Security and Privacy
Summary

Isaac Metthez

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, 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

- current(S) := min(current(S), 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

- level(O) := min(level(O), 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.

- 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

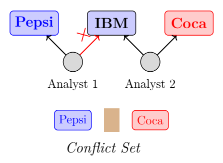


Figure 1: Chinese Wall Model: Analysts who access data from one company in a conflict set cannot access competing companies' data

- 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 $\rightarrow$ 25 keys $\rightarrow$ 4.6 bits $\rightarrow$ insecure
- Substitution cipher: $26! \approx 4 \times 10^{26}$ keys ($\approx$ 88 bits) $\rightarrow$ 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

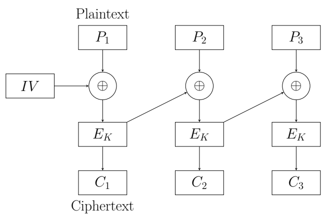
- Acts like a permutation over all possible blocks
- Deterministic → same plaintext block ⇒ same ciphertext ⇒ pattern leaks
- Solution: use modes of operation with randomization (IV)

10.3 Block Cipher Modes

10.3.1 ECB – Electronic Code Book

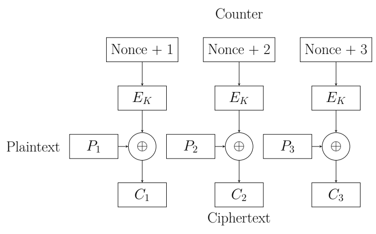
- Encrypt blocks independently: $C_i = E_K(M_i)$
- Leaks identical patterns (e.g., image structure)

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: 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!**

15 Authentication (Detailed)

15.1 Authentication Factors

- **What you know**: Passwords, PINs, secret keys
- **What you are**: Biometrics (fingerprint, face, iris)

- **What you have**: Tokens, smart cards, phones
- **Where you are**: Location, IP address

15.2 Secure Password Transfer

- **Problem**: Passwords intercepted over network
- **Solution**: TLS/HTTPS (combines DH + signatures + hybrid encryption)
- **Replay attack**: Eve captures encrypted message, replays it later

15.3 Challenge-Response Protocol

1. Alice: "I want to login"
2. Server: generates random challenge R (nonce)
3. Alice: sends $E_k(\text{password}, R)$
4. Server: verifies, deletes R

Why it works: Each R used only once → captured responses useless

15.4 Password Storage

Salted Hashes (Correct)

- Generate random salt s per user
- Store $(s, H(\text{password}\|s))$
- **Salt prevents rainbow tables**: same password → different hashes
- Salt can be public
- Use slow hash: bcrypt, Argon2, PBKDF2

Dictionary Attacks

- Attacker tries common passwords: "123456", "password"
- Precomputed rainbow tables accelerate attack
- **Defense**: Slow hash functions, rate limiting

15.5 Biometric Authentication

- **FAR** (False Accept Rate): Adversary accepted
- **FRR** (False Reject Rate): Legitimate user rejected
- Trade-off: Lower FAR ⇒ Higher FRR
- **Problems**: Cannot revoke, privacy concerns, probabilistic

15.6 Time-Based One-Time Passwords (TOTP)

- Generate time-varying codes: $v_n = \text{HMAC}_{\text{seed}}(\lfloor T/X \rfloor)$
- New code every 30 seconds (X = 30)
- **Must use HMAC**: Hash alone allows adversary to predict $v_{n+1} = H(v_n)$
- HOTP: Counter-based variant

15.7 Two-Factor Authentication (2FA)

- Requires two different factor types
- **Examples**: Password + SMS, Password + App code, Card + PIN
- **Benefits**: Compromising one factor insufficient
- **SMS risks**: SIM swapping, interception
- **App-based**: More secure (TOTP locally generated)

16 Network Security

16.1 Four Network Security Properties

1. **Naming**: IP-domain bindings not influenced by adversary
2. **Routing**: Message delivery not influenced by adversary
3. **Session**: Message ordering/integrity maintained
4. **Content**: Messages confidential and unmodified

16.2 ARP Spoofing

Background

- ARP translates IP → MAC addresses on LAN
- No authentication, accepts unsolicited replies

Attack

- Send fake ARP reply: "Bob's IP = Attacker's MAC"
- Victim sends traffic to attacker
- **Man-in-the-Middle**: Spoof both Alice and Bob
- **DoS**: Provide invalid MAC address

Defense

- Static ARP entries (doesn't scale)
- ARP spoofing detection software (ArpWatch, XArp)
- Monitor for IP-MAC conflicts

16.3 DNS Spoofing

DNS Cache Poisoning

- Attacker floods resolver with fake responses
- If fake response arrives before real one → cached
- Must guess transaction ID (16 bits ≈ 65k possibilities)
- **Kaminsky attack**: Query many subdomains to increase chances

DNSSEC Defense

- Digitally sign DNS records
- Chain of trust from root to leaf
- Prevents cache poisoning
- **Limitation**: Adoption still incomplete

16.4 BGP Hijacking

- BGP announces routes without authentication
- Attacker announces shorter/more specific route
- Traffic routed through attacker
- **Defense**: BGPsec (cryptographic verification), route filtering

16.5 TCP Session Hijacking

Attack

- Predict sequence numbers
- Inject packets into existing connection
- Take over session

Defense

- Random initial sequence numbers
- Use TLS (encrypts + authenticates)

16.6 TLS (Transport Layer Security)

- Provides: authentication, confidentiality, integrity
- **Handshake**: DH key exchange + certificate verification
- **Record protocol**: Encrypt-then-MAC data transfer
- Defends against: eavesdropping, tampering, MITM

16.7 Denial of Service (DoS)

SYN Flood

- Send many TCP SYN packets (connection requests)
- Server allocates state, waits for ACK (never comes)
- Server memory exhausted
- **Defense**: SYN cookies (stateless until ACK received)

DDoS

- Distributed attack from many sources (botnet)
- Overwhelms bandwidth or server capacity
- **Defense**: Traffic filtering, CDN, over-provisioning

16.8 Firewalls

Types

- **Stateless (packet filter)**: Inspect each packet independently
- **Stateful**: Track connection state (TCP/UDP sessions)
- **Application (Deep Packet Inspection)**: Inspect content

Limitations

- Cannot authenticate principals
- Cannot filter encrypted traffic (without decryption)
- **Not a substitute** for host security
- Filter "definitely bad" traffic, not "allow only good"

DMZ (De-Militarized Zone)

- Three zones: WAN → DMZ → LAN
- Public services in DMZ
- Inner firewall protects LAN
- **Defense in depth**: Compromise of DMZ doesn't expose LAN

17 Web Security

17.1 HTTP Basics

- **Stateless**: Each request independent
- **GET**: Retrieve data (parameters in URL)
- **POST**: Send data (parameters in body)
- **Cookies**: Store session state
- **Ambient authority**: Cookies automatically included in requests

17.2 Same Origin Policy (SOP)

- Origin = (protocol, host, port)
- Scripts can only access data from same origin
- **Does NOT prevent CSRF**: Cookies sent regardless of origin
- SOP only prevents *reading* cross-origin responses

17.3 SQL Injection

Vulnerability

```
$query = "SELECT * FROM users WHERE name='" . $_GET['name'] . "'";
```

Input: `'  OR '1'='1`
Result: `SELECT * FROM users WHERE name=" OR '1'='1'`

Defense

- **Parameterized queries**: `prepare("SELECT * FROM users WHERE name=?")`
- Never concatenate user input into SQL
- Input validation (whitelist allowed characters)

17.4 Cross-Site Scripting (XSS)

Reflected XSS

- User input echoed directly in response
- Example: `search.php?q=<script>steal_cookies()</script>`
- Script executes in victim's browser

Stored XSS

- Malicious script stored in database (e.g., comment)
- Served to all users viewing page
- More dangerous than reflected

Defense

- **Output encoding**: Escape HTML special chars (<, >, ")
- **Content Security Policy (CSP)**: Restrict script sources
- **HTTPOnly cookies**: Prevent JavaScript access

17.5 Cross-Site Request Forgery (CSRF)

Attack

1. Victim logged into `bank.com`
2. Victim visits `evil.com`
3. `evil.com` triggers POST to `bank.com/transfer`
4. Browser includes `bank.com` cookies
5. Transfer executes with victim's authority

Defense

- **CSRF tokens**: Include unpredictable token in forms
- **SameSite cookies**: `SameSite=Strict/Lax`
- Check `Referer`/`Origin` header
- Re-authenticate for critical actions

17.6 Command Injection

- User input passed to shell command
- Example: `system("ping " .  $_GET['ip'])`
- Input: `8.8.8.8; rm -rf /`
- **Defense**: Never pass user input to shell, use APIs directly

18 Software Security

18.1 Memory Layout

Segment	Contents
Stack	Local vars, return addresses (LIFO)
Heap	Dynamic allocation (malloc)
BSS	Uninitialized globals
Data	Initialized globals
Text	Code (read-only)

18.2 Buffer Overflow

Stack Buffer Overflow

```
void vulnerable(char *input) {
    char buffer[64];
    strcpy(buffer, input);  // No bounds check!
}
```

Exploit: Input > 64 bytes overwrites return address

Consequences

- Overwrite return address → control execution
- Overwrite function pointers
- Inject shellcode

18.3 Format String Vulnerability

```
printf(user_input);  // WRONG
```

Input: `%x %x %x` → leaks stack contents
Input: `%n` → writes to memory

Defense

```
printf("%s", user_input);  // CORRECT
```

18.4 Use-After-Free

```
char *p = malloc(10);
free(p);
*p = 'A';  // ERROR: memory freed
```

Exploit: Allocate new object, control freed object's contents

18.5 Mitigations

DEP (Data Execution Prevention) / W⊕X

- Memory either writable OR executable, not both
- Prevents code injection
- **Bypass**: Return-Oriented Programming (ROP)

ASLR (Address Space Layout Randomization)

- Randomize stack/heap/library addresses
- Attacker cannot predict addresses
- **Bypass**: Information leaks reveal addresses

Stack Canaries

- Place random value before return address
- Check canary before function return
- Detects buffer overflow
- **Bypass**: Leak canary value, overwrite with same value

18.6 Fuzzing

Goal

- Automatically generate test inputs
- Find crashes/bugs

Coverage-Guided Fuzzing

1. Execute program with input
2. Measure code coverage
3. If new coverage → save to corpus
4. Mutate corpus inputs
5. Repeat

Tools: AFL, LibFuzzer, Honggfuzz

18.7 Sanitizers

AddressSanitizer (ASan)

- Detects: buffer overflows, use-after-free, double-free
- Places red zones around objects
- 2x slowdown
- Compile: `-fsanitize=address`

UndefinedBehaviorSanitizer (UBSan)

- Detects: integer overflow, null pointer, division by zero
- Only sanitizer usable in production
- Compile: `-fsanitize=undefined`

19 Privacy

19.1 Definitions

- **Freedom from intrusion**: "Right to be let alone"
- **Autonomy**: Freedom from unreasonable constraints on identity construction
- **Control**: Informational self-determination
- Privacy ≠ Secrecy (context-dependent)

19.2 Privacy IS a Security Property

- **Individuals**: Protection against profiling, manipulation, identity theft
- **Companies**: Trade secrets, business strategy, competitive intelligence
- **Governments**: National secrets, law enforcement, diplomatic activities
- **Shared infrastructure**: Denying privacy to some = denying to all

Privacy vs Security False Dichotomy Common misconception that privacy and security are opposed. Reality:

- Surveillance is ineffective (sophisticated adversaries evade it)
- Surveillance tools can be abused (NSA, Spanish ministry)
- Surveillance tools can be subverted (Greek Vodafone 2004-2005: backdoors exploited to monitor 106 people)

19.3 Three Categories of Privacy Enhancing Technologies (PETs)

19.3.1 Category 1: Social Circle Adversary

- **Concerns**: "My boss knows I'm job hunting", "My parents saw my pictures"

- **Goals**: Don't surprise user (contextual feedback, privacy nudges, easy defaults)
- **Limitations**: Trusted service provider required, user expectations-based
- **Industry approach**: Facebook, Twitter, LinkedIn (make users comfortable)

19.3.2 Category 2: Institutional Privacy (GDPR)

- **Concerns**: Data collected without consent, illegitimate processing
- **Goals**: Compliance with data protection principles
  - Informed consent, purpose limitation, data minimization
  - Subject access rights, security, auditability
- **Technical measures**: Access control, logging, anonymization (limited!)
- **Limitations**: Trusted provider, never questions collection necessity
- **Anonymization myth**: No magic solution for perfect anonymization with full utility

19.3.3 Category 3: Anti-Surveillance (Network-Level)

- **Concerns**: Infrastructure-level disclosure, censorship, surveillance
- **Goals**: Minimize default disclosure, minimize trust requirements
- **Limitations**: Narrow designs, usability problems, no industry incentives

19.4 Metadata & Traffic Analysis

Key Insight: 'Metadata absolutely tells you everything. If you have enough metadata, you don't really need content.' – Stewart Baker (NSA)

Traffic Analysis Deducing information from communication patterns (not content):

- Identities, timing, frequency, duration, location, volume, device
- **Military origins**: WWI (locate submarines), WWII (assess forces)
- **Modern**: Tempora, MUSCULAR, XKeyscore focus on metadata

Network Protocol Headers Even with encryption, headers reveal:

- IPv4: Source/destination IP, packet length, TTL, protocol
- Same for Ethernet, TCP, SMTP, IRC, HTTP
- **Address leakage**: Storage location reveals content (medical DB example)
- **Location leakage**: Sending from oncology clinic reveals sender info

Browser Fingerprinting

- Screen resolution, fonts, timezone, user agent, plugins, canvas/WebGL
- **AmIUnique.org**: Most users uniquely identifiable without cookies

19.5 End-to-End Encryption

- **True E2E**: Only sender/receiver decrypt (Signal, WhatsApp)
- **Not E2E**: Provider can decrypt (Gmail, Outlook)
- Provides confidentiality, integrity, authenticity, forward secrecy
- **Limitation**: Protects content, NOT metadata

19.6 Anonymous Communications

19.6.1 Use Cases

Journalists, whistleblowers, activists, executives, military, abuse victims, ordinary users avoiding tracking

19.6.2 Abstract Model

- **Adversary**: ISPs, sysadmins, intelligence agencies, network operators
- **Protect**: Sender/receiver IDs, timing, volume, frequency, relationships
- **Single proxy problems**: Low throughput, single failure point, coercion (Penet.fi 1996)
- **Solutions**: Bitwise unlinkability (crypto), (re)packetizing, (re)scheduling, (re)routing, load balancing

19.7 Tor (The Onion Router)

How It Works

1. Select path: 3 relays (entry guard, middle, exit)
2. Prepare circuit: Authenticated DH with each relay
3. Send stream: Encrypt in layers, each relay removes one

Properties

- Entry sees client, exit sees destination, middle sees neither
- Overlay network at application layer (not internet routers)
- Traffic through regular internet between relays

Limitations

- **Assumes**: Adversary cannot observe both circuit ends
- **Global passive adversary**: Can correlate entry/exit traffic
- **Exit relay monitoring**: Sees unencrypted traffic, can inject/modify
- **Circuit compromise**: Controlling entry+exit enables correlation
- Low latency prioritized over maximum anonymity

VPN

- **Protects**: Confidentiality from ISP, IP hiding, geo-restrictions
- **Does NOT protect**: Anonymity from VPN, traffic analysis, legal jurisdiction
- **Key**: Centralized trust – VPN sees everything

19.8 Application-Layer Anonymity

Network anonymity insufficient – application behavior reveals identity (logins, emails, cookies, fingerprinting)

19.8.1 Anonymous Credentials (Attribute-Based)

Prove attributes without revealing identity: 'I'm subscribed to CNN' not 'I am user X'

PKI (Traditional)	Anonymous Credentials
Signed by trusted issuer	Signed by trusted issuer
Certification of attributes	Certification of attributes
Authentication via secret key	Authentication via secret key
No data minimization	Data minimization
Users identifiable	Users anonymous
Linkable across contexts	Unlinkable across contexts

Cryptographic Guarantees Server cannot: (1) identify user, (2) learn beyond disclosed attributes, (3) distinguish users with same attributes, (4) link multiple uses

Technical : Zero-knowledge proofs, blind signatures, commitment schemes

19.9 Other PETs

- **Private Set Intersection (PSI)**: Compute intersection without revealing sets (private search)
- **Blind Signatures**: Server signs without seeing message (eCash)
- **Secure Multiparty Computation (MPC)**: Joint computation keeping inputs private (hospital statistics)
- **Private Information Retrieval (PIR)**: Query database without revealing query

19.10 Privacy Quantification

No Free Lunch Theorem (Kifer & Machanavajjhala, 2011) For every algorithm with even a sliver of utility, there exists some adversary with prior knowledge such that privacy is NOT guaranteed.

Implications

- Perfect privacy + full utility = impossible
- Privacy guarantees depend on adversary model & prior knowledge
- Data minimization most effective protection
- All techniques involve utility trade-offs

19.11 Key Principles

1. Privacy IS a security property (individuals, companies, governments)
2. Privacy $\neq$ Security trade-off (false dichotomy)
3. Metadata = content (implicit = explicit data)
4. Different adversaries $\rightarrow$ different PETs
5. No free lunch (privacy $\leftrightarrow$ utility)
6. Layered protection (network + application)

20 Malware

Previous Attacks vs. Malware

- **Previous**: Expert adversary, manual coding/testing, deep understanding required
- **Malware**: Can be exploited by non-experts, automated, scales easily

20.1 Definition & Statistics

Malware = Malicious Software

- Software fulfilling author's malicious intent
- Intentionally written to cause adverse effects
- Malware $\neq$ Virus (virus is a KIND of malware)

20.2 Why the Rise?

- **Homogeneous computing**: Windows/Android = tempting targets
- **Clueless users**: Many vulnerable targets
- **Unprecedented connectivity**: Remote/distributed attacks easier
- **Profitable**: Compromised computers sold, used for money/Bitcoin
- **Attacker engineering**: Exploit new capabilities & less prepared entities

20.3 Taxonomy

		Need host	Self-contained
Spread	Self	Virus	Worm
	Non	-	Trojan, Rootkit, Spyware

Modern malware: Combines 'the best' of categories to achieve purpose

20.4 Virus

Definition

- Infects programs to monitor/steal/destroy
- Modifies programs to include (possibly modified) copy of itself
- **Cannot survive without host**
- Permissions = host's permissions (confused deputy!)
- OS/hardware specific

Replication Spreads when host spreads (network/hardware): email, web, USB

Infection Types

- **File**: Overwrite (substitute) or Parasitic (append/modify)
- **Macro**: Overwrite macro in MS Office (Excel, Word) - needs exploit
- **Boot**: Most difficult & dangerous - infect boot partition

Defenses

- **Antivirus Software**:
  - Signature-based (byte/instruction sequences, hashes of known malware)
  - Heuristics (anomalies: incorrect headers, suspicious sections)
  - Behavioral signatures (detect series of changes)
- **Sandboxing**: Run untrusted apps in restricted environment (VM)

20.5 Worm

Definition

- Self-replicating program using network to spread
- **Does NOT need host program**
- Autonomous spread
- Email (needs human interaction) vs Network (automated)

Spreading Methods

- Email harvesting (address book, inbox, browser cache)
- Network enumeration, scanning (random/targeted)

Defenses Host-level:

- Protect from remote exploitation (stack protection, diversity)
- Antivirus (email-based worms)

Network-level:

- Limit outgoing connections
- Personal firewall (block unknown SMTP)
- Intrusion Detection Systems (IDS)
- Heterogeneous systems (different OS/programs)

20.6 Intrusion Detection Systems (IDS)

	Host-based	Network-based
Location	Process on host	Network appliance
Detects	Local malware	All traffic

	Signature-based	Anomaly-based
Method	Known patterns	Behavior different from legitimate
Pro	Low false alarms	Adapts to new attacks
Con	Needs updates, can't find new	High false alarms

20.7 Trojan Horse

Definition

- Appears to perform desirable function BUT performs undisclosed malicious activities
- Requires users to **explicitly** run program
- **Cannot replicate**

Activities

- Spy on sensitive data (spyware, keylogger)
- Allow remote access (backdoor)
- Base for further attacks (mail relay for spam)
- Damage routines (corrupt files)

Defenses Sign programs? Train users? **Least privilege principle!**

20.8 Rootkit

Definition

- Adversary-controlled code deep within TCB
- Hides presence by modifying OS
- Installed after system compromise
- **Difficult to detect**

Capabilities

- Replace system programs with trojaned versions
- Modify kernel data structures (hide processes/files/network)
- Allow adversary to return later

Defense Integrity checkers (user/kernel level) - difficult!

20.9 Backdoor

Definition: Hidden functionality bypassing security mechanism

Trust Problem

- Can audit program source
- But what if compiler is malicious?
- Chain of reasoning suspect ALL programs down to first compiler!
- 'Reflections on Trusting Trust' (Thompson, 1984)

20.10 Botnets

Definition Millions of compromised hosts ('zombies'/'bots') under control of single entity

Command & Control (C&C): System to track bots and send commands

Key: Attacks at scale!

20.11 Botnet Topologies

20.11.1 Star Topology

- Central C&C server
- All bots connect to C&C
- **Problem**: C&C = single point of failure
- Violates **least common mechanism** principle

20.11.2 P2P Topology

- No central C&C
- Bots connect to each other
- **Problems**:
  - Difficult management (join/leave?)
  - Vulnerable to Sybil attacks (too many bots taken over)

20.11.3 Hybrid

Combines star + P2P: bots connect to P2P network of servers

20.12 Monetizing Botnets

- **Rental**: 'Pay to use my botnet'
- **DDoS extortion**: 'Pay or I take down your business'
- **Bulk traffic**: 'Pay to boost website visits'
- **Click fraud**: Simulate ad clicks for revenue
- **Ransomware distribution**: 'I encrypted your drive'
- **Spam/advertising**: Leave comments across web
- **Bitcoin mining**

20.13 Botnet Defenses

- **Attack C&C infrastructure**:
  - Take communication channel offline
  - Hijack/poison DNS (route to black hole)
- **Honeypots**: Vulnerable system to attract attackers, study behavior in controlled environment

20.14 Key Takeaways

1. Malware = intentionally malicious software
2. Non-experts can exploit (unlike previous attacks)
3. Many types: virus (needs host), worm (self-replicating), trojan (disguised), rootkit (OS-level), backdoor (hidden access)
4. Modern malware combines multiple types
5. Defenses: antivirus (signatures/heuristics/behavioral), IDS (host/network, signature/anomaly), sandboxing, integrity checkers
6. Botnets = attacks at scale (millions of compromised hosts)
7. Topologies: Star (single C&C, single point of failure), P2P (no C&C, Sybil vulnerable), Hybrid (both)
8. Profitable: rental, DDoS, click fraud, ransomware, Bitcoin mining