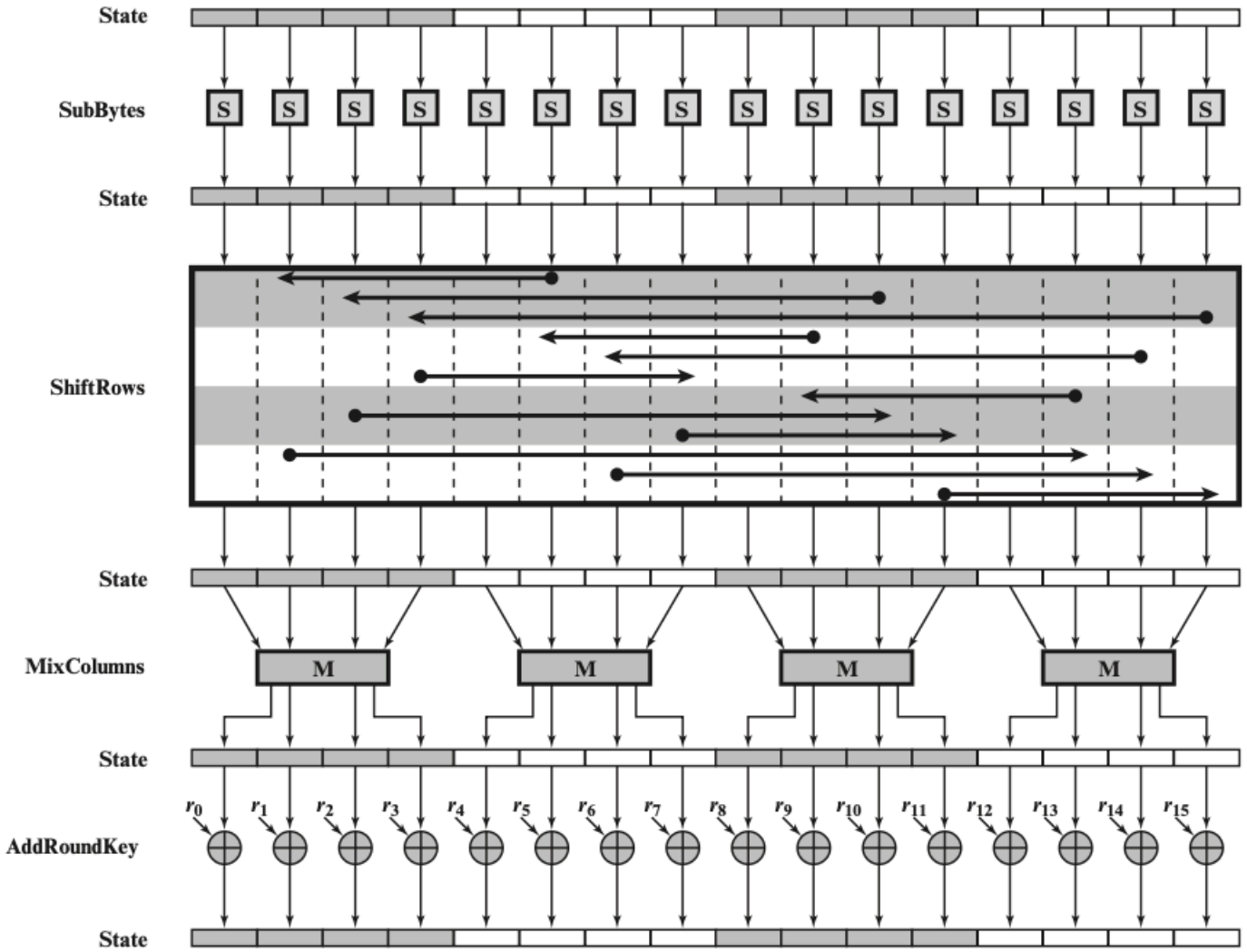


Figure 6.4 AES Encryption Round

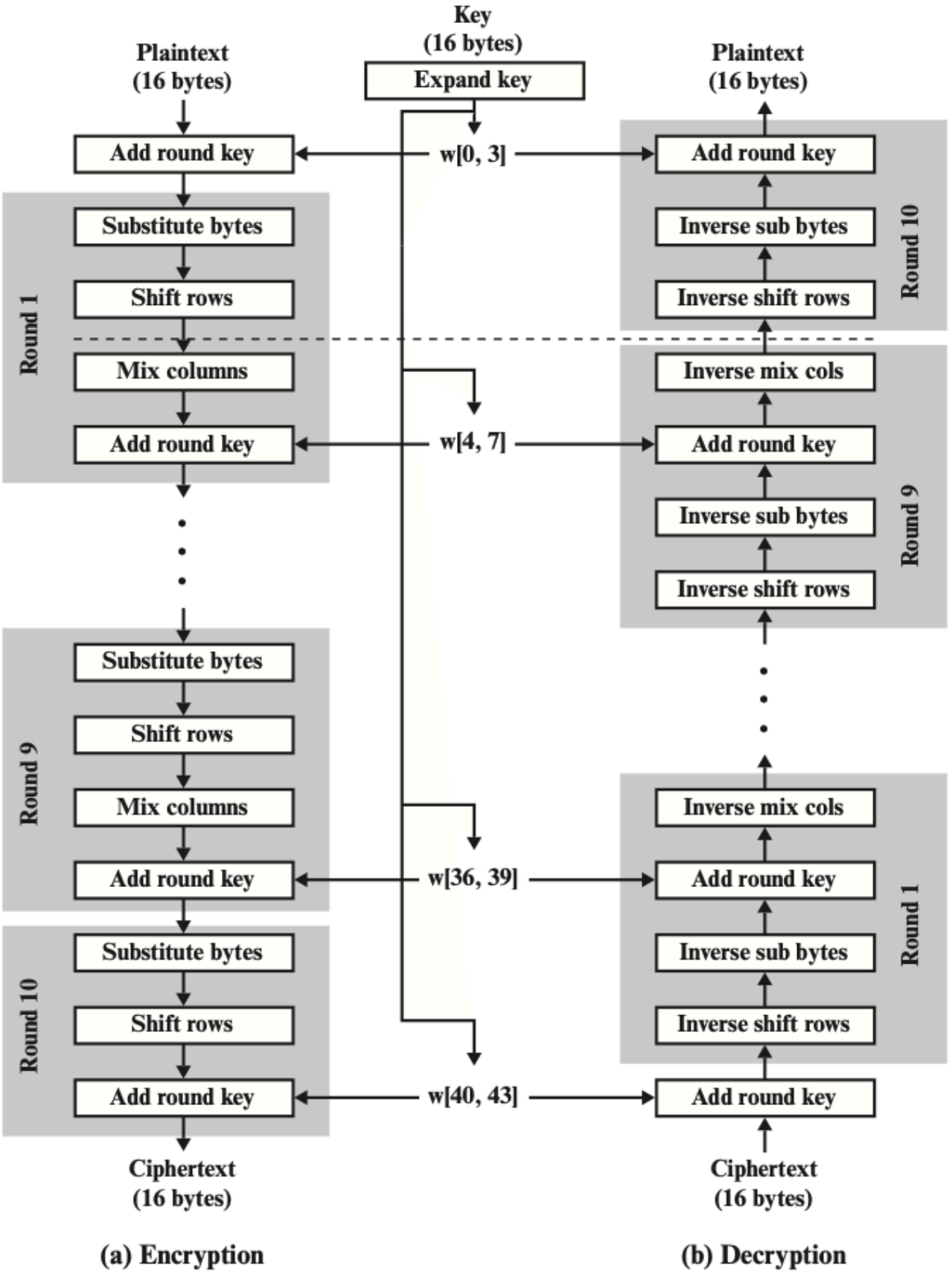


Figure 6.3 AES Encryption and Decryption

BLOCK CIPHER MODES

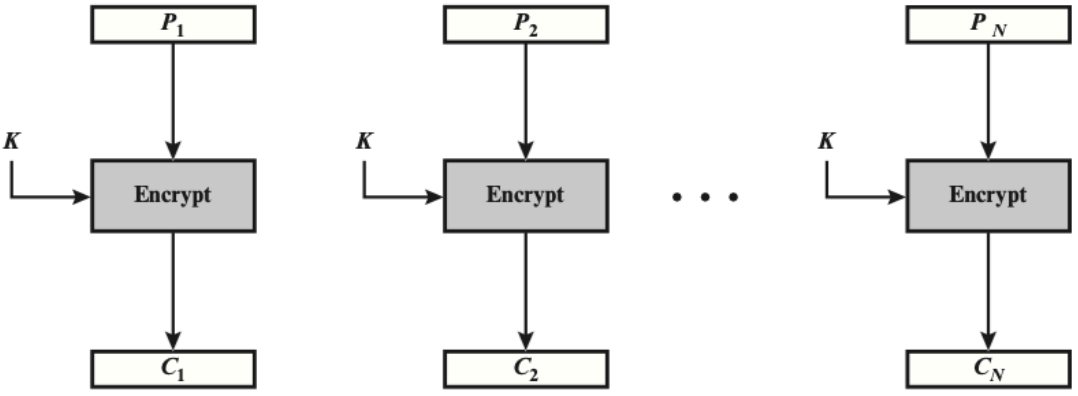
Ciphers must handle messages of arbitrary length. However, DES and AES above only handle inputs of fixed block size.

How do we make a cipher handle variable length input?
= Split messages in blocks and process them according to a cipher block mode.

ELECTRONIC CODE BOOK MODE (ECB MODE)

The simplest idea:

- Just split the input into blocks of the correct block size
- Encrypt each block separately
- Perform padding for the last block if necessary



(a) Encryption

- ➢ Pros
  - Very fast because it is completely parallelisable
  - Very simple to implement
  - If you want to access something in the middle of an encrypted file, it is very simple to do so (just find the right block and decrypt it)
- ➢ Cons
  - In the diagram above, if P_1 and P_3 are the same message, they will have the exact same cipher text.
  - 

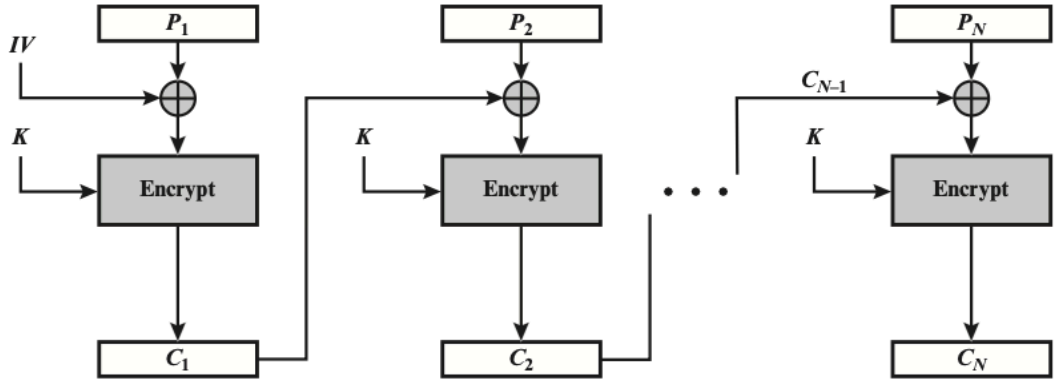
Figure: Before ECB encryption.

Figure: After ECB encryption.

CIPHER BLOCK CHAINING (CBC MODE)

We want a block mode that converts two identical blocks into separate cipher texts.

In CBC mode, the input to the encryption algorithm is the XOR of the current plaintext block and the preceding ciphertext block; the same key is used for each block. To produce the first block of ciphertext, an initialization vector (IV) is XORed with the first block of plaintext.

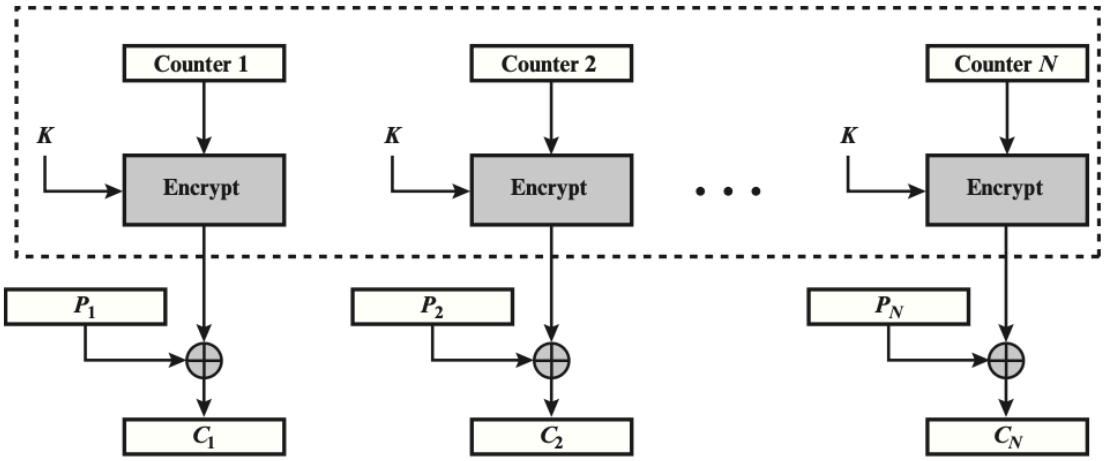


(a) Encryption

However, it is not parallelisable and is quite slow.

COUNTER MODE (CTR MODE)

Counter Mode is the most commonly used mode nowadays. It involves not encrypting the plaintext, but rather encrypting a nonce and then XORing this output with the plaintext block.



(a) Encryption

This fulfills all the properties we want: parallelisable, the same plaintext does not produce the same ciphertext, etc.

WEEK 5: ASYMMETRIC CRYPTOGRAPHY

Content
Asymmetric/Public-key cryptography: - What is it? - RSA Algorithm **Key exchange:** - Diffie-Helman key exchange

INTRODUCTION

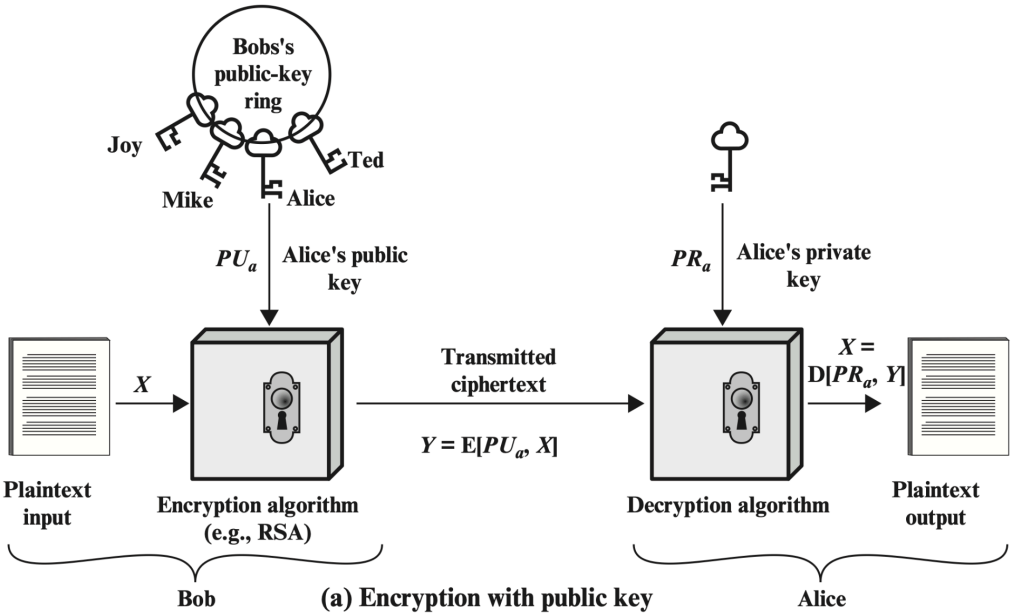
- ➢ Symmetric cryptography:
  - Sender and receiver use the same key to transfer the message
  - Two design goals: confusion and diffusion i.e. one character in the ciphertext is dependent on multiple characters in the plaintext, and one character in the ciphertext is also dependent on multiple characters in the key.
- ➢ Asymmetric cryptography:
  - Each participant has a public key (publicly distributed) and a private key (secret).
  - Anyone may use the receiver's public key to encrypt a message
  - Only the receiver can decrypt it using their private key

Public-key cryptography can be used in two main ways.

1. Confidentiality

If Bob encrypts a message using Alice's public key, then only Alice can decrypt it using her private key.

$$Y = E[PU_a, X]$$
$$X = D[PR_a, Y]$$

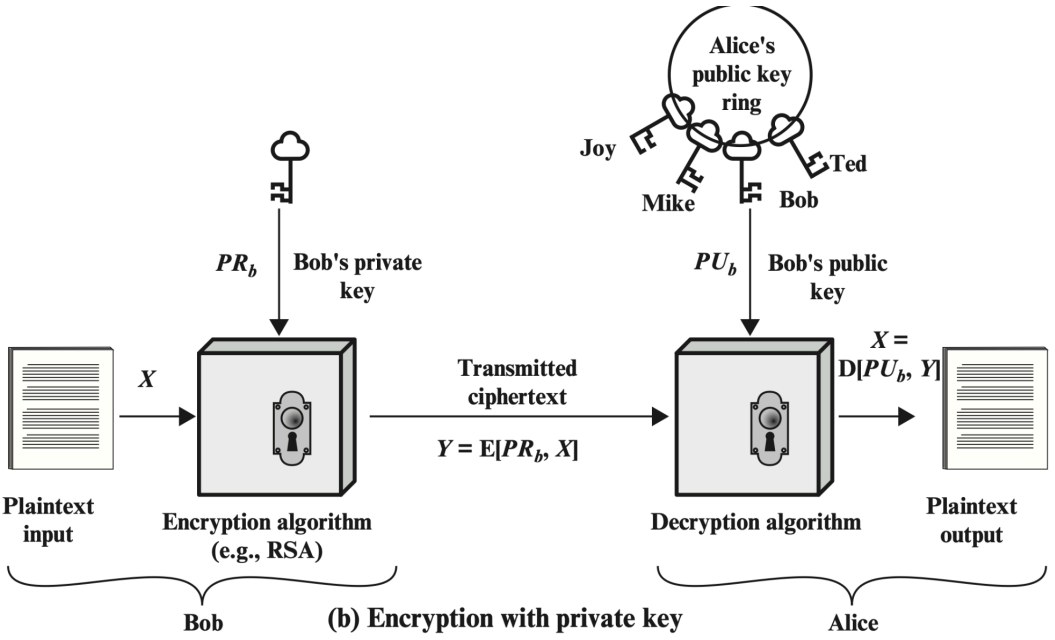


(a) Encryption with public key

2. Origin Authentication

If Bob encrypts a message using his private key, then sends the encrypted version to Alice, Alice can use Bob's public key to "decrypt" the message to check that the message came from Bob.

$$Y = E[PR_b, X]$$
$$X = D[PU_b, Y]$$



TRAPDOOR FUNCTIONS

We are looking for a function with the following properties:

- Computationally fast to compute the function value $f(x) = y$
- Computationally infeasible to compute the inverse function $f^{-1}(y) = x$
- Unless we are in possession of a piece of information that allows to speed it up dramatically
- Corollary: it must be computationally infeasible to compute the private key from the public key (without the trapdoor information)

DISCRETE LOGARITHM PROBLEM (DIFFIE-HELMAN)

Given y, g, and p, satisfying the equation

$$y = g^x \bmod p$$

Where p is a prime number and g is a primitive root of p, how do we find x?

RSA PROBLEM

When $c = m^e \bmod n$, and c, e, and n are known, can you find m?

COPRIME NUMBERS

We say two numbers are coprime if they share only one divisor, and that divisor is 1.

- Example: 8 and 27 are coprime:
  - 8 = 2 x 2 x 2
  - 27 = 3 x 3 x 3
- The original numbers do not need to be prime

We define $\mathbb{Z}_n^*$ as all non-negative integers $< n$ that are coprime to the integer n.

e.g. $\mathbb{Z}_{12}^* = \{1, 5, 7, 11\}$

THE RSA PROBLEM

RSA was developed in 1977 by Ron Rivest, Adi Shamir, and Len Adleman at MIT.

➢ Let p and q, $p \neq q$ and $n = p \cdot q$, be prime

- In practice these are very large prime numbers (need at least 2048 bits to represent the number)

➢ Calculate Euler's phi function: counts the number of elements in $\mathbb{Z}_n^*$
- For example:
  - $\mathbb{Z}_8^* = \{1, 3, 5, 7\}$
  - So $\phi(8) = 4$
- When p and q are prime, there is a simple formula to calculate phi:
  - $\phi(n) = (p-1)(q-1)$

➢ Choose e for the public key $\{e, n\}$. You can publish this set online for everyone to see.
- $1 < e < \phi(n)$
- $GCD(e, \phi(n)) = 1$, in other words, e is coprime with $\phi(n)$.
- Easy way to do this is to find $max(p, q)$ and then set e to be the next prime number larger than this max.

➢ Compute d, the private key.
- $d \equiv e^{-1} mod\ \phi(n)$
- $\Leftrightarrow de\ mod(\phi(n)) \equiv 1$
- i.e. d is the modular inverse of e

Key Generation by Alice

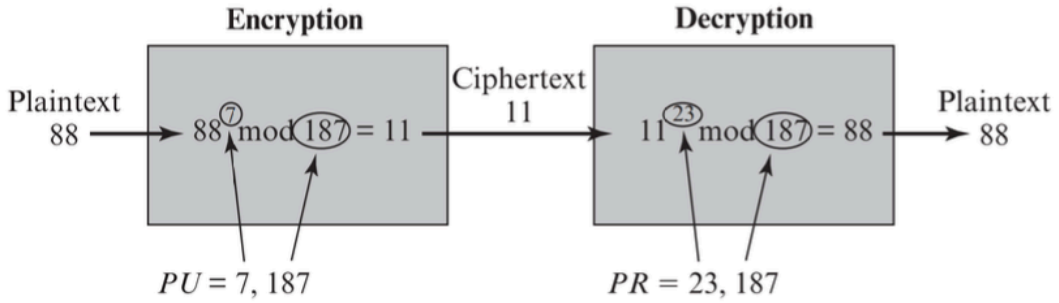
Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calcuate $\phi(n) = (p-1)(q-1)$	
Select integer e	$\gcd(\phi(n), e) = 1; 1 < e < \phi(n)$
Calculate d	$d \equiv e^{-1} \pmod{\phi(n)}$
Public key	$PU = \{e, n\}$
Private key	$PR = \{d, n\}$

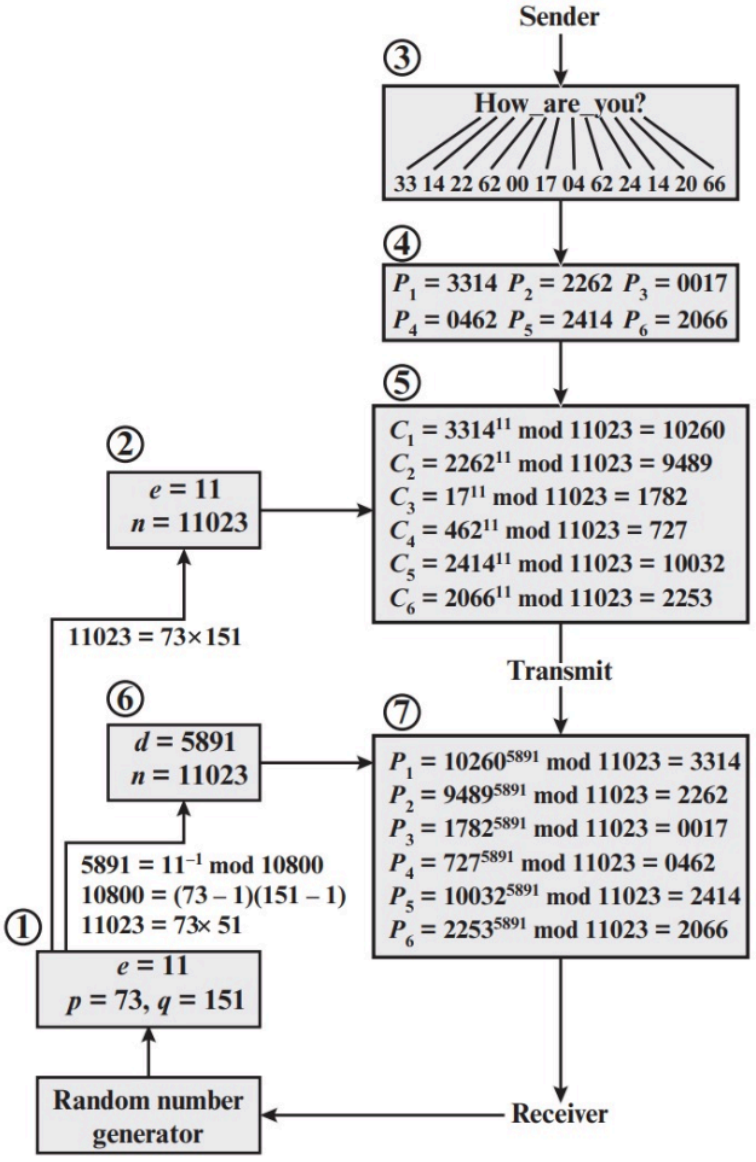
Encryption by Bob with Alice's Public Key

Plaintext:	$M < n$
Ciphertext:	$C = M^e \bmod n$

Decryption by Alice with Alice's Private Key

Ciphertext:	C
Plaintext:	$M = C^d \bmod n$





The security of RSA rests on:

- obtaining the e-th root of a number is computationally infeasible unless you know d
- factoring $n = pq$, for very large p and q, is computationally infeasible

Never use RSA without padding

WEEK 6: HASHES AND MESSAGE AUTHENTICATION CODES

Content
➢ Cryptographic hash functions ➢ Security Requirements for Cryptographic Hash Functions ➢ Secure Hash Algorithm (SHA) ➢ Message Authentication Codes (MACs) ➢ Hash MAC (HMAC) ➢ Authenticated Encryption (AE) ➢ Digital signatures

Week 4 + 5: How to encrypt data (confidentiality)
Week 7: How to exchange keys, how to authenticate parties

But we still need to guarantee:

1. Integrity: has the message been tampered with?
2. Authenticity: did it actually come from the person you think it did?
3. Non-repudiation: can the sender later deny sending it?

Week 6 teaches the tools for ensuring a message hasn't been tampered with (integrity) and that it came from the right sender (authentication).

- Hashes: integrity only
- MACs (HMAC): integrity + authentication using shared keys
- Authenticated Encryption: confidentiality + integrity + authentication
- Digital Signatures: integrity + authentication + non-repudiation with public keys

This lecture is about integrity and authenticity.

CRYPTOGRAPHIC HASH FUNCTIONS

"Digital fingerprints" of data. They provide integrity, collision resistance, preimage resistance, second-preimage resistance. Hash functions alone give integrity but not authentication.

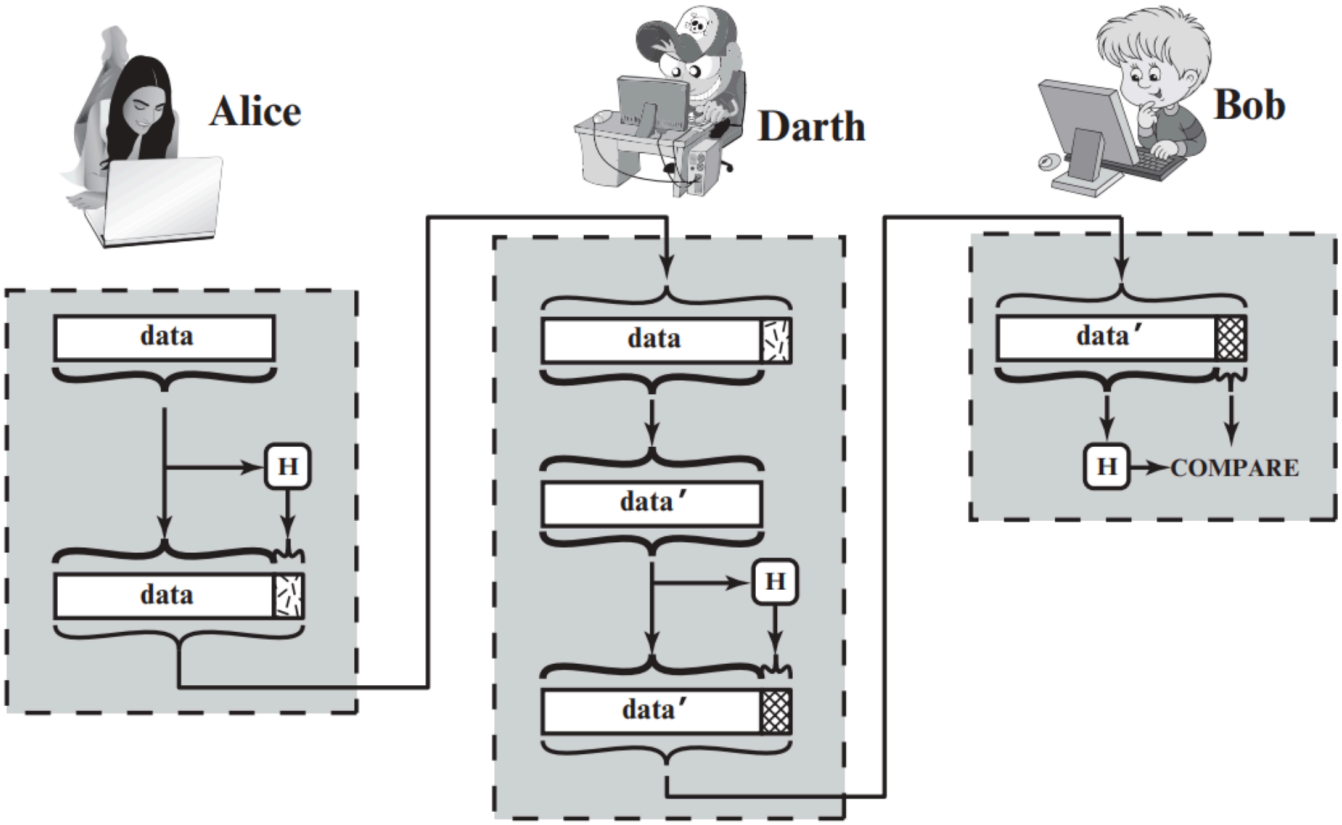
Idea: define a function to create a checksum ('hash function')

- Sender applies hash function to message and obtains a checksum ('hash value' or just 'hash')
- Sender encrypts message and sends together with hash value of plaintext
- Receiver decrypts message, applies hash function, and compares with the transmitted hash value
- If the hash values match, the plaintext must be correct

Hash functions produce a fixed-length output.

Three problems:

1. Attacker can get the plaintext from the hash value alone (defeats encryption)
2. Attacker can find a different bit sequence that has the same hash value
3. Attacker can perform a MITM attack



(b) Man-in-the-middle attack

FORMAL SECURITY REQUIREMENTS

All hash functions have fixed-length output. Hence, every hash function H has collisions, i.e., $H(a) = H(b)$, $a \neq b$.

PREIMAGE RESISTANCE

Given a hash output h, it should be computationally infeasible to find any message m such that H(m) = h.
i.e. “Find a message that produces this hash”, “Invert” the hash.

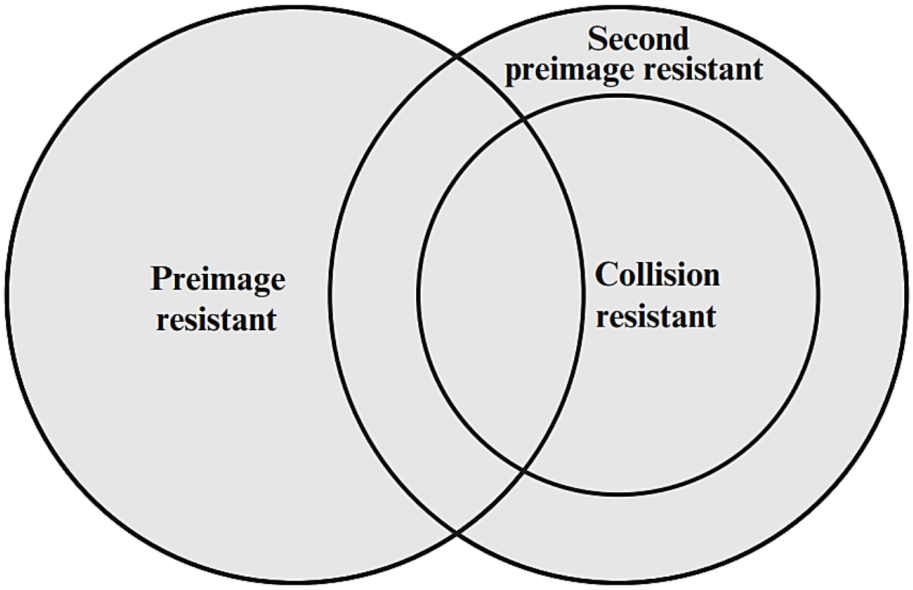
Note that it is not impossible – you can always brute-force it. But it is *computationally infeasible* i.e. too hard to do in practice.

SECOND PREIMAGE RESISTANCE

Given a message x, it should be computationally infeasible to find a different message x’ such that H(x) = H(x’).
i.e. “Find another input with the same hash.”

COLLISION RESISTANCE

It should be computationally infeasible to find any two distinct messages x and x’ such that H(x) = H(x’).



- ➢ Collision resistance ⇒ second preimage resistance
  - Identically, no second preimage resistance ⇒ no collision resistance
  - If I can take any given message x and find a different message x' with the same hash, that's a collision.
  - So, collision resistance is stronger than second preimage resistance.

To prevent collisions, we need hash functions to have a sufficiently long output length. Currently that sits at 224 bits.

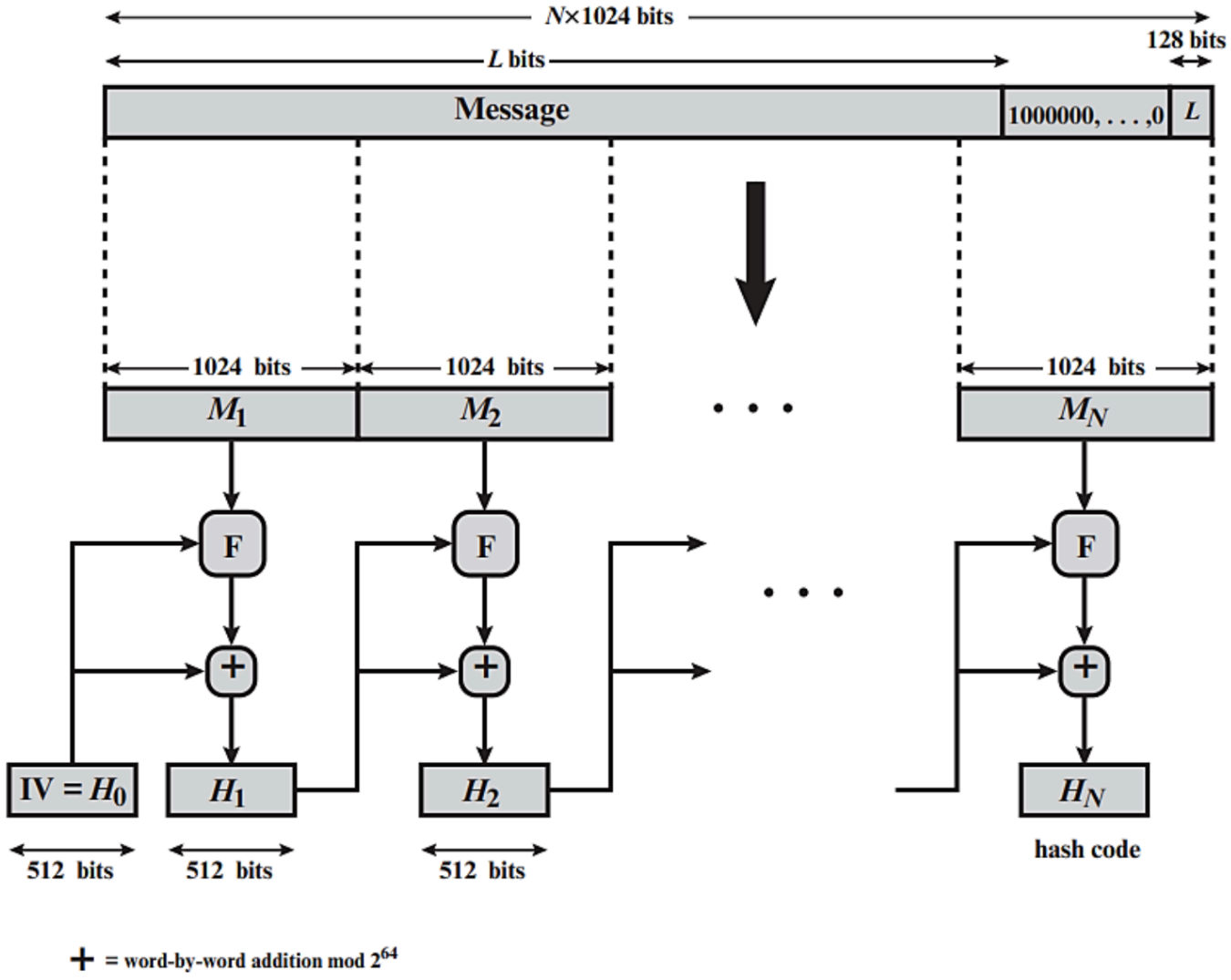
SECURE HASHING ALGORITHM (SHA)

SHA2

SHA2-224, SHA2-256, SHA2-384, SHA2-512

e.g. SHA2-512
One round:

- Takes input message with a maximum length of 2^128 bits and outputs 512 bit message digest.
- Input is processed in 1024 bit blocks.



SHA-512 has 80 rounds.

MESSAGE AUTHENTICATION CODES

Adds a (shared) key to hashing: "Anyone who knows the secret key can compute/verify the MAC." MACs provide integrity and authentication (because only someone with the key can produce the MAC). The most important MAC is HMAC.

AUTHENTICATED ENCRYPTION

Combines confidentiality (encryption), integrity and authenticity in a single construction, e.g. AES-GCM.

1. Hashing followed by encryption: $Enc_k(m,\ H(m))$
2. Authentication followed by encryption (MAC-then-encrypt): $Enc_{k_1}(m,\ MAC_{k_2}(m))$

DIGITAL SIGNATURES

MACs authenticate with a shared key. Digital signatures authenticate with public/private keys. Signatures provide non-repudiation which MACs alone cannot.

WEEK 7: AUTHENTICATION & KEY DISTRIBUTION

Content
➢ Symmetric key distribution (using symmetric encryption) ➢ Public-key/Asymmetric key distribution and digital certificates ➢ Public-key Infrastructure ➢ Authentication and key establishment (AKE) ➢ Remote user-authentication ➢ Kerberos

Recall:

- Week 4: Symmetric cryptography
- Week 5: Asymmetric cryptography
- Week 6: Hashing

When we discussed the above, we assumed that both parties had the required keys in order to perform decryption. But **how do entities transfer keys between them in order to perform cryptography?**

An aside on <u>Threat Models</u>

For this course we will only consider the **Dolev-Yao Model,** in which the attacker:

- Has full control over the network
    - "The attacker carries the message"
    - Can eavesdrop on any message (i.e. passive attack)
    - Delay, delete, modify, replace, and inject messages.
- Is limited by cryptography
    - Cannot modify integrity-protected message without the receiver noticing
    - Cannot decrypt the encrypted message

THE PROBLEM OF KEY DISTRIBUTION

Boyd's theorem

Assuming the absence of a secure channel, two entities cannot establish an authenticated session without the existence of an entity that can mediate between the two and which both parties trust and have a secure channel with.

In other words, if A and B want to establish a secure, authenticated session, they must already have existing, established keys OR they need a trusted third party to help them.

Why not Diffie-Hellman?

Diffie-Hellman key exchange is secure *if the attacker is passive.* With an active attacker, we need to protect the Diffie-Hellman key exchange with some form of origin authentication.

[REVISIT THIS PART OF THE LECTURE]

OPTIONS FOR KEY DISTRIBUTION

Without a third party:

- You could physically ship the keys
- This is expensive and not scalable → only done for very high-value communication e.g. with embassies.

With a third party:

- Symmetric case: Key Distribution Centers (KDC)
- Asymmetric case: Public Key Infrastructure (PKI)

These are both “trusted” third parties.

SYMMETRIC KEY DISTRIBUTION

How do we distribute keys between Alice and Bob when using symmetric encryption (i.e. the key is the same?)

Options:

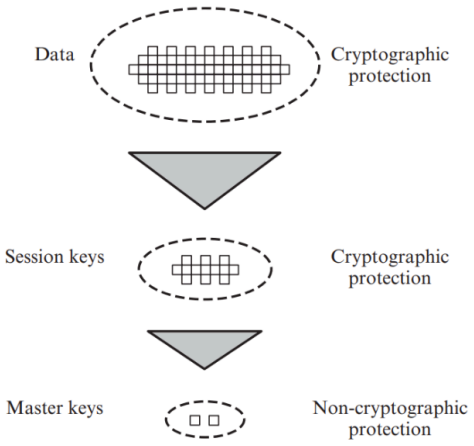
- ➢ Alice can select a key and physically deliver it to B
  - Expensive and not scalable
- ➢ If Alice and Bob have previously used a key, one party can transmit the new key to the other encrypted using the old key.
  - But, if an attacker ever gains access to one key, then all subsequent keys will be revealed.
- ➢ If Alice and Bob each have an encrypted connection to a third party Charlie, Charlie can deliver a key on the encrypted links to Alice and Bob.

KEY DISTRIBUTION CENTRES

Symmetric Key Distribution using Symmetric Encryption: how do Alice and Bob both get the same key?

A key distribution centre (KDC) is a trusted third party responsible for distributing keys to pairs of users as needed. KDCs depend on the use of a hierarchy of keys.

- ➢ **Session key**
  - For communication between the end systems
  - Used for the duration of a logical connection and then discarded
  - Transmitted in encrypted form using the master key
- ➢ **Master key**
  - Is shared by the KDC and and an end system
  - Distributed in some non-cryptographic way, e.g. physical delivery, comes preinstalled with software



KDCs are the basis for the famous Kerberos protocol, used across many OSes.

NEEDHAM-SCHROEDER PROTOCOL

Assuming that:

- Each user shares a long-term symmetric master key with the KDC
  - User A has master key K_a, known only to itself and the KDC
  - Similarly, User B has a master key K_b

- User A wants to establish a logical connection with User B and requires a one-time session key K_s to protect the data transmitted over the connected
- N_1 is a unique identifier (also called *nonce*). It could be a timestamp, counter, or random number, as long as it differs with each request.
- ID_A, ID_B are the unique identifiers for A and B e.g. network address

1. A issues a request to the KDC for a session key to protect a logical connection to B.

   ID_A || ID_B || N_1

2. The KDC responds with:

   E(K_A, [K_s || ID_A || ID_B || N_1])
   and
   E(K_B, [K_s || ID_A])

   a. Message 1:
      i. The one-time session key K_s to be used for the session
      ii. The original request message, including the nonce. This enables A to verify the *freshness* of a response – that the reply came in reaction to this specific request and is not just some old message being replayed. If there were no nonce, an attacker could trick A into reusing an old session key.
      iii. This message is encrypted using K_A, so only A can read the message.
   b. Message 2: (will be sent to B, not read by A)
      i. The one-time session key K_s to be used for the session
      ii. The identifier of A, ID_A

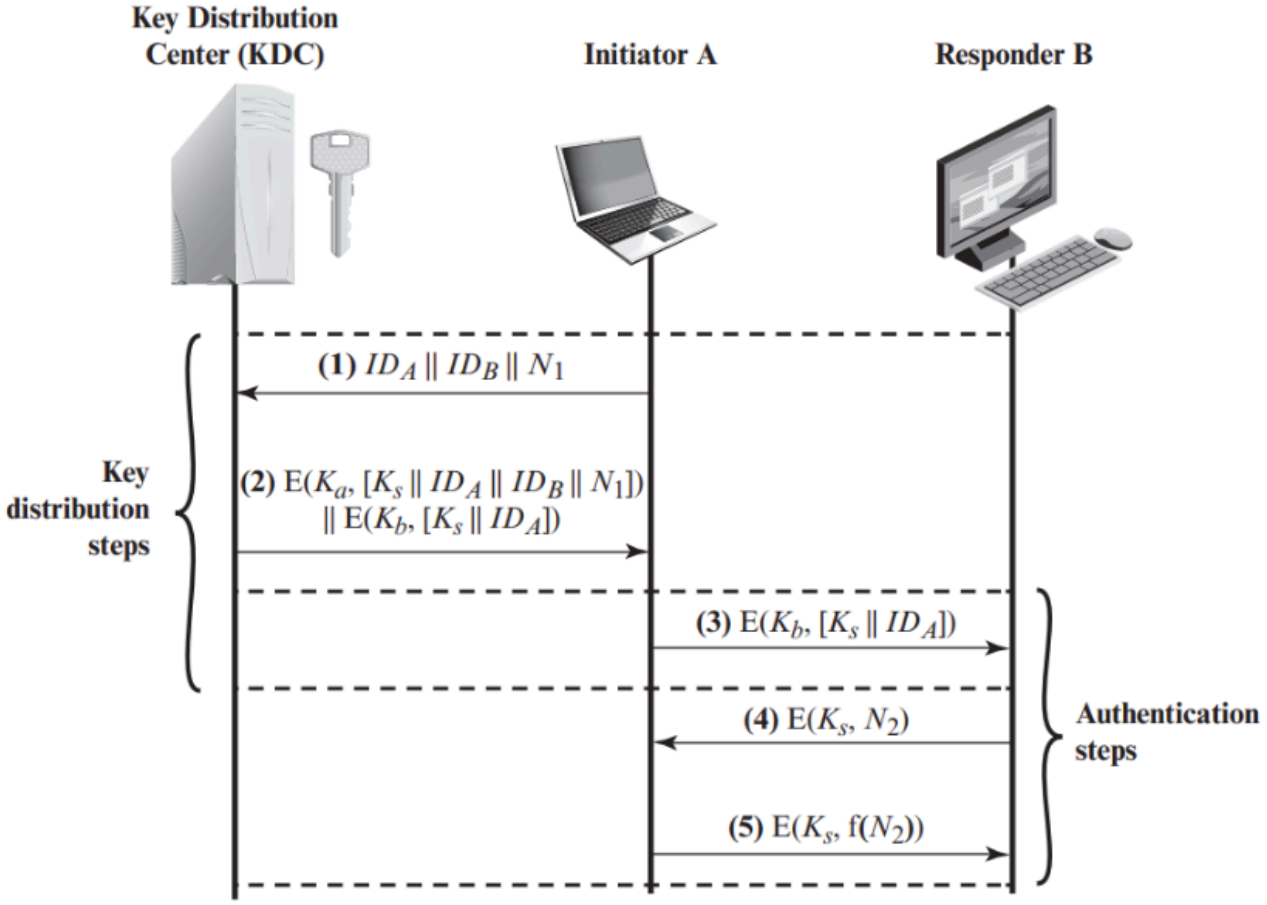
3. A will store the session key for use in the upcoming session and forward to B the information from the KDC intended for B: E(K_B, [K_s || ID_A]).
   a. Because this information is encrypted with K_B, it is protected from eavesdropping.
   b. B now knows the session key, knows that the other party is A (from ID_A), and knows that the information originated at the KDC (because it is encrypted using K_B).

At this point, a session key has been securely delivered to A and B and they may now begin their protected exchange.

4. B will send a message to A which has been encrypted with K_s:

   E(K_s, N_2)

   a. B will also send back a new nonce N_2 to prove freshness
5. A will send back a message to B, with some function applied to N_2 to again prove freshness (the function could just be adding 1 to the number).



This is also called Needham-Schroeder protocol

PUBLIC KEY DISTRIBUTION

Asymmetric Key Distribution using Certificates: how does Alice get Bob's public key and truly know it is Bob's

There are 4 possibilities to securely distributing public keys:

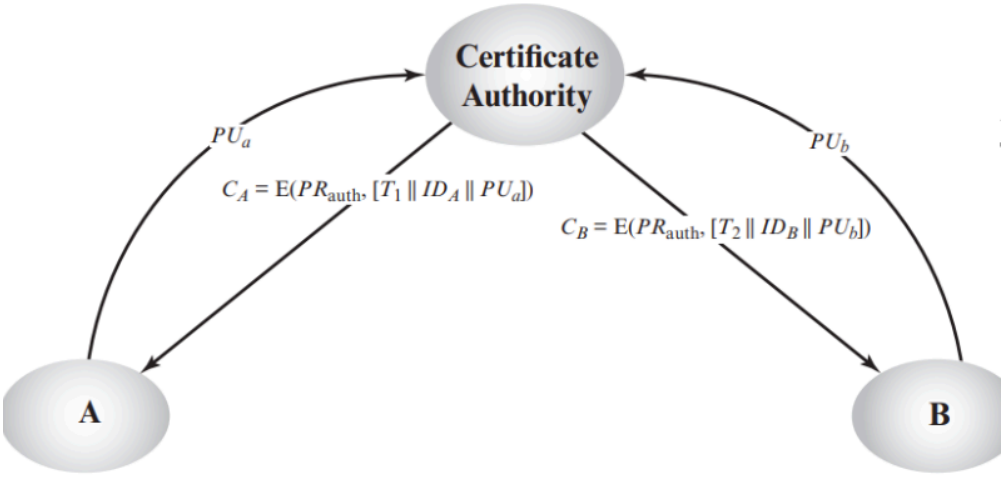
1. Public announcement
    a. Alice can send their public key to Bob, or broadcast the announcement.
    b. Weakness: Man-in-the-middle attacks, anyone can forge a public announcement
2. Publicly available directory (like a phone directory)
    a. Maintain a public, dynamic directory of public keys
    b. Weakness: If the directory is compromised, the contents of the directory can be forged. You will think you're talking to X but it's actually an attacker.
3. Public key authority
    a. Based on Needham-Schroeder protocol.
    b. Weakness: Scalability, could make public-key authority a bottleneck in the system
4. Public key certificate
    a. Can be used by anyone to exchange keys without contacting a public key authority
    b. Public Key Infrastructures (PKIs)
    c. We share certificates and the recipient can check if our certificate is valid.

PUBLIC KEY INFRASTRUCTURES (PKIs)

CERTIFICATES

A certificate is a cryptographic binding between an identity and a public key. It is signed by a trusted third party – Certificate Authority (CA) – in accordance with Boyd's theorem.

OBTAINING CERTIFICATES



(a) Obtaining certificates from CA

1. A sends the CA their public key, PU_A.
2. Once the CA is satisfied that you are actually A (e.g. by asking you to prove that you control a domain), they will create a structure like this:

   [T_1 || ID_A || PU_A]

   a. T_1, a timestamp (e.g. when did they sign it or how long this certificate is valid for)
   b. ID_A is the identity of A (e.g. domain name)
   c. PU_A, the public key of A.

3. The CA digitally signs it with its own private key PR_auth, creating the certificate:

   C_A = E(PR_auth, [T_1 || ID_A || PU_A])

Because the CA signs it with its private key, anyone with the CA's public key can decrypt it.

How does the CA prove that A is actually A?
i.e. "Prove to me that you are the owner of this website."
There are industry-agreed steps that must be carried out which vary depending on the purpose of the certificate and its value.

Baseline verification:
- Upload a file with this specific format to this part of your website
- Receive email under that domain name

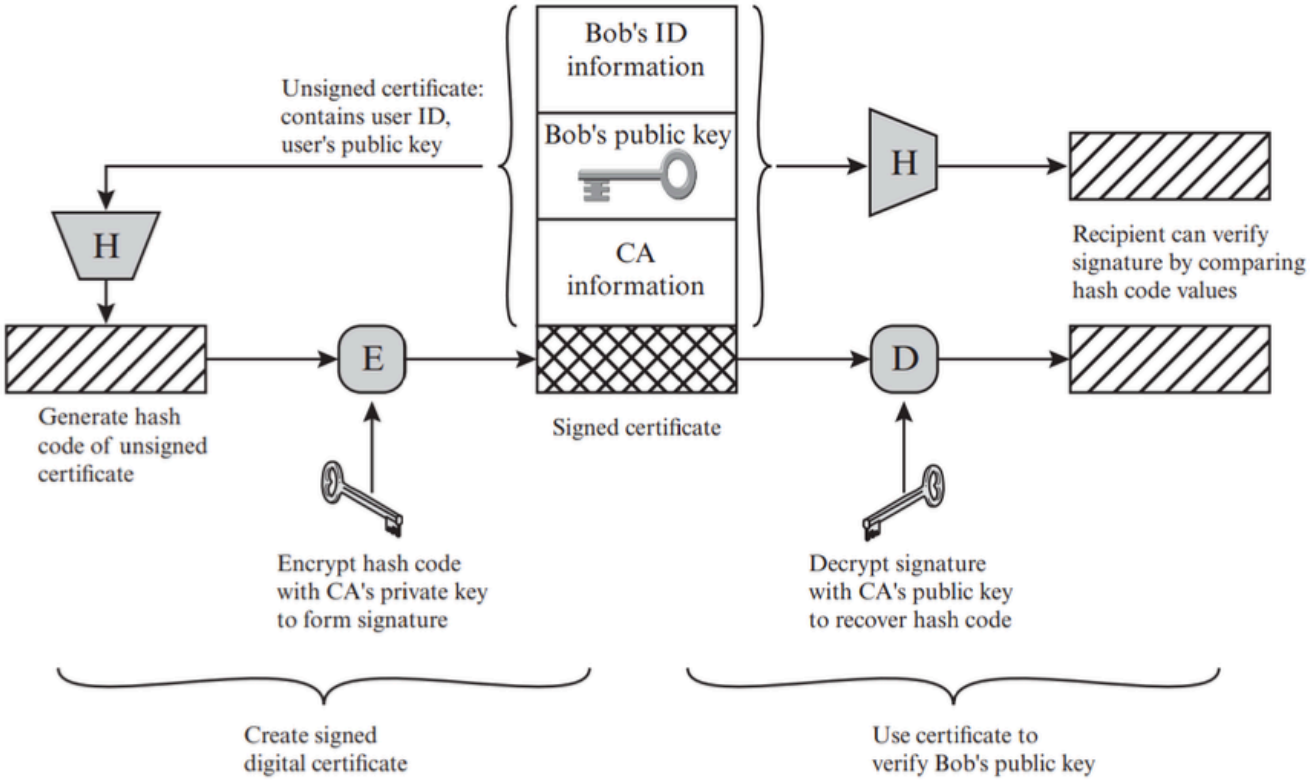
Extended verification:
- Uses legal documents
- Expensive form, but is done for high-importance purposes

USING CERTIFICATES

Now, A and B can exchange certificates. They can “decrypt” or verify the certificates by using the CA’s public key. If the verification works, they know:

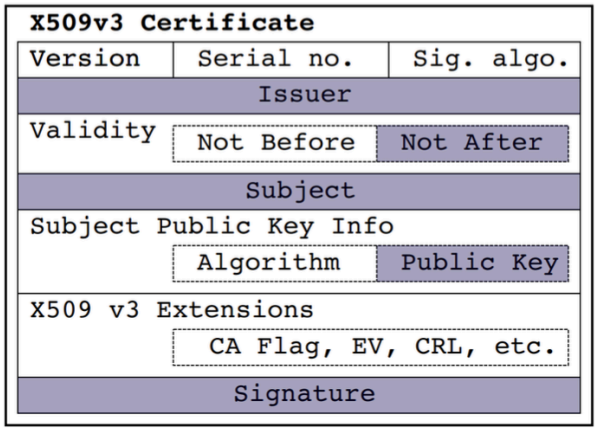
1. The certificate really came from the CA
2. The data inside hasn’t been altered
3. PU_A indeed belongs to identity ID_A and PU_B really belongs to identity ID_B.

Now A has B’s public key (and knows it’s definitely from B) and vice-versa. They can now safely perform the Diffie-Helman key exchange and agree on a key.



X.509 CERTIFICATE

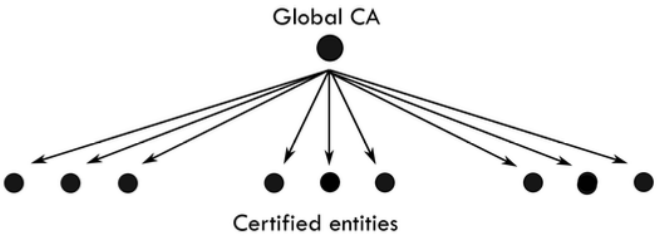
The X.509 standard defines the format of public-key certificates.



X.509 Format

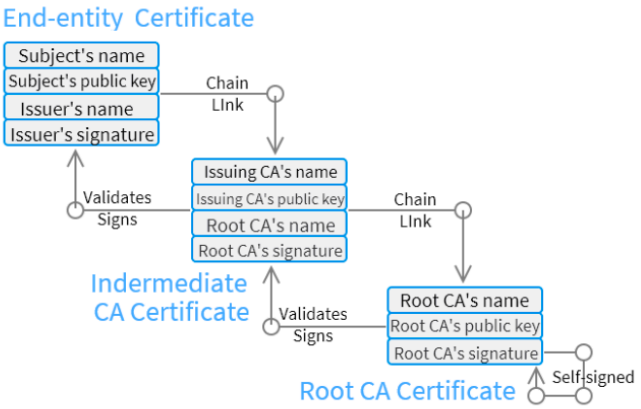
ROOT STORES & MULTIPLE CERTIFICATION AUTHORITIES

Naive form: one global issuer trusted by everyone

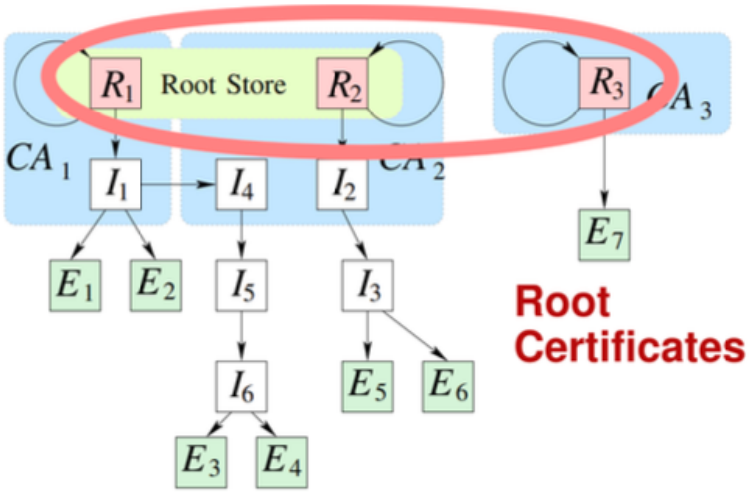


Why does this not work?

- Different jurisdictions have different identity requirements
- Performance bottleneck – only one global CA issuing all certificates
- Increases vulnerability for the global CA whose private key has been used to sign millions of certificates → online day-to-day operations can be done using the private key of the intermediate authority.



- Some root authorities exist e.g. CloudFlare. These authorities will have *self-signed* certificates.
- Root certificates will certify intermediate authorities who then ultimately certify end users.
- These are called *Intermediate Certificates.*



Hence, we now have many CAs and accept them all as “trusted.”

- The public keys of “trusted” CAs are shipped with your OS, browser in **root stores**.
- Root stores hold certificates of trusted CAs
- Every application that uses X.509 has to have a root store
- Operating Systems have root stores: Windows, Apple, Linux, etc.

- Browsers have root stores

CERTIFICATE REVOCATION

Once a certificate has been issued, the CA has no control over that certificate anymore. So, what happens if a key is compromised after issuance?

CERTIFICATE REVOCATION LISTS (CRLs)

Each Certificate Authority updates and maintains Certificate Revocation Lists, which contain a list of certificates that are considered revoked.

- A browser (client) should download CRL, update it after a given time, and lookup a host certificate every time it connects to a server
- Too slow → not done anymore in practice

ONLINE CERTIFICATE STATUS PROTOCOL (OCSP)

1. User wants to visit a domain
2. The browser checks the browser's certificate (and each intermediate certificate until the root certificate) to check if it is in the root store.
3. If it is, it performs one final check:
    a. It queries the OCSP to check if that certificate is *still valid*.
    b. Response contains cert status, must be signed

AUTHENTICATION

We briefly mentioned authentication in the context of access control, when we discussed things like users and privileges:

- Before you can use the filesystem, you need to login
- The login is a form of authentication
- After authentication, you have authorisation for certain things

Formal definition:
Authentication is the process whereby one party is assured (through the acquisition of corroborative evidence) of the identity of a second party involved in a protocol, and that the second has actually participated.

Note the elements needed to ascertain the identity:

- Corroborative evidence
- Process between at least two parties
- Involvement and participation of the second party

CORROBORATIVE EVIDENCE

We are looking for factors that are unique to an entity. Classic categorisation:

- ➢ Possession: Something the entity/user has
    - Physical key, phone to send SMS messages to
- ➢ Inherence: Something the user is
    - Biometrics: fingerprints, iris scan, face and voice recognition
- ➢ Knowledge: Something the user knows
    - Passwords, security questions (mother's maiden name, etc.)

POSSESSION

Something the user has.

Problems:

- ➢ Loss of the physical device
- ➢ A physical card only checks the possession of the ID, not who possesses it
- ➢ One-time password sent to SMS only checks who has the phone → who has the SIM card
  - SMS messages are sent to SIM card → can be rerouted or stolen

INHERENCE

Something the user is.

Problems:

- ➢ Some can be easily inferred
  - Fingerprint recognition is getting better
  - 3-D models of faces
- ➢ If biometric is gone, it is annoying or impossible to use
  - Injured finger → no fingerprint sensor

KNOWLEDGE

Something the user knows.

We have already discussed the usability and strength of passwords

- Note that databases of cracked passwords exist
- https://haveibeenpwned.com

Problems:
Security questions are often very standard, with predictable answers and limited possibilities

- Mother's maiden name? - depending on culture, try Smith, Chang, Kim, Schmidt, …
- First car? – try Golf, Yaris, Corolla, …
- Social networks help collect additional information about a person

MULTI-FACTOR AUTHENTICATION

Increased security by requiring multiple factors for authentication (in general from different categories).

- Example: password and code sent as text to phone
  - Checks knowledge and possession
  - Needs very motivated attacker, and assumes targeted attack, not broad sweep
- Often, MFA comes with mechanisms that check the plausibility
  - e.g. if a request to login is coming from an unusual country, from which the user never logged in before
  - or if many failed login attempts occur
  - or if IP is from a range that is known to be the origin of other attacks

PROCESS & INVOLVEMENT OF TWO PARTIES

Solved using *authentication protocols.*

What's the challenge of measuring participation?

- When you are logging into a VM or some console, you have to manually type in a password. This guarantees participation.

- But what about when two machines authenticate each other?

AUTHENTICATION PROTOCOLS

- ➢ Sequence of well-defined steps to achieve a goal: this is a **protocol**
- ➢ If we add cryptographic protection: **cryptographic protocol**

To do this, we need to put together everything we have learnt so far (primitives):

- ➢ Cryptographically secure random numbers
  - Essential ingredient in all that follows
- ➢ Cryptographic hash functions give us:
  - MACs (by mixing in shared secret)
- ➢ Public-key cryptography gives us:
  - Key exchange over certain untrusted channels
  - Confidentiality (→ hybrid: encrypt symmetric keys)
  - Origin authentication and data integrity
- ➢ Symmetric-key cryptography gives us:
  - Confidentiality (→ actual data transfer)
  - Origin authentication and data integrity → MACs

Kerberos = OKTA

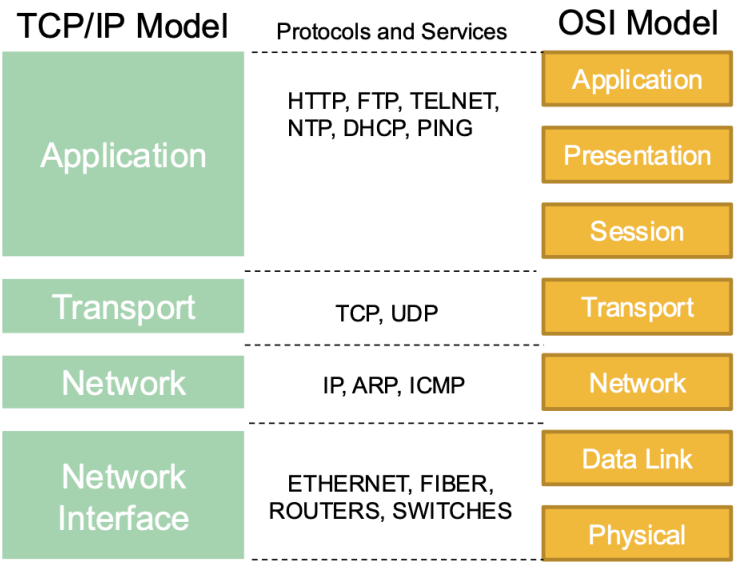
WEEK 8: NETWORK SECURITY – PROTOCOLS

A **protocol** defines the format, order of messages sent and received among network entities, and actions taken on message transmission and receipt.

THE TCP/IP MODEL

The four layers of the TCP/IP model:

1. Application layer
2. Transport layer
3. Network layer
4. Network interface layer (it can be separated as data link layer and physical layer)



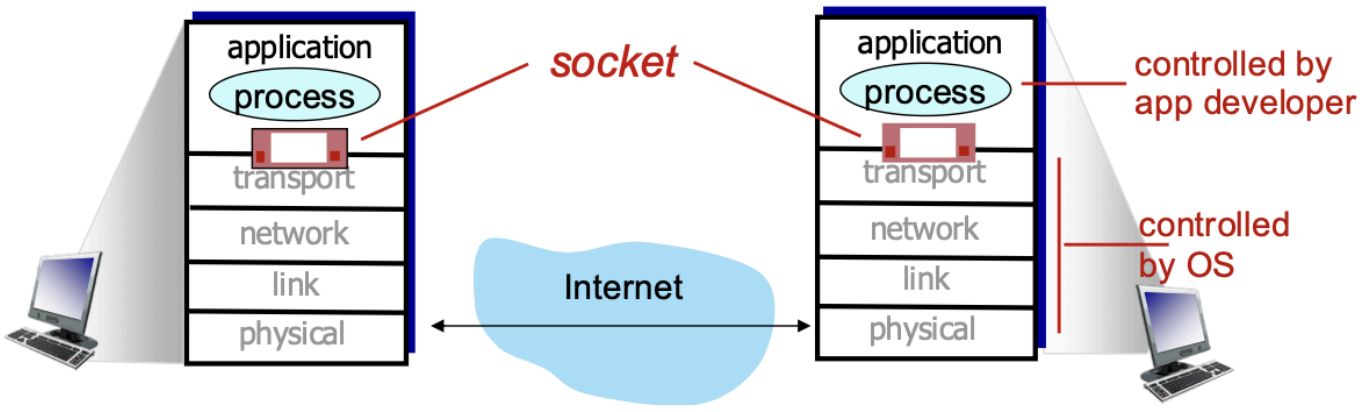
APPLICATION LAYER

The Application layer of the TCP/IP model is where network applications and their application-layer protocols reside. For example:

- HTTP (Hypertext Transfer Protocol)
- SMTP (Simple Mail Transfer Protocol)
- FTP (File Transfer Protocol)
- DNS (Domain Name System)

SOCKETS

- A **process** is a program running within a host
- A process sends/receives messages to/from its **socket**
- A socket is analogous to a door
  - Sending process shoves message out door
  - Sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process
  - Two sockets involved: one on each side



TRANSPORT LAYER

The Transport layer of the TCP/IP model provides logical communication between application processes running on different hosts.

- sender: breaks application messages into **segments**, passes to the network layer
- receiver: reassembles segments into messages, passes to the application layer

Two main transport protocols: TCP and UDP

UDP (USER DATAGRAM PROTOCOL)

"Send and hope for the best"

- Segments may be lost or delivered out of order
- But there is extremely limited overhead
    - No setup/handshaking needed
    - Can function when network service is compromised

Hence, UDP is used in streaming apps (even if a few segments are missing its okay + best use of bandwidth), DNS.

UDP looks like:
PC1: I want to send you some data. It will be in 12 packets, each 256 bytes long.
PC2: I am ready to receive 12 packets, each 256 bytes long.
PC1: I am sending packet 1 of 12.
PC1: I am sending packet 2 of 12.
PC1: I am sending packet 3 of 12.
PC1: I am sending packet 4 of 12.

TCP (TRANSMISSION CONTROL PROTOCOL)

"Reliable, in-order byte stream":

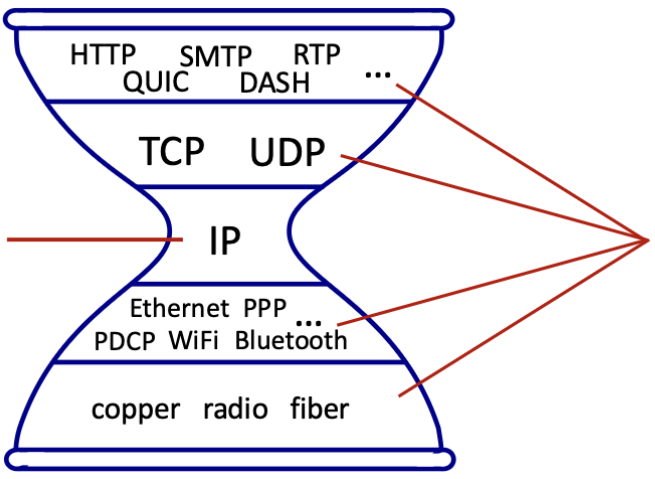
- Connection-oriented: 3-way handshake (exchange of control messages) initialises sender-receiver state before data exchange
- Flow controlled: sender will not overwhelm receiver

TCP looks like:
PC1: I want to send you some data. It will be in 12 packets, each 256 bytes long.
PC2: I am ready to receive packet 1 of 12.
PC1: I am sending packet 1 of 12, it is 256 bytes long.
PC2: I have received packet 1 of 12, it was 256 bytes long. I am ready to receive packet 2 of 12.
PC1: I am sending packet 2 of 12, it is 256 bytes long.
PD2: I have received packet 2 of 12, but it was only 183 bytes long. Please resend.
PC1: I am sending packet 2 of 12, it is 256 bytes long.
PC2: I have received packet 2 of 12, it was 256 bytes long. I am ready to receive packet 3 of 12.

NETWORK/INTERNET LAYER

Internet's "thin waist":

- **one network layer protocol: IP**
- must be implemented by every (billions) of Internet-connected devices



many protocols in physical, link, transport, and application layers

LINK LAYER

PROTOCOLS BY LAYER

Computer networks are built in layers (the TCP/IP model):

1. Application Layer – user-facing (e.g., web, email)
2. Transport Layer – moves data between apps (e.g., TCP, UDP)
3. Network Layer – moves packets between machines (e.g., IP)
4. Link Layer – local data transfer (e.g., Wi-Fi, Ethernet)

Each layer can be attacked in different ways (eavesdropping, spoofing, tampering, etc.), so we use security protocols at each layer to defend against those attacks.

Layer	**Example Security Protocol**	**Main Goal**
Application	OAuth, OpenID	Authenticate users & authorize access
Transport	TLS (was SSL)	Encrypt data between apps (e.g., HTTPS)
Network	IPSec	Secure all IP packets (VPNs, routers)
Link	WPA2, WPA3	Secure Wi-Fi connections

APPLICATION LAYER: OAUTH AND OPENID

OAUTH (OPEN AUTHORIZATION)

Purpose: Authorization – letting one service access another service on behalf of a user, without giving away the user's password.

Key Idea: Instead of sharing credentials (like your Google password), OAuth lets you grant limited access via a token.

Example: You let a calendar app read your Gmail contacts → Gmail issues a secure access token to that app after you approve it.

Role	**Description**
Resource Owner	The user who owns the data (you).
Client	The app requesting access (e.g., calendar app).
Authorization Server	Issues tokens after the user grants permission (e.g., Google Auth server).
Resource Server	Hosts the protected data (e.g., Gmail contacts).

How it works:

1. Client requests permission from the user.
2. User is redirected to Authorization Server (e.g., Google login).

3. User logs in and grants access.
4. Authorization Server gives the client a token.
5. Client uses the token to access the Resource Server's data.

Notes:

- Tokens are limited (e.g., read-only, time-bound).
- Uses HTTP and HTTPS for transport.
- Complex protocol

OPENID

Purpose: Authentication – verifying who you are. It's built on top of OAuth 2.0, using the same flow but for identity, not access.

How it works:

- The site (client) doesn't ask for your password.
- Instead, it redirects you to Google (OpenID Provider).
- Google authenticates you and sends back an ID token proving your identity.

TRANSPORT LAYER: TLS

TLS sits between the application (like a web browser) and the transport layer (TCP). It is sometimes called "layer 4.5".

Goals:

1. Confidentiality: encrypt data so attackers can't read it
2. Integrity: ensure data hasn't been altered
3. Authentication: verify who you're talking to (e.g., a real server, not a fake one)

INTERNET LAYER: IPSec

IPSec (Internet Protocol Security) is a suite of protocols that secures data at the Network Layer, the same layer where IP packets live. That means it can protect any kind of network traffic (web, email, file transfer, etc.), without apps even knowing.

Security Goal	**What It Means**
Confidentiality	Encrypts packets so outsiders can't read them
Integrity	Detects if packets are tampered with
Authentication	Confirms packets really come from the right source
Replay Protection	Stops attackers from re-sending captured packets
Access Control	Can restrict which IPs or protocols are allowed

WEEK 9: NETWORK SECURITY – FIREWALLS

WEEK 10: SOFTWARE SECURITY

WEEK 11: WEB SECURITY

The Web is the largest application running on top of the internet.

- Originally envisaged as a way to share research results between academic institutes.
- Designed for document sharing → not really made with security in mind.

Principles:

- ➢ All input is evil. Never trust input data.
- ➢ Careful checking of workflows and data flows required when designing a site to avoid data leakage.
  - e.g. Adding an item to cart → checking out. This is a workflow.
- ➢ Access control is a workflow! It is not provided as part of the protocols in any meaningful way.
  - You need to check the access control workflow carefully, because it is not simply implemented by any protocol.

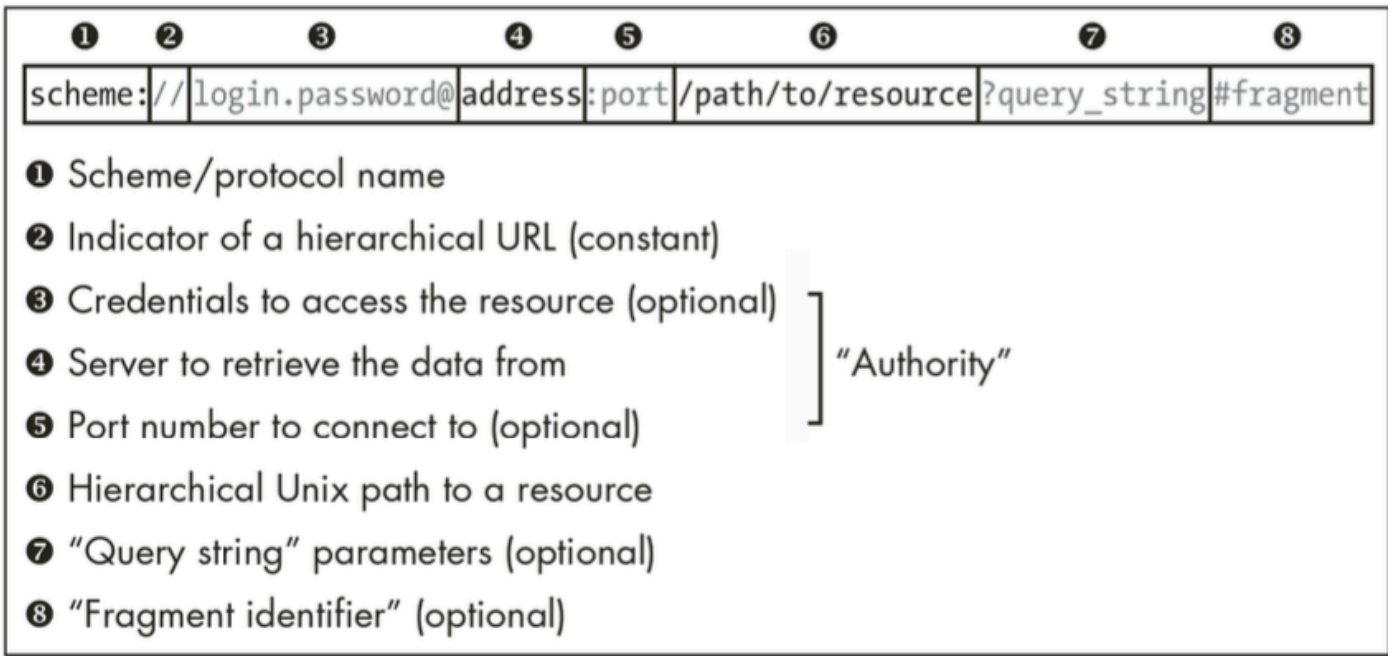
- The Web is a giant distributed system of many different technologies.
- Securing your web app requires knowing precisely what each tech does, and how to secure it.

INTRODUCTION TO THE WEB

URL: Uniform Resource Locator

- ➢ Every resource on the web needs some sort of address
  - Image, form, page
- ➢ URL defines an address

Example: https://m.facebook.com/index.php



1. Scheme/protocol name
    a. HTTP, HTTPS, FTP, SMTP, etc
2. Indicator of a hierarchical URL (constant)
3. Credentials to access the resource
    a. We do not use this anymore as it's insecure!
4. Server to retrieve the data from
    a. Domain name → DNS lookup
    b. Or just directly put in the IP address
5. Port number
    a. Browser will automatically put this in
    b. HTTPS: 443 by default
6. Path to the resource
    a. Relative to the root of the web server, where is this resource located?
7. Query string
    a. One way to give inputs to the server
8. Fragment identifier
    a. Which part of the webpage you are referring to

Authority

DOMAIN NAMES

➢ Domain Name System (DNS) is used by browsers to map domain names to IP addresses

- DNS system was originally only for English, later extended to international characters.
- But the way they extended it was to convert any other characters into some sort of Latin alphabet representation.
    - e.g. ΓΝΩΘΙΣΕΑΥΤΟΝ.com → xn--mxadglfwep7amk6b.com
    - This is called **punycode**
    - The prefix xn shows that the domain name is in a different language

Problem: homoglyphs

- The Greek letters alpha-rho-rho-iota-epsilon look like: αρριε
- This is http://xn--80ak6aa92e.com
- But will be rendered as www.αρριε.com in some browsers

HTTP AND HTTPS

HTTP: protocol to communicate with Web servers and get/put content.

- Several types of requests: GET, POST, PUT, DELETE
- Parameters may be sent via the URL (query string)
- More parameters can be sent via headers
- Some requests allow data sent in the body of the request e.g. JSON and XML

HTTPS = HTTP used over TLS

WEEK 12: GUEST LECTURE

WEEK 13: REVIEW

What the hell is actually going on here? What's the big picture?

Client talks to Server (e.g. your browser and Google).

FIRSTLY, how does the Client know that it's actually talking to the server, and not some man in the middle?
- The browser downloads the server's certificate
- The browser verifies the signature

SECONDLY, these guys need to talk securely i.e. they want all their communication to be encrypted. How do they encrypt the data?

(1) They could use symmetric cryptography
- They could use a stream cipher?
  - ChaCha20
- They could use a block cipher?
  - DES? 3-DES? = deprecated
  - AES?

(2) They could use asymmetric cryptography, in particular RSA? i.e. they could encrypt every message with the other machine's public key.

Why do we not use asymmetric cryptography to encrypt all data?
- Too slow. RSA is 1000-100,000 times slower than AES
- Uses huge numbers and therefore is CPU and energy intensive
- RSA can only encrypt a tiny amount of data (a 2048-bit RSA key can only encrypt 256 bytes). Cannot encrypt images, webpages, emails, etc.
- RSA is vulnerable to key compromise → does not allow perfect forward secrecy

Okay, so we use (1) symmetric cryptography.

Now the issue is that because symmetric cryptography uses the same key for encryption and decryption, both the server and the client need to actually GET the same key in the first place. How do we do this?

Originally, the key was generated on the client-side and then encrypted using the server's RSA public key. The server could then decrypt it using its own private key. But, this doesn't allow for perfect forward secrecy because if an attacker records the encrypted key, and then 5 years later gets access to the server's private key, they can get access to the key and therefore all old traffic between the server and client.

So, now we use Diffie-Helman, which does not encrypt the key at all, but instead allows both sides to derive the key mathematically using public DH values.

IN CONCLUSION, the client now knows that it is definitely talking to the server, and all future communication between the client and server will be encrypted using a symmetric encryption algorithm like AES or ChaCha20 because we used DH to securely share a key between them.