Security and Privacy

Midterm Exam Notes

Isaac Metthez

November 6, 2025

1 Definition and Basics

1.1 Computer Security Definition

Computer security: Properties (defined by the security policy) of a computer system must hold in the presence of a resourced strategic adversary (described by the threat model).

1.2 Properties

- **Confidentiality**: Prevent unauthorized disclosure of information
- **Integrity**: Prevent unauthorized modification of information
- **Availability**: Prevent unauthorized denial of service
- **Authenticity**: Prevent unauthorized usage of other authenticity
- **Non-repudiation**: Prevent denial of actions or message origin

1.3 Security Policy

- **Assets**: Valuable objects (data, files, memory) to protect
- **Principals**: Entities acting on assets (users, programs, services)
- **Policy**: Defines required security properties linking assets and principals
- **Examples**:
  - Confidentiality: only authorized users read
  - Integrity: only authorized programs write
  - Availability: authorized services can access

1.4 Resourced Strategic Adversary

Threat model: Defines adversary's resources and capabilities (observe, influence, corrupt). Adversary always uses optimal strategy.

1.5 Adversary Vocabulary

- **Threat model**: Defines adversary's capabilities, e.g., observe connections, corrupt machine, control employee
- **Vulnerability**: Weakness exploitable by adversary, e.g., API unprotected, password in plain text
- **Threat**: Feared event (goal of adversary), e.g., hacker steals money, student learns password
- **Harm**: Consequence when threat materializes, e.g., money stolen, access blocked, password leaked

1.6 Securing a System

Ensure the security policy holds under the threat model.

- **Security mechanism**: Technical control (software, hardware, crypto, people) preventing policy violation
- **Security argument**: Shows mechanisms are effective under the model (must constrain adversary)
- **Composition**: Defense in depth (ok if ≥ 1 holds), weakest link (fail if one fails)

Asymmetry between attackers and defenders An attacker only needs to find one way to violate one security property within the threat model. A defender must prove that no adversary can violate the security policy. A system is 'secure' if an adversary **constrained** by a **specific threat model** cannot violate the **security policy**.

2 Security Mechanisms

2.1 Eight Base Mechanisms

1. **Economy of mechanisms**: *'Keep the design as simple and small as possible'* - Simple designs reduce the Trusted Computing Base (TCB) and are easier to audit and verify.
2. **Fail-safe defaults**: *'Base access decisions on permission rather than exclusion'* - Default to secure state when failures occur. Use whitelists over blacklists.
3. **Complete mediation**: *'Every access to every object must be checked for authority'* - A reference monitor must mediate all actions from subjects on objects and verify them against current access permissions.
4. **Open design**: *'The design should not be secret'* - Security mechanisms should not depend on the secrecy of their design. Only keys, passwords, or specific noise patterns should be kept secret (Kerckhoff's principle).
5. **Separation of privilege**: *'No single accident, deception, or breach of trust is sufficient to compromise the protected information'* - Require multiple conditions to execute an action (e.g., two-factor authentication, two keys for safe).
6. **Least privilege**: *'Every program and every user should operate using the least set of privileges necessary to complete the job'* - Rights should be added only as needed and discarded after use (need-to-know principle).
7. **Least common mechanism**: *'Minimize the amount of mechanism common to more than one user'* - Every shared mechanism represents a potential information path. Minimize shared mechanisms to prevent unintended information leaks or privilege abuse.
8. **Psychological acceptability**: *'The human interface must be designed for ease of use'* - Users must routinely and automatically apply protection mechanisms correctly. The mental model of users must match the security policy.

3 Access Control

Check that all accesses and actions on objects by principals are within the security policy. First line of defense. Authentication binds an actor to a principal (not seen here). Authorization checks that the principal is authorized.

3.1 Discretionary Access Control (DAC)

Object owners assign permissions (Facebook, Strava, Linux).

3.1.1 Access Control Matrix

Abstract model describing all authorized (subject, object, right) triplets in a system.

Subject/Object	file1	file2	file3
Alice	r,w	-	r
Bob	r,w	r,w	-

Conceptual model, not practical for large systems (sparse, inefficient).

3.1.2 Access Control Lists (ACLs)

- Store permissions with objects
- Each object lists which subjects can access it and with what rights

Example:

- file1: {(Alice, r/w)}
- file2: {(Bob, r/w)}
- file3: {(Alice, r), (Bob, r/w)}

Advantages

- Easy to check who can access a given object
- Easy to revoke access to a specific object

Drawbacks

- Hard to list all accesses of one user
- Hard to remove all rights from a user (must scan all ACLs)
- Delegation and auditing more complex

3.1.3 Capabilities

- Store permissions with subjects
- Each subject lists which objects it can access and how

Example:

- Alice: {(file1, r/w), (file3, r)}
- Bob: {(file2, r/w), (file3, w)}

Advantages

- Easy to audit or delegate (subject carries its capabilities)
- Portable and flexible

Drawbacks

- Hard to revoke one object's rights once shared
- Risk of capability leakage or uncontrolled transfer
- Authenticity must be ensured (non-forgeable tokens)

3.1.4 Role-Based Access Control (RBAC)

- Permissions are assigned to roles, not users directly
- Users get permissions through the roles they hold
- Common in organizations (doctor, admin, student, etc.)

Steps

1. Assign permissions to roles
2. Assign roles to users
3. User activates one or more roles $\rightarrow$ inherits role permissions

Problems

- Role explosion: too many fine-grained roles
- Least privilege: hard to maintain minimal rights
- Separation of duty: ensuring distinct users for critical actions

3.1.5 Group-Based Access Control

- Permissions grouped by access need, subjects grouped by membership
- Subjects inherit rights from all groups they belong to
- Simplifies ACLs and management for similar users

Notes

- Groups $\approx$ coarse-grained roles
- May include negative permissions to restrict exceptions

3.1.6 Ambient Authority & Confused Deputy Problem

- **Ambient authority**: Programs use implicit subject identity (e.g., process owner) $\rightarrow$ program actions automatically use its full privileges

Confused deputy

- Program with authority acts on behalf of a less-privileged user
- User manipulates program to perform unauthorized actions

3.1.7 Linux (UNIX) Access Control

- **Principals**: Users (UID), Groups (GID)
- **Everything is a file**: Each file has an owner, group, and mode bits (r,w,x)
- 3 sets of bits: Owner (u), Group (g), Other (o)

rwx	File meaning	Dir meaning	Example
r	read file	list contents	"ls"
w	modify file	add/delete files	"touch"
x	execute file	enter dir	"cd"

Access order

1. If UID == owner $\rightarrow$ check owner bits
2. Else if GID matches $\rightarrow$ check group bits
3. Else $\rightarrow$ check "other" bits

Special bits

- suid/sgid: run with file owner's privileges (needed for /bin/passwd)
- sticky bit: only the owner can delete in shared directories (/tmp)
- root (UID 0): bypasses checks $\rightarrow$ in Trusted Computing Base (TCB)

Example: ls -l output (Linux)

```
drwxrwxr-x 2 caasi devs 4096 Nov 2 12:10 project/
-rw-r--r-- 1 caasi devs 1200 Nov 2 12:05 report.txt
-rwsr-xr-x 1 root root 27768 Aug 20 2020 /bin/passwd
```

- 1st character: file type (d=directory, -=file)
- Next 9: permissions (owner/group/other)
- s in rws $\rightarrow$ setuid bit (runs as owner)
- t at end $\rightarrow$ sticky bit (only owner can delete in shared dir)

Interpretation example

- report.txt: owner can read/write; group can read; others can read
- project/: owner & group can list/create; others can read & traverse
- /bin/passwd: executable running with root privileges (setuid)

3.2 Mandatory Access Control (MAC)

Central security policy assigns permissions (Military, Hospital, etc.)

3.2.1 Bell-LaPadula (BLP) Model

Focus on confidentiality. Too low level, not expressive, does not ensure confidentiality because of covert channels. Each object has one label and belongs to one or more categories.

- **Label**: Unclassified, confidential, secret, etc.
- **Categories**: Nuclear, army, crypto, etc.

Dominance: Security level (l_1, c_1) dominates (l_2, c_2) iff $l_1 \geq l_2$ and $c_2 \subseteq c_1$.

Three Core Properties

- **ss-property** (Simple Security): 'No Read Up (NRU)'
  Subject can read object only if level(S) dominates level(O)
- ***-property** (Star Property): 'No Write Down (NWD)'
  Subject can write to O_2 only if level(O_2) $\geq$ level(O_1) (prevents info leak to lower levels)
- **ds-property** (Discretionary Security): Need-to-know within same level
  Access (S, O, action) must be authorized in access control matrix: $(S, O, \text{action}) \in M$

Actions Read, write, execute, append

Covert Channels Unintended communication paths violating security policy:

- **Storage channels**: shared resources (file IDs, counters, disk space)
- **Timing channels**: CPU time, cache state, response delay variations
- **Mitigation**: isolation (prevent shared resources), noise injection (randomize timing)

- Complete elimination infeasible; typical reduction to < 1 bit/s (insufficient for crypto keys)

Declassification Controlled lowering of classification level for document release. Risks: covert channels, residual data in metadata (Word revision history, PDF hidden text).

3.2.2 BIBA Model

Focus on integrity. Dual of Bell-LaPadula.

Two Core Properties

- **Simple Integrity Property**: 'No Read Down'
  Subject can read object only if level(S) $\leq$ level(O)
  Prevents high-integrity subjects from being corrupted by untrusted data
- ***-Integrity Property**: 'No Write Up'
  Subject can write object only if level(S) $\geq$ level(O)
  Prevents low-integrity subjects from contaminating trusted data

Actions Read, Write, Invoke

Biba Variants **Low-water-mark for subjects:** Subjects downgraded when reading lower-integrity data

- $\text{current}(S) := \min(\text{current}(S), \text{level}(O))$ when reading
- Temporary sandbox, avoids high-level contamination
- **Risk**: label creep (everything becomes low integrity over time)

Low-water-mark for objects: Objects downgraded when written by lower-integrity subjects

- $\text{level}(O) := \min(\text{level}(O), \text{level}(S))$ when writing
- Detects but does not prevent integrity loss
- **Mitigation**: replicate, sanitize or delete polluted copy

Invocation: Allow controlled cross-level interaction

- **Simple invocation**: level(S_1) $\geq$ level(S_2) (high $\rightarrow$ low)
  Protects high data, unclear output level

- **Controlled invocation**: level(S_2) $\geq$ level(S_1) (low $\rightarrow$ high)
  High acts as gatekeeper, hard to verify integrity flows

3.2.3 Chinese Wall Model

Prevent conflicts of interest.

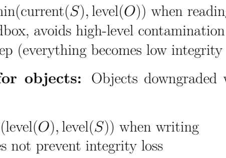


Figure 1: Chinese Wall Model: Analysts who access data from one company in a conflict set cannot access competing companies' data

- Each object has a label of origin (company, client, etc.)
- Conflict sets group competing entities
- Each subject has a history of access
- A subject can access an object (read/write) only if it does not create an information flow between two objects in the same conflict set
- Example: An analyst who accessed Pepsi data cannot later access Coca-Cola data

4 Applied Cryptography

4.1 Core Goal

- **Confidentiality**: Ensure Eve (adversary) cannot read data over insecure channel
- **Plaintext (M)**: Original message
- **Ciphertext (C)**: Encrypted message
- **Key (K)**: Secret controlling encryption/decryption
- **Encryption**: $C = E_K(M)$
- **Decryption**: $M = D_K(C)$
- **Invertibility**: $D_K(E_K(M)) = M$

- **Security requirement**: Without K, decryption must be computationally infeasible

4.2 Hardness & Key Space

- **Brute force**: Try all possible keys
- **Bits of security**: $\log_2(|\text{keyspace}|)$
- Example: Caesar cipher → 25 keys → 4.6 bits → insecure
- Substitution cipher: $26! \approx 4 \times 10^{26}$ keys (≈ 88 bits) → still breakable via frequency analysis (statistical attack)
- **Cryptanalysis**: Exploiting structure or frequency of plaintext to reduce key search space

4.3 Key Terms

- **Encryption algorithm**: Deterministic or randomized transformation parameterized by K
- **Decryption algorithm**: Inverse function using K
- **Key space**: All possible keys, defines theoretical security upper bound
- **Security level**: Effort (2^n operations) required for best known attack

5 Adversary Types

5.1 Passive Eavesdropper

Eve only reads ciphertext → limited model (e.g., substitution cipher).

5.2 Known Plaintext Attack (KPA)

- Eve knows pairs $(M, C = E_K(M))$ → infers key patterns
- Realistic: headers or predictable data leak info (e.g., “From:” field)

5.3 Chosen Plaintext Attack (CPA)

- Eve can choose messages and get encrypted outputs (encryption oracle)
- Stronger than KPA → full break for substitution cipher (choose “abcdefghijklmnopqrstuvwxyz”)

5.4 Side-Channel Attacks

- Exploit physical information during encryption: timing, power use, electromagnetic leaks
- Common on devices holding third-party keys (e.g., smart cards, DRM)

6 One-Time Pad (OTP)

6.1 Principle

- Key = truly random bits, same length as message
- Encryption: $C = M \oplus K$; Decryption: $M = C \oplus K$
- Guarantees perfect secrecy: C gives zero information about M

6.2 Conditions for Perfect Secrecy

Key must be: random, as long as message, used only once.

Reusing key → breaks secrecy ($C_1 \oplus C_2 = M_1 \oplus M_2$).

6.3 Integrity Flaw

- OTP ensures confidentiality only
- Bit-flip in ciphertext flips same bit in plaintext → no integrity protection

6.4 Practical Limitation

- Key distribution problem: hard to securely share large random keys
- OTP ideal but impractical for real-world use

7 Stream Ciphers

7.1 KSG Principle

A stream cipher mimics OTP: it generates a pseudo-random stream $S = \text{KSG}(K, IV)$ from a secret key K and a public IV.

- Both sides recompute the same S to encrypt/decrypt with XOR
- The IV changes for each message so S differs every time, avoiding key reuse
- Knowing IV or the KSG algorithm does not reveal K if the cipher is secure
- Reusing IV or breaking K compromises all messages

7.2 Idea

- Emulate OTP using short key + keystream generator (KSG)
- Inputs: secret key K + public IV → pseudo-random stream S
- Encryption: $C = M \oplus S$; Decryption: same operation

7.3 Properties

- Symmetric key: same K for encryption/decryption
- IV: ensures two messages use different keystreams
- Security: S must be computationally indistinguishable from random

7.4 Pros / Cons

- **Pro**: Fast, low memory, low error propagation
- **Cons**: Low diffusion → bit-level tampering easy (no integrity)
- **Cons**: Vulnerable if keystream repeats (periodicity)

7.5 Attacks & Flaws

- Finite KSG state → eventually periodic stream
- Short period ⇒ pattern repetition ⇒ message recovery
- Linear designs (e.g., LFSR) predictable → broken (A5/1 GSM)

8 Public Key Cryptography

8.1 Diffie–Hellman Key Exchange

- Solve key distribution: allow two parties to agree on a shared secret over an insecure channel
- Public parameters: prime p, generator g
- Alice → picks a, sends $A = g^a \mod p$
- Bob → picks b, sends $B = g^b \mod p$
- Shared secret: $K = g^{ab} \mod p = (B^a = A^b \mod p)$
- Eve knows (A, B, g, p) but cannot recover a, b → Discrete Log Problem

8.2 Security & Limits

- Security = hardness of computing discrete log
- Provides key agreement, not authentication → vulnerable to man-in-the-middle
- Solution: add digital signatures or certificates (CA) to verify identities
- Examples of related systems: RSA (factoring), ECC (elliptic-curve discrete log), post-quantum (lattice)

9 Authenticity

9.1 Public Key Cryptography

- Each user owns public key (PK) and secret key (SK)
- Anyone can encrypt with PK → only SK decrypts
- Enables confidentiality without shared secret
- Needs trusted Public Key Infrastructure (PKI) to bind identities to keys

9.2 Digital Signatures

- Sign: $S = \text{Sign}_{SK}(M)$
- Verify: $\text{Verify}_{PK}(M, S)$ → true if valid
- Ensures authenticity, integrity, and non-repudiation
- Forgery infeasible without SK

9.3 Hash-Based Signing

- Instead of signing M directly, sign $H(M)$ (faster, smaller)
- Required hash properties:
  - Second pre-image resistance: can't find $M' \neq M$ with same hash
  - Collision resistance: can't find any (M, M') with same hash
- Pre-image resistance less critical (M often public)

9.4 Hash Functions

- Input any length → fixed short digest
- Core properties:
  - Pre-image: given H, can't recover M
  - Second pre-image: given M, can't find M' with same H
  - Collision: can't find any pair (M, M') with same H
- Use: SHA-2, SHA-3. Avoid MD5, SHA-1
- Applications: signatures, HMACs, password storage, integrity checks

10 Block Ciphers

10.1 Principle

- Process data in fixed-size blocks (e.g., 128 bits)
- Use same secret key K for encryption and decryption
- Deterministic mapping: $C = E_K(M)$, $M = D_K(C)$
- Example: AES (Advanced Encryption Standard)

10.2 Goal & Limitation

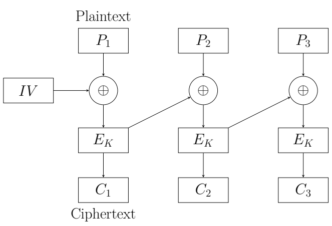
- Acts like a permutation over all possible blocks
- Deterministic → same plaintext block ⇒ same ciphertext ⇒ pattern leaks
- Solution: use modes of operation with randomization (IV)

10.3 Block Cipher Modes

10.3.1 ECB – Electronic Code Book

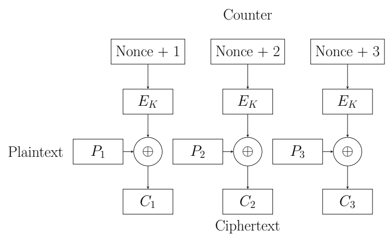
- Encrypt blocks independently: $C_i = E_K(M_i)$
- Leaks identical patterns (e.g., image structure)

10.3.2 CBC – Cipher Block Chaining



- Adds diffusion with XOR chaining
- Hides patterns, needs random IV
- Sequential encryption (not parallelizable)

10.3.3 CTR – Counter Mode



- Turns block cipher into stream cipher
- Parallelizable, random access, requires unique nonce per key
- Same nonce reuse ⇒ catastrophic ($C_1 \oplus C_2 = M_1 \oplus M_2$), like OTP reuse

10.3.4 Summary

- ECB: insecure, reveals structure
- CBC: secure if IV random; sequential
- CTR: fast, parallel, secure if nonce unique

11 Authenticated Encryption & Integrity

11.1 Why Authentication Matters

- Confidentiality alone ≠ security
- Need to detect if ciphertext was modified (tampering, replay)
- Combine encryption + integrity protection

11.2 MAC – Message Authentication Code

- Symmetric integrity check: $\text{Tag} = \text{MAC}_K(M)$
- Verification: recompute $\text{MAC}_K(M)$ and compare
- Detects modification and confirms origin (shared key)
- No third-party proof (no non-repudiation)
- Examples: HMAC-SHA256, CMAC-AES

11.3 Authenticated Encryption (AE)

Single scheme ensuring confidentiality + integrity.

Common designs:

- **Encrypt-then-MAC**: Encrypt first, then authenticate ciphertext → secure (modification detected before decryption)
- **Encrypt-and-MAC**: Process done independently → weak, integrity not linked to ciphertext; some attacks possible if verification skipped
- **MAC-then-Encrypt**: Encrypts both message and tag → weak, attacker can modify ciphertext and mislead error handling (e.g., old TLS)

11.4 Why Encrypt-then-MAC is Best

- Verify integrity **before** decryption → reject tampered ciphertext early
- Prevents padding oracle attacks, chosen-ciphertext attacks
- Generic composition works with any secure encryption + MAC
- Modern modes (GCM, CCM, ChaCha20-Poly1305) implement this principle

12 Public Key Infrastructure (PKI)

12.1 The Key Distribution Problem

- Public key cryptography enables encryption/signatures without shared secrets
- **Problem**: How to trust that a public key belongs to the claimed identity?
- Example: Alice receives PK_B claiming to be Bob's key. How does she verify?
- Without verification → Man-in-the-Middle attacks possible

12.2 Certificates & Certificate Authorities (CA)

Certificate: digitally signed statement binding identity to public key

Certificate structure

- Subject name (e.g., 'alice@example.com' or 'www.bank.com')
- Subject's public key (PK)
- Validity period (not before / not after dates)
- Issuer (CA) name
- CA's digital signature: $\text{Sign}_{SK_{CA}}(H(\text{certificate data}))$

Certificate Authority (CA)

- Trusted third party that verifies identities and issues certificates
- CA's public key (PK_{CA}) is widely known and trusted
- Anyone can verify certificate: $\text{Verify}_{PK_{CA}}(\text{cert}, \text{signature})$

12.3 Certificate Chains & Trust Hierarchy

- **Root CA**: Top-level CA, self-signed certificate
- **Intermediate CA**: Issued by Root CA, issues end-entity certificates
- **End-entity certificate**: Issued to users/servers/devices
- **Chain verification**: End-entity ← Intermediate ← Root
- Browsers/OS pre-install trusted root CA certificates

12.4 PKI Security Properties

- **Authenticity**: Certificates bind verified identities to public keys
- **Integrity**: Signatures prevent certificate tampering
- **Trust anchor**: Security relies on protecting root CA private keys
- **Weakest link**: Compromise of any CA in chain breaks trust

13 Password Security

13.1 Storing Passwords

Correct approach: Hash + Salt

- Generate random salt s for each user
- Store $(s, H(\text{password}\|s))$
- Verification: recompute $H(\text{input}\|s)$ and compare
- **Salt prevents rainbow tables**: each user has different hash even with same password
- Salt can be public (stored with hash)

13.2 Password Attacks

- **Dictionary attack**: Try common passwords (123456, password, qwerty)
- **Brute-force**: Try all possible combinations (slow with bcrypt/Argon2)
- **Credential stuffing**: Reuse leaked passwords from other breaches
- **Phishing**: Social engineering to steal credentials directly
- **Timing attacks**: Measure comparison time to leak password length

13.3 Password Strength

- **Entropy**: $\log_2$(possible passwords)
- Example: 8 random chars (a-z, A-Z, 0-9, symbols) ≈ 52 bits
- Trade-off: entropy vs memorability
- **Passphrases**: 'correct horse battery staple' → high entropy, memorable
- **Best practice**: password managers generate & store random passwords

14 Man-in-the-Middle (MITM) Attack

14.1 Diffie-Hellman Vulnerability

Key observation: DH provides key agreement but NOT authentication

The Attack

1. Alice sends $A = g^a \mod p$ to Bob
2. **Eve intercepts**, sends $E_1 = g^{e_1} \mod p$ to Bob (pretending to be Alice)
3. Bob sends $B = g^b \mod p$ to Alice
4. **Eve intercepts**, sends $E_2 = g^{e_2} \mod p$ to Alice (pretending to be Bob)
5. Alice computes shared key with Eve: $K_{AE} = E_2^a = g^{ae_2} \mod p$
6. Bob computes shared key with Eve: $K_{BE} = E_1^b = g^{be_1} \mod p$
7. Eve controls all traffic: Alice $\xleftrightarrow{K_{AE}}$ Eve $\xleftrightarrow{K_{BE}}$ Bob

14.2 Consequences

- Eve decrypts all messages from Alice, re-encrypts for Bob (and vice versa)
- Neither Alice nor Bob detect the attack
- Eve can read, modify, or block any message
- **Complete confidentiality breach despite using DH!**