Security Toolbox
+ Secure Programming

CST2412 — Data Security / Privacy / Ethics
(Ch. 2: Authentication, Access Control, Cryptography • Ch. 3: Programs & Programming)

Agenda

- Why “toolbox” + “programming” belong together
- Ch 2: Identification & authentication
- Ch 2: Access control (authorization) + auditing
- Ch 2: Cryptography — what it can and can’t do
- Ch 3: Code as a security boundary + vulnerability classes
- In-class lab: design access control + safe credential storage
- In-class writing: security vs privacy vs ethics tradeoff argument

Instructor pacing ~35–40 min lecture • ~15–20 min lab • ~10–15 min writing

Security, Privacy, Ethics: three questions

Security

- What can go wrong?
- How likely is it?
- How bad is it?

Privacy & Ethics

- Who is exposed by the fix?
- Who is excluded by the fix?
- Who gets power from the fix?

Today's habit Every mechanism we learn → ask what harms it prevents AND what harms it can cause.

CHAPTER 2

The Toolbox

Mechanisms: identity • authorization • cryptography
...but mechanisms only work when they match threats.

Identity words people mix up

- Identification: “I claim I’m Alex.” (a label)
- Authentication: “Prove you are Alex.” (evidence)
- Authorization: “Alex is allowed to do X.” (policy + decision)
- Accounting/Auditing: “Record what Alex did.” (logs)

Quick check Logging is not access control. It’s accountability after the fact.

Authentication factors (and why MFA helps)

- Something you know (password, PIN)
- Something you have (phone, security key)
- Something you are (biometrics)
- MFA works because attackers rarely steal all factors at once

Ethics note Biometrics raise consent, permanence, and misuse risks.

Passwords: what makes them weak in practice

- Humans reuse passwords across sites
- Attackers can do credential stuffing at scale
- Phishing steals passwords without “cracking” them
- Databases leak → offline guessing becomes possible
- “Security questions” are often guessable social data

Key shift Assume passwords will leak; design as if compromise is normal.

What "good" password handling looks like

Do

- Hash, don't encrypt passwords
- Use slow hashing (bcrypt/Argon2)
- Unique salt per password
- Rate-limit login attempts
- Offer MFA / passkeys

Don't

- Store plaintext passwords
- Use fast hashes (SHA-256) alone
- Reuse a single salt
- Log passwords or reset tokens
- Build your own crypto

Privacy note Logs are where secrets go to die. Treat logging as a data-ethics surface.

Authorization: policy vs mechanism

- Policy: the rules (who can do what, under what conditions)
- Mechanism: the implementation (ACLs, RBAC tables, middleware)
- A secure system needs both — and they must match

Common failure “We have roles” (policy) but no consistent enforcement (mechanism).

Access control models (choose intentionally)

- DAC (Discretionary): owners share at their discretion (common in file sharing)
- MAC (Mandatory): central labels/clearances (military-style)
- RBAC: roles → permissions (common in orgs/apps)
- ABAC: attributes → decisions (context-aware: time, location, purpose)

Ethics note ABAC can become discrimination if attributes encode bias or proxies.

Access control matrix (the mental model)

- Rows: subjects (users/roles/services)
- Columns: objects (data/resources)
- Cells: allowed operations (read/write/delete/admin)
- Implementations are usually ACLs or capabilities

Practice In the lab, you will literally build one of these for a scenario.

Access Control Lists vs capabilities (tradeoffs)

ACLs (object-centered)

- Easy to answer: who can access this file?
- Centralized management per object
- Revocation can be straightforward

Capabilities (subject-centered)

- Easy to delegate a token/handle
- Works well in distributed systems
- Revocation can be tricky

Privacy hook Delegation is power. Track how permissions propagate.

Principles that keep access control sane

- Least privilege: only the permissions you need, only when you need them
- Separation of duties: split sensitive actions across people/steps
- Fail-safe defaults: deny by default, allow explicitly
- Complete mediation: check every access, not just the first time

Design lens These are as much ethics as security: they limit power concentration.

Auditing: accountability vs surveillance

- Why we log: incident response, forensics, fraud detection
- What we risk: surveillance, chilling effects, secondary use
- Minimize: log what you need, redact, retain briefly, restrict access

Rule of thumb If you wouldn’t show it to the user, question whether you should collect it.

CHAPTER 2

Cryptography

Confidentiality • Integrity • Authenticity
Crypto is powerful — but it does not fix broken authorization.

Three crypto jobs (memorize these)

- Confidentiality: keep data secret (encryption)
- Integrity: detect tampering (hash/MAC)
- Authenticity / non-repudiation: prove who created something (signatures)

Check Encryption ≠ integrity. A ciphertext can be modified unless integrity is included.

Symmetric vs asymmetric (when to use what)

Symmetric (one shared key)

- Fast
- Great for bulk data
- Key distribution is the hard part

Asymmetric (public/private keys)

- Solves key exchange at scale
- Enables signatures
- Slower; often used to establish symmetric keys

Reality Most systems use both: public key to set up, symmetric to communicate.

Hashes, MACs, and signatures (don’t blur them)

- Hash: fingerprint of data (no key) — detects accidental or malicious changes (with context)
- MAC: keyed integrity — proves “someone with the key wrote this”
- Signature: private-key proof — verifies identity with public key

Common misuse A plain hash does not prove who sent the message.

Key management is the real cryptography

- Keys must be generated securely (good randomness)
- Keys must be stored securely (HSM/KMS/secrets manager)
- Keys must rotate and be revocable
- Access to keys is an access control problem

Uncomfortable truth If attackers get the keys, encryption becomes decoration.

Crypto + privacy: what it helps, what it can’t solve

- Helps: protect data in transit and at rest from outsiders
- Doesn’t solve: misuse by insiders or authorized systems
- Doesn’t solve: collection/retention harms (you can encrypt too much data)
- Ethical prompt: who controls decryption and under what governance?

Takeaway Privacy is about limits; encryption is about protection within those limits.

CHAPTER 3

Programs & Programming

Why code is a security boundary
...and why “normal bugs” become security incidents.

Code is where trust meets reality

- Every program enforces assumptions
- Attackers search for assumption violations
- Trust boundaries: where data crosses from untrusted → trusted
- Security programming is about validating, constraining, and checking

Mantra Be kind to users; be strict with inputs.

Vulnerability class: Injection

- Untrusted input changes the meaning of a command or query
- Classic: SQL injection, shell injection, template injection
- Defense: parameterized queries, strict APIs, least-privileged DB accounts

Check Sanitizing strings is fragile. Parameterize instead.

Vulnerability class: Broken access control

- The system fails to enforce authorization rules consistently
- Common: IDOR (insecure direct object references), missing checks, privilege escalation
- Defense: server-side checks everywhere, deny-by-default, test authorization paths

Ethics note Access control failures often become privacy incidents (exposed records).

Vulnerability class: XSS (and output encoding)

- Attacker-controlled content runs as code in a victim's browser
- Often from unescaped HTML or unsafe templating
- Defense: output encoding, safe templating, Content Security Policy (CSP)

Security meets ethics XSS can become account takeover → impersonation → targeted harm.

Vulnerability class: Memory safety failures

- Buffer overflows, use-after-free, out-of-bounds reads/writes
- Often language/runtime dependent (C/C++ risk, but not only)
- Defense: safer languages, compiler hardening, fuzzing, bounds checks

Teaching note We focus on prevention patterns—not exploit steps.

Vulnerability class: Insecure deserialization / parsing

- Untrusted data becomes objects/commands with side effects
- Risky when “constructors” execute logic implicitly
- Defense: safe formats, strict schemas, allowlists, avoid eval-like behavior

Privacy note Parsers often touch sensitive data—crashes can leak via logs.

Vulnerability class: Secrets handling

- API keys in repos, tokens in logs, credentials in screenshots
- Defense: secrets managers, short-lived tokens, redaction, least privilege
- Ethical angle: incidents often expose user data + developer data

Reality check If a secret can be copied, it will eventually be copied.

Secure design principles (classic set)

- Economy of mechanism: keep it simple
- Open design: don’t rely on secrecy of design
- Least privilege: minimize authority
- Complete mediation: check every access
- Psychological acceptability: usable security people will actually do

Ethics tie-in Usability is not fluff; unusable security shifts burden onto people.

Secure coding: patterns that scale

Boundary patterns

- Validate input at trust boundaries
- Use allowlists over denylists
- Encode output by context
- Parameterize queries/commands

Process patterns

- Code review + threat modeling
- Static analysis + dependency scanning
- Fuzzing for parsers & edge cases
- Security tests for authorization

Takeaway Security is a system: code + process + culture.

Connecting Ch 2 → Ch 3

- Authn without authz → anyone who logs in can see everything
- Authz without complete mediation → bypass through a forgotten endpoint
- Crypto without key governance → “encrypted” data leaks anyway
- Logs without privacy design → security tooling becomes surveillance

One sentence Controls fail at boundaries — human boundaries and code boundaries.

IN-CLASS LAB

Design + Implement (Defensive)

Part A: access control policy
Part B: safe password storage
Part C: privacy-aware logging choices

Lab A (10–12 min): Access control design

- Scenario: Student Support Portal (SSP) stores sensitive notes and referrals
- Roles: student, counselor, instructor, admin, auditor
- Objects: student_profile, referral, session_notes, appointment, audit_log
- Task: build an access control matrix + choose RBAC or ABAC
- Deliverable: Describe The System

Privacy constraint Counselor notes are highly sensitive; instructor access is tightly limited.

Lab A checklist (use this to sanity-check)

- Least privilege: does anyone have broad read “just because”?
- Separation of duties: who approves exports or role changes?
- Complete mediation: where will checks live in code?
- Audit design: what gets logged, who can see logs, retention period
- Equity: who gets locked out by your design? (device, time, process)

Instructor tip If your policy is hard to describe, it will be harder to implement securely.

IN-CLASS WRITING

Mini-argument (10–12 min)

Security decision → privacy impact → ethical justification

Prompt (choose ONE)

- A) Should SSP require MFA for all users? Argue yes/no with equity impacts.
- B) Should SSP keep detailed access logs for 1 year? Argue yes/no with privacy impacts.
- C) Should instructors ever see counseling-related flags? Argue policy boundaries and harms.

Required structure Claim • Reasons • Stakeholders • Counterargument • Ethical lens • Conclusion