ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG

BÁO CÁO THỰC HÀNH: LAB 04

HASHING, PKI, VÀ CÁC KỸ THUẬT TẤN CÔNG THỰC TẾ

(MD5 Collision & Length-Extension Attack)

MÔN HỌC: MẬT MÃ HỌC

Giảng viên hướng dẫn: ThS. NGUYỄN BÙI KIM NGÂN
Lớp: NT209.Q12.ANTT
Sinh viên thực hiện: **NGUYỄN HOÀNG QUÝ** – 24521494

Thành phố Hồ Chí Minh, tháng 11 năm 2025

Mục lục

1 Mục tiêu học tập (Learning Outcomes)

Lưu ý về Đạo đức (Ethics): Bài thực hành này chỉ phục vụ mục đích giáo dục phòng thủ (Defensive Education). Sinh viên chỉ được sử dụng các tập tin kiểm thử ngoại tuyến (offline) thuộc sở hữu cá nhân trong môi trường biệt lập. Nghiêm cấm thực hiện tấn công vào các hệ thống thực tế.

Các mục tiêu chính của bài Lab bao gồm:

1. Cài đặt và đánh giá hiệu năng (Benchmark) các hàm băm (Hash functions).
2. Phân tích (Parse) và kiểm tra tính hợp lệ của chứng chỉ X.509.
3. Thực hiện tấn công đụng độ (Collision) và mở rộng độ dài (Length-extension) trên MD5 một cách an toàn nhằm minh họa lỗ hổng.
4. Đề xuất các biện pháp phòng chống và giảm thiểu rủi ro (Mitigations).

2 Thiết kế & Bảo mật (Design & Security)

(Phần này trình bày kiến trúc tổng quan của bộ công cụ, các thư viện sử dụng và các nguyên tắc bảo mật được áp dụng trong quá trình lập trình.)

3 Nội dung thực hiện (Tasks)

3.1 Task 4.1: Hashing Suite

3.1.1 Yêu cầu (Requirements)

Xây dựng công cụ dòng lệnh (CLI Tool) thực hiện các chức năng sau:

- **Hỗ trợ đa thuật toán:** Tính toán được các hàm băm SHA-2 (224/256/384/512), SHA-3 (224/256/384/512), và SHAKE (128/256).
- **SHAKE Support:** Cho phép người dùng tùy chỉnh độ dài đầu ra thông qua tham số `-outlen BYTES`.
- **Hiệu năng (Performance):** Hỗ trợ xử lý tập tin kích thước lớn (multi-GB) sử dụng kỹ thuật Streamed I/O để tối ưu bộ nhớ.
- **Định dạng đầu ra:** Hiển thị mã Hex trên màn hình và lưu dữ liệu Raw (binary) xuống tập tin.
- **Kiểm thử:** Xác minh độ chính xác với bộ test vectors KATs (Known Answer Tests).

3.1.2 Hiện thực hóa (Implementation)

Mã nguồn được cài đặt trong tập tin `hash.cpp`, sử dụng thư viện **Crypto++** để gọi các digest.

Để giải quyết vấn đề xử lý tập tin dung lượng lớn mà không gây tràn bộ nhớ (Memory Overflow),Em áp dụng kỹ thuật **Pipelining/Streaming** (sử dụng `FileSource` của Crypto++). Dữ liệu được đọc và xử lý theo từng khối (chunk) thay vì nạp toàn bộ vào RAM.

Sau khi quá trình băm hoàn tất, chương trình sẽ tự động trích xuất một tập tin `.json` (sidecar file) chứa các thông tin thống kê bao gồm:

- Thời điểm thực thi (Timestamp).
- Kích thước tập tin đầu vào (Input Size).
- Thuật toán sử dụng (Algorithm Mode).
- Thời gian xử lý thực tế (Runtime/Benchmark Duration).

3.2 Task 4.2: PKI & Certificates

3.2.1 Yêu cầu

Sử dụng thư viện OpenSSL để phân tích (Parse) một chứng chỉ X.509 và thực hiện:

- In ra các thông tin: Subject, Issuer, Thuật toán ký, Thời hạn (Validity), Key Usages và SANs.
- Xác thực chữ ký số (Signature Verification): Kiểm tra chữ ký của chứng chỉ dựa trên Public Key của Issuer (nếu có) hoặc kiểm tra tính toàn vẹn cấu trúc TBS nếu không có Issuer.

3.2.2 Thực hiện (Implementation

(Mô tả cách dùng các hàm của OpenSSL như `PEM_read_X509`, `X509_verify` tại đây...)

3.3 Task 4.3: Collisions with MD5

Mục tiêu: Chứng minh sự thiếu an toàn của hàm digest MD5 bằng cách tạo ra hai tập tin khác nhau (binary khác nhau) nhưng lại có cùng giá trị băm MD5 (Collision).

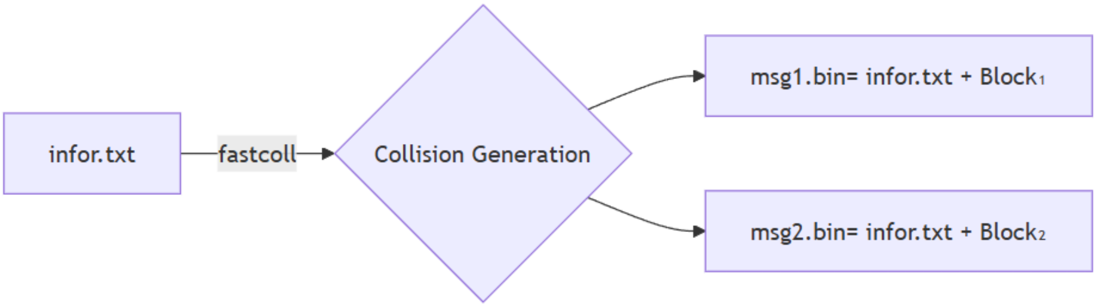
3.3.1 Thực hiện(Implementation

Sử dụng công cụ fastcoll (thuộc bộ hashclash) để tấn công theo phương pháp Identical Prefix Collision. Chào bạn, dưới đây là nội dung báo cáo chi tiết cho Task 4.3, được trình bày theo cấu trúc bạn yêu cầu. Mình đã sửa lại một chút lỗi logic trong câu hỏi của bạn (thực tế là MD5 giống nhau nhưng binary khác nhau).

Kịch bản thực hiện:

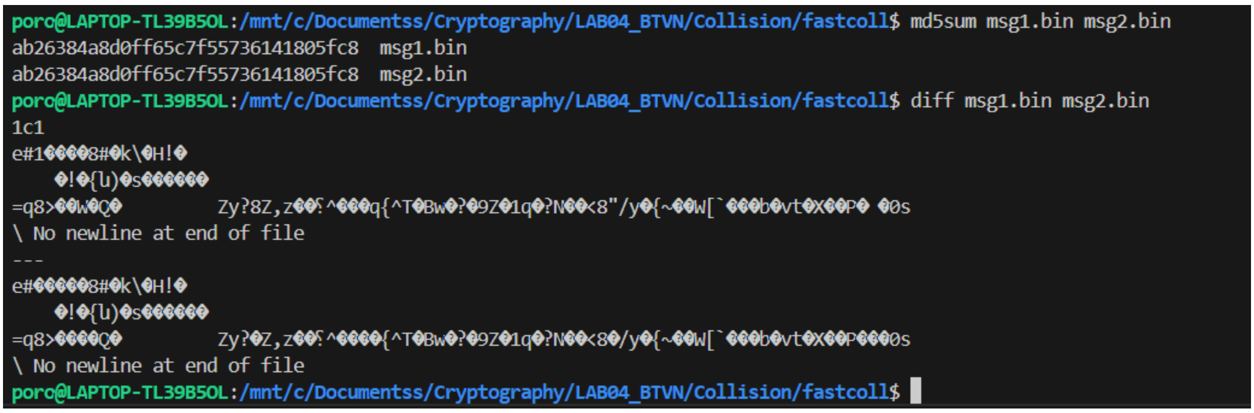
- Bước 1 (Chuẩn bị Input): Tạo một file văn bản đầu vào có tên `infor.txt`. File này chứa thông tin prefix (tiền tố) chung cho cả hai file va chạm sau này.
- Bước 2 (Tạo Collision): Sử dụng lệnh `fastcoll` để xử lý file `infor.txt`. Công cụ sẽ tính toán trạng thái băm (internal state) sau khi xử lý nội dung của `infor.txt`, sau đó tự động tìm kiếm và chèn thêm hai khối dữ liệu đặc biệt (collision blocks - mỗi khối 128 bytes) vào cuối.
- Bước 3 (Kết quả Output): Công cụ sinh ra hai file đầu ra là `msg1.bin` và `msg2.bin`.

Sơ đồ quá trình:



Hình 1: Quá trình fastcol tạo collision

3.3.2 Kết quả(Result)



Hình 2: Kết quả so sánh mã hash md5

3.3.3 Giải thích tại sao MD5 không còn an toàn trong hiện tại

MD5 (Message Digest Algorithm 5) là hàm băm mật mã tạo ra giá trị băm 128-bit. Mặc dù từng được sử dụng rộng rãi để kiểm tra tính toàn vẹn dữ liệu và lưu trữ mật khẩu, hiện nay MD5 được

coi là **đã bị phá vỡ hoàn toàn về mặt mật mã học** (cryptographically broken). Dưới đây là các lý do chính dẫn đến kết luận này:

- **Phá vỡ tính Kháng va chạm (Collision Resistance):** Theo lý thuyết, với độ dài 128-bit, một cuộc tấn công brute-force tìm va chạm (Birthday Attack) sẽ cần khoảng 2^{64} phép tính. Tuy nhiên, các nhà nghiên cứu (tiêu biểu là Xiaoyun Wang và Marc Stevens) đã phát hiện ra các điểm yếu trong cấu trúc hàm nén của MD5.

  Thực nghiệm trong bài Lab này với công cụ `fastcoll` cho thấy việc tìm ra hai tập tin có cùng mã MD5 chỉ mất vài giây trên máy tính cá nhân. Điều này chứng tỏ chi phí tấn công đã giảm từ mức "bất khả thi"xuống mức "tầm thường".

- **Tấn công Tiền tố Giống nhau (Identical Prefix Attack):** Kỹ thuật này cho phép kẻ tấn công chuẩn bị sẵn một phần nội dung hợp lệ (Prefix), sau đó tính toán và chèn thêm các khối dữ liệu rác (Collision Blocks) để tạo ra hai tập tin có nội dung khác nhau nhưng cùng mã băm. Điều này đặc biệt nguy hiểm đối với các định dạng tập tin phức tạp như tài liệu văn bản, mã thực thi (executable files) hoặc chứng chỉ số.

- **Các cuộc tấn công thực tế (Real-world Exploits):** Sự yếu kém của MD5 không chỉ nằm trên lý thuyết. Điển hình là mã độc **Flame (2012)**, đã lợi dụng kỹ thuật va chạm MD5 (Chosen-prefix collision) để giả mạo chữ ký số của Microsoft. Điều này cho phép mã độc tự nhận là một bản cập nhật Windows hợp pháp để xâm nhập vào hệ thống nạn nhân mà không bị phát hiện.

- **Lỗ hổng mở rộng độ dài (Length Extension Attack):** MD5 được xây dựng dựa trên cấu trúc Merkle-Damgård. Nếu không sử dụng các cơ chế như HMAC, kẻ tấn công biết $H(m)$ và độ dài của m có thể tính toán được $H(m||padding||m')$ mà không cần biết nội dung ban đầu của m (như đã minh họa trong Task 4.4).

Kết luận: Do không còn đảm bảo được các tính chất an toàn cơ bản, đặc biệt là tính kháng va chạm, MD5 hiện bị cấm sử dụng trong các ứng dụng yêu cầu bảo mật cao như chữ ký số (Digital Signatures), chứng chỉ SSL/TLS hay lưu trữ mật khẩu. Các tổ chức tiêu chuẩn (như NIST) khuyến nghị chuyển sang sử dụng các dòng thuật toán SHA-2 (SHA-256/512) hoặc SHA-3.

3.4 Task 4.4: Length-Extension on MACs of the form H(k || m)

Length-Extension Attack là cuộc tấn công nhắm vào các hàm băm theo cấu trúc Merkle–Damgård (như MD5, SHA-1, SHA-256), cho phép attacker biết hash(k‖m) và độ dài k có thể tính ra hash(k‖m‖padding‖m’ mà không cần biết khóa bí mật. Vì trạng thái nội bộ của hàm băm có thể được tiếp tục mở rộng từ giá trị hash đã có, attacker có thể tạo ra thông điệp mới hợp lệ và MAC mới tương ứng.

3.4.1 Kịch bản mô phỏng

Chúng ta sẽ giả lập một kịch bản như sau:

- Key (k): secret (Độ dài = 6 byte).
- Message ban đầu (m): data.
- Hàm băm: SHA-256
- Dữ liệu muốn chèn thêm (m'): _append
- Tiếp đến dùng HashPump để giả mạo chữ ký hợp lệ
- Sau khi thực hiện tấn công thì kết quả MAC mong đợi là:

```
8e5fa3a068086feb2a7484d2c43a007d332a3a76data\x80\x00\x00\x00\x00\x00\
    x00 ...\x00P_append
```

3.4.2 Thực hiện(Implementation)

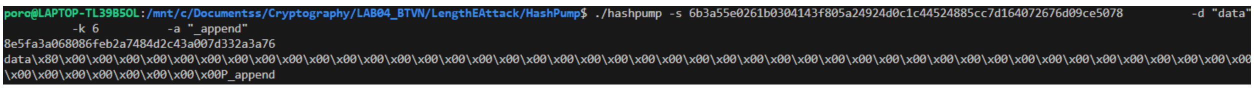
Bước 1: Tạo chữ kí số hợp lệ.

```
poro@LAPTOP-TL39B5OL:/mnt/c/Documentss/Cryptography/LAB04_BTVN/
    LengthEAttack/HashPump$ echo -n "secretdata" | sha256sum
ca2c6fe0b879f58a8afea413fef7202c94a5156ccfb5d22c4ef690af5117a081
```

Mã hash hiện tại của chúng ta là:

```
ca2c6fe0b879f58a8afea413fef7202c94a5156ccfb5d22c4ef690af5117a081
```

Bước 2.1: Tạo chữ kí giả mạo bằng HashPump.



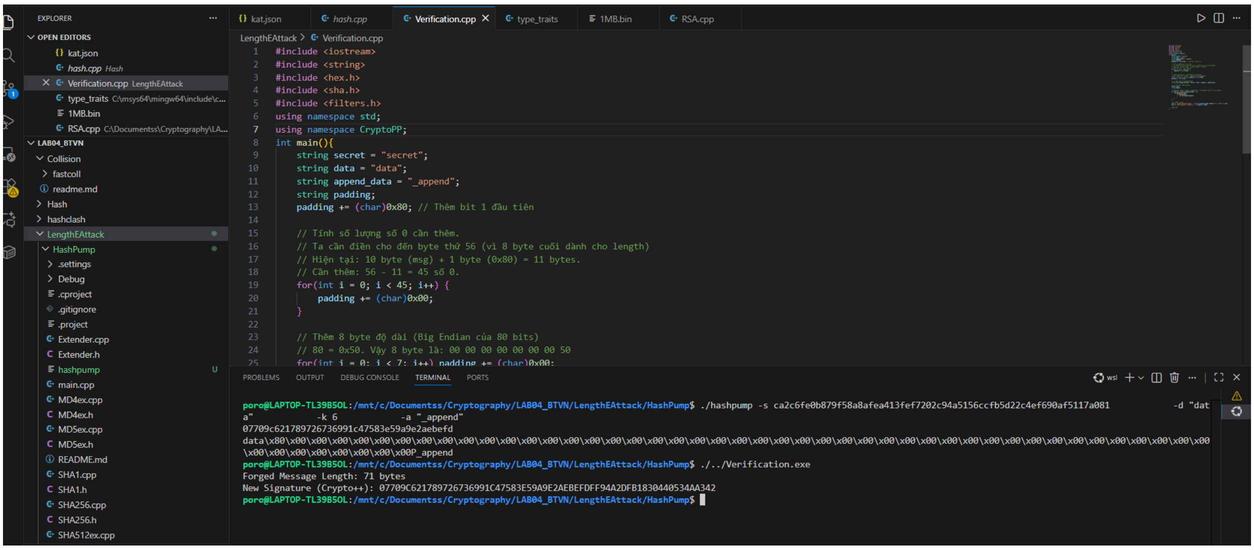
Hình 3: Chữ kí giả mạo tạo ra bởi HashPump

Chữ kí giả mạo:

```
07709c621789726736991c47583e59a9e2aebefd
```

Có thể thấy là ta đã thành công tính được H(k || m || pad || m') mà không cần biết secretkey.

Bước 2.2: Tạo chữ kí giả bằng code Cpp (Bonus). Trong hàm Verification.cpp thực hiện thủ công việc padding và thêm message mới rồi in ra chữ kí giả mạo mới.



Hình 4: Chữ kí giả mạo của Verification.cpp

Chữ kí giả mạo từ Verification.cpp:

```
New Signature (Crypto++):
07709C621789726736991C47583E59A9E2AEBEFDFF94A2DFB1830440534AA342
```

Có thể thấy là 2 chữ kí hoàn toàn giống nhau.

4 Kiểm thử (Testing)

4.1 So sánh đối chiếu (Cross-Verification)

Kết quả của chương trình được so sánh với công cụ như **OpenSSL CLI** để đảm bảo tính chính xác.

4.2 Known Answer Tests (KATs)

Chương trình tích hợp chế độ tự động kiểm tra (`-kat`) dựa trên bộ dữ liệu chuẩn **FIPS 140 (8/2015)** do NIST ban hành. Các test vectors bao gồm nhiều trường hợp biên như: chuỗi rỗng, chuỗi ngắn, và chuỗi lặp lại kích thước lớn.Các vector được lưu trong kat.json.

```
poro@LAPTOP-TL39B5OL:/mnt/c/Documentss/Cryptography/LAB04_BTVN/hash$ cat hash_kat_results.csv
filename,count,algorithm,pass
kat.json,1,SHA-1,1
kat.json,1,SHA-224,1
kat.json,1,SHA-256,1
kat.json,1,SHA-3-224,1
kat.json,1,SHA-3-256,1
kat.json,1,SHA-3-384,1
kat.json,1,SHA-3-512,1
kat.json,1,SHA-384,1
kat.json,1,SHA-512,1
kat.json,1,SHA-1,1
kat.json,1,SHA-224,1
kat.json,1,SHA-256,1
kat.json,1,SHA-3-224,1
kat.json,1,SHA-3-256,1
kat.json,1,SHA-3-384,1
kat.json,1,SHA-3-512,1
kat.json,1,SHA-384,1
kat.json,1,SHA-512,1
kat.json,1,SHA-1,1
kat.json,1,SHA-224,1
kat.json,1,SHA-256,1
kat.json,1,SHA-3-224,1
kat.json,1,SHA-3-256,1
kat.json,1,SHA-3-384,1
kat.json,1,SHA-3-512,1
kat.json,1,SHA-384,1
kat.json,1,SHA-512,1
kat.json,1,SHA-1,1
kat.json,1,SHA-224,1
kat.json,1,SHA-256,1
kat.json,1,SHA-3-224,1
kat.json,1,SHA-3-256,1
kat.json,1,SHA-3-384,1
kat.json,1,SHA-3-512,1
kat.json,1,SHA-384,1
kat.json,1,SHA-512,1
kat.json,1000000,SHA-1,1
kat.json,1000000,SHA-224,1
kat.json,1000000,SHA-256,1
kat.json,1000000,SHA-3-224,1
kat.json,1000000,SHA-3-256,1
kat.json,1000000,SHA-3-384,1
kat.json,1000000,SHA-3-512,1
kat.json,1000000,SHA-384,1
kat.json,1000000,SHA-512,1
```

Hình 5: Kết quả kiểm thử KAT

5 Đánh giá hiệu năng (Performance)