

Môn: An toàn và Phục hồi Dữ liệu
Bộ môn: Bộ môn Công nghệ Tri thức

Đồ án: Sao lưu & phục hồi dữ liệu an toàn

Thời gian: 2 tuần
Nộp bài: Code + Report (PDF)

Nhóm: 4 sinh viên/nhóm
Demo: Video \leq 5 phút

1. Mục tiêu và phạm vi

Mục tiêu: xây dựng một chương trình dòng lệnh (CLI) để **sao lưu** và **phục hồi** một thư mục dữ liệu theo dạng snapshot, có **kiểm chứng tính toàn vẹn dữ liệu, chống rollback, an toàn khi crash**, và có **audit log** kèm policy kiểm soát thao tác.

Chương trình phải đảm bảo:

- **Phục hồi đúng:** `restore` tái tạo lại thư mục giống hệt thời điểm tạo snapshot (đúng nội dung file và cấu trúc thư mục).
- **Toàn vẹn:** `verify/restore` phát hiện dữ liệu trong backup store bị sửa/thiếu; nếu sai thì dừng và từ chối phục hồi.
- **Rollback:** phát hiện tình huống snapshot mới nhất bị thay bằng snapshot cũ hơn.
- **An toàn khi crash:** nếu bị tắt đột ngột trong lúc `backup`, lần chạy sau backup store vẫn nhất quán; snapshot tạo dở, bị lỗi không được xuất hiện như snapshot hợp lệ.
- **Audit & policy:** mọi lệnh phải được ghi vào audit log; một policy đơn giản quyết định ai được phép chạy lệnh nào.

Phạm vi dữ liệu: sao lưu và phục hồi **một thư mục** gồm nhiều file bất kỳ và thư mục con (ví dụ: log, tài liệu, ảnh, mã nguồn, dữ liệu xuất ra từ ứng dụng).

2. Nội dung chi tiết:

2.1. Phần A — Backup / Verify / Restore

Nhóm xây dựng một backup store cho thư mục `dataset/`:

- `backup dataset/ --label "<text>"`: tạo snapshot (chia file thành các chunk, lưu chunk theo hash, tạo manifest + metadata + Merkle root).
- `verify <snapshot_id>`: kiểm chứng snapshot (phát hiện sửa/thiếu/rollback).
- `restore <snapshot_id> <out_dir>`: tái tạo snapshot vào `<out_dir>` sao cho `<out_dir>` giống hệt `dataset/` tại thời điểm snapshot.
- Crash consistency: nếu kill/crash khi `backup`, snapshot đang dở không được xuất hiện; store vẫn chạy bình thường ở lần chạy sau.

2.2. Phần B — Audit log + policy kiểm soát thao tác

Phần B yêu cầu nhóm bổ sung hai cơ chế bảo mật vận hành thường gặp trong hệ thống sao lưu thực tế:

- **Policy kiểm soát thao tác:** xác định ai được chạy lệnh nào (ví dụ: chỉ admin được `init`, chỉ operator được `backup`, `restore` là thao tác nhạy cảm nên phải bị kiểm soát).
- **Audit log:** mọi lệnh (thành công hay thất bại) đều được ghi vào một log dạng append-only có **hash chain**; nếu ai đó sửa/xoá/chèn log thì `audit-verify` phải phát hiện được.

Lưu ý: Đồ án **không** làm login/OAuth, và bỏ qua việc kiểm tra bảo mật ở bước xác định danh tính. Chương trình phải xác định USER dựa trên tài khoản hệ điều hành:

- Mặc định: USER = `os_user` (username tương ứng với `getuid()` / `whoami`).
- Nếu chạy qua `sudo`: nếu môi trường có `SUDO_USER`, bắt buộc dùng USER = `SUDO_USER` (người gọi sudo), **không** ghi `root` để tránh mờ hồ.
- Nếu không xác định được username: từ chối chạy và ghi audit log với `STATUS=FAIL`.

Gợi ý cấu trúc thư mục demo:

- `dataset/` (thư mục cần backup/restore)
- `store/` (backup store do nhóm tạo)
- `policy.yaml` (policy)

3. Thuật ngữ và quy ước

- **Chunk:** đoạn dữ liệu cắt từ nội dung file để lưu trữ và tái sử dụng.
- **Content-addressable storage:** lưu chunk theo hash của chính chunk đó.
- **Snapshot:** phiên bản của toàn bộ thư mục tại một thời điểm, gồm manifest + metadata và tham chiếu tới chunks.
- **Manifest:** mô tả cây thư mục và cách tái tạo từng file từ các chunk.
- **Merkle root:** giá trị hash tổng hợp đại diện cho toàn bộ snapshot; sai lệch ở bất kỳ phần nào ⇒ Merkle root thay đổi.
- **Journal/WAL:** log append-only ghi lại cập nhật metadata khi tạo snapshot để đảm bảo nhất quán sau crash.
- **Policy:** file cấu hình quyết định user/role nào được phép chạy lệnh nào (user ở đây là **OS user**).
- **Audit log:** log append-only có hash chain; mọi thay đổi về quá khứ (sửa/xoá/chèn) phải bị phát hiện khi kiểm tra.

Quy ước:

- Hash bắt buộc dùng **SHA-256**.

- Chunking: nhóm chọn kích thước chunk **cố định** và ghi rõ trong README (khuyến nghị 1MiB).
- Lhi restore: so sánh theo cấu trúc cây thư mục + nội dung file. Không bắt buộc khôi phục metadata hệ điều hành (owner/permission/mtime).

4. Yêu cầu chức năng — Phần A (Backup/Verify/Restore)

4.1. CLI tối thiểu

Chương trình phải có CLI với các lệnh tương đương sau (tên lệnh có thể khác, nhưng phải đúng chức năng, có thể thêm số nhưng phải ghi rõ trong báo cáo).

- `init <backup_store_path>`
- `backup <source_path> --label "<text>"`
- `list-snapshots`
- `verify <snapshot_id>`
- `restore <snapshot_id> <target_path>`
- `audit-verify`

4.2. Snapshot và lưu trữ

Hệ thống phải:

- Chia nội dung file thành các chunk và lưu chunk theo hash (content-addressable).
- Tạo manifest mô tả mapping đường dẫn file tương đối → danh sách chunk hashes theo đúng thứ tự.
- `restore` tái tạo lại toàn bộ snapshot tại `<target_path>`.
- Tái sử dụng chunk giữa các snapshot nếu nội dung không đổi (dedup theo hash).

Khuyến nghị dataset test tối thiểu: 200MB hoặc 2000 files (nhóm có thể tự sinh dữ liệu).

4.3. Canonical manifest và Merkle root

Để Merkle root tính ra là tất định (deterministic), nhóm phải chuẩn hoá manifest theo quy ước:

- Liệt kê tất cả file theo đường dẫn tương đối với thứ tự tăng dần.
- Với mỗi file, danh sách chunk hashes giữ đúng thứ tự từ đầu file đến cuối file.
- Canonical encoding: nhóm chọn JSON hoặc YAML nhưng phải mô tả rõ trong README cách encode để đảm bảo không thay đổi giữa các lần chạy.

Merkle root của snapshot được tính từ danh sách entry của manifest sau chuẩn hoá. Nhóm phải ghi rõ thuật toán Merkle trong README và report.

4.4. verify và restore

Hệ thống phải:

- Lưu Merkle root trong metadata của snapshot.
- Khi chạy **verify** và **restore**, bắt buộc tính lại Merkle root và so khớp.
- Nếu không khớp: dừng và báo lỗi rõ ràng, không tiếp tục phục hồi.

Demo:

1. Sửa 1 byte trong một chunk đã lưu \Rightarrow **verify fail**.
2. Sửa manifest hoặc metadata \Rightarrow **verify fail**.
3. **Rollback:** snapshot mới bị thay bằng snapshot cũ \Rightarrow phải phát hiện được.

4.5. Chống rollback

Nhóm chọn một cơ chế và triển khai kèm test:

- **Hash chain:** snapshot mới chứa `prev_root`; kiểm tra chuỗi liên tục.
- **Append-only roots log:** lưu danh sách root dạng append-only;
- **Sign root (mô phỏng):** có khoá authority (mô phỏng/hardcode) để ký lên root; verify chữ ký.

4.6. Crash consistency cho metadata bằng journal/WAL

Nếu chương trình bị kill/crash trong lúc backup, lần chạy sau phải đảm bảo:

- Snapshot chỉ xuất hiện trong `list-snapshots` khi đã commit đầy đủ.
- Không có snapshot không hợp lệ làm **verify/restore** lỗi.
- Backup store trở về trạng thái nhất quán.

Yêu cầu tối thiểu:

- Journal/WAL là file append-only chứa `BEGIN`, các record cập nhật metadata cần thiết, và `COMMIT`.
- Khi khởi động lại, hệ thống đọc journal để xử lý giao dịch chưa commit.

5. Yêu cầu chức năng — Phần B (Policy & audit log)

5.1. Policy: `policy.yaml`

Nhóm cung cấp file `policy.yaml` đặt cạnh chương trình (hoặc trong backup store, nhóm tự chọn nhưng phải ghi rõ trong README).

Định nghĩa key user: Key trong `users` là `osUserName`, và phải có đúng định dạng:

- **Linux/macOS:** osUserName là username hệ điều hành (ví dụ: alice, bob).
- **Windows:** osUserName là DOMAIN\Username nếu có domain, nếu không thì Username (ví dụ: ACME\bob hoặc bob).

Schema bắt buộc:

- **users:** mapping osUserName -> role, ví dụ alice: admin.
- **roles:** mapping role -> allowedCommands.

Trong đồ án này, **role** bắt buộc có 3 loại sau:

- **admin:** được phép chạy mọi lệnh.
- **operator:** được phép backup, list-snapshots, verify, restore, audit-verify.
- **auditor:** chỉ được phép list-snapshots, verify, audit-verify.

Yêu cầu bắt buộc: Nếu USER không tồn tại trong policy hoặc role không cho phép lệnh, chương trình phải:

- từ chối thực hiện,
- in thông báo lỗi rõ ràng,
- và ghi audit log với trạng thái DENY.

5.2. Audit log

Chương trình phải ghi audit log append-only vào một file (ví dụ: store/audit.log). Mỗi lần chạy một lệnh, chương trình phải ghi đúng 1 dòng vào audit log, theo định dạng cố định sau:

- ENTRY_HASH PREV_HASH UNIX_MS USER COMMAND ARGS_SHA256 STATUS

Trong đó:

- **UNIX_MS:** thời gian dạng epoch milliseconds.
- **USER:** danh tính lấy từ OS theo mục 2.2 (ưu tiên SUDO_USER nếu có).
- **ARGS_SHA256:** SHA-256 của chuỗi tham số của lệnh. Nhóm định nghĩa rõ trong README; khuyến nghị: join các token sau tên lệnh bằng 1 dấu cách.
- **STATUS:** chỉ được là OK hoặc DENY hoặc FAIL.

Ý nghĩa STATUS:

- **OK:** policy cho phép và lệnh thực thi thành công.
- **DENY:** bị từ chối do policy hoặc thiếu --ticket hoặc (nếu có --user) thì --user không khớp USER từ OS.
- **FAIL:** policy cho phép nhưng thất bại khi thực thi hoặc khi kiểm chứng toàn vẹn (ví dụ verify mismatch, thiếu chunk, lỗi I/O).

5.3. Lệnh audit-verify

Lệnh `audit-verify` phải:

- đọc toàn bộ audit log,
- tính lại `ENTRY_HASH` cho từng dòng,
- kiểm tra chuỗi `PREV_HASH`,
- nếu phát hiện sai: dừng và báo số dòng bị lỗi + lý do.

Kết quả:

- nếu hợp lệ: in `AUDIT OK` và hash của dòng cuối (head).
- nếu không hợp lệ: in `AUDIT CORRUPTED` và chi tiết.

6. Kiểm thử

Nhóm phải có script/lệnh tự động tạo lỗi và chạy `verify/restore/audit-verify`. Tối thiểu:

1. Xoá một số file từ source, `restore` từ snapshot và so sánh kết quả (cây thư mục + nội dung file).
2. Sửa tối thiểu 1 byte trong chunk; `verify` phải fail.
3. Sửa manifest/metadata; `verify` phải fail.
4. Rollback: thay snapshot mới bằng snapshot cũ; sau đó chương trình phải phát hiện được.
5. Kill chương trình giữa lúc `backup`; lần chạy sau không được có snapshot lỗi và store vẫn hoạt động.
6. Policy: chạy một lệnh không được phép dựa theo role của OS user hiện tại và phải bị từ chối và có audit log `DENY`.
7. Audit: sửa 1 ký tự trong `audit.log` hoặc xoá 1 dòng; `audit-verify` phải báo `AUDIT CORRUPTED`.

7. Yêu cầu nộp bài

- **Source code + README:**

- hướng dẫn cài đặt/chạy;
- mô tả chunk size, canonical manifest, cách tính Merkle;
- mô tả cơ chế chống rollback và cách reproduce test rollback;
- mô tả journal/WAL và cách reproduce crash test;
- file `policy.yaml`: schema + ví dụ user/role (OS user);
- audit log: định dạng dòng, cách tính hash, và cách chạy `audit-verify`.
- cách chương trình xác định USER từ OS (ưu tiên `SUDO_USER` nếu có).

- **Report (PDF):**

- **Test report:** mô tả từng test, cách reproduce, kết quả.
 - **Design report (≤ 10 trang):**
 - * Threat model và giả định (đặc biệt: danh tính dựa trên OS user).
 - * Thiết kế snapshot/chunk store/manifest.
 - * Thiết kế Merkle verification và canonical manifest.
 - * Thiết kế chống rollback.
 - * Thiết kế journal/WAL và logic phục hồi sau crash.
 - * Thiết kế policy enforcement và audit log hash chain.
 - * Giới hạn hệ thống.
 - **Video demo (≤ 5 phút):** tạo snapshot, tạo lỗi (tamper/rollback/crash), chứng minh verify/restore đúng, demo policy deny, và demo audit-verify. Video demo upload lên Google drive và nộp link chia sẻ.

Phụ lục A. Ví dụ policy.yaml

```
users:
  alice: admin
  bob: operator
  eve: auditor

roles:
  admin:
    - init
    - backup
    - list-snapshots
    - verify
    - restore
    - audit-verify
  operator:
    - backup
    - list-snapshots
    - verify
    - restore
    - audit-verify
  auditor:
    - list-snapshots
    - verify
    - audit-verify
```

Phụ lục B. Ví dụ 2 dòng audit log

Tài liệu tham khảo

- CMU 15-445/645 (Fall 2019), Project #4: Logging & Recovery:
<https://15445.courses.cs.cmu.edu/fall2019/project4/>
- CMU 15-445/645 (Fall 2022), Lecture Notes #19: Logging Schemes:
<https://15445.courses.cs.cmu.edu/fall2022/notes/19-logging.pdf>
- MIT 6.5840:
<https://pdos.csail.mit.edu/6.824/>
- Stanford CS255 (Winter 2016), Project 1 (Merkle root verification):
https://crypto.stanford.edu/~dabo/courses/cs255_winter16/hw_and_proj/proj1.pdf
- OWASP Cheat Sheet Series, Logging Cheat Sheet:
https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html