

## BÁO CÁO THỰC HÀNH

Môn học: An toàn dữ liệu, khôi phục thông tin sau sự cố  
Tên chủ đề: Hash và Digital Signature trong Data Integrity  
GVHD: Ngô Khánh Khoa

1. **THÔNG TIN CHUNG:**

(Liệt kê tất cả các thành viên trong nhóm)

Lớp: NT212.Q11.ANTT

STT	Họ và tên	MSSV	Email
1	Lâm Xuân Thái	22521317	22521317@gm.uit.edu.vn
2	Phạm Minh Tân	22521310	22521310@gm.uit.edu.vn

Phần bên dưới của báo cáo này là tài liệu báo cáo chi tiết của nhóm thực hiện.

# BÁO CÁO CHI TIẾT

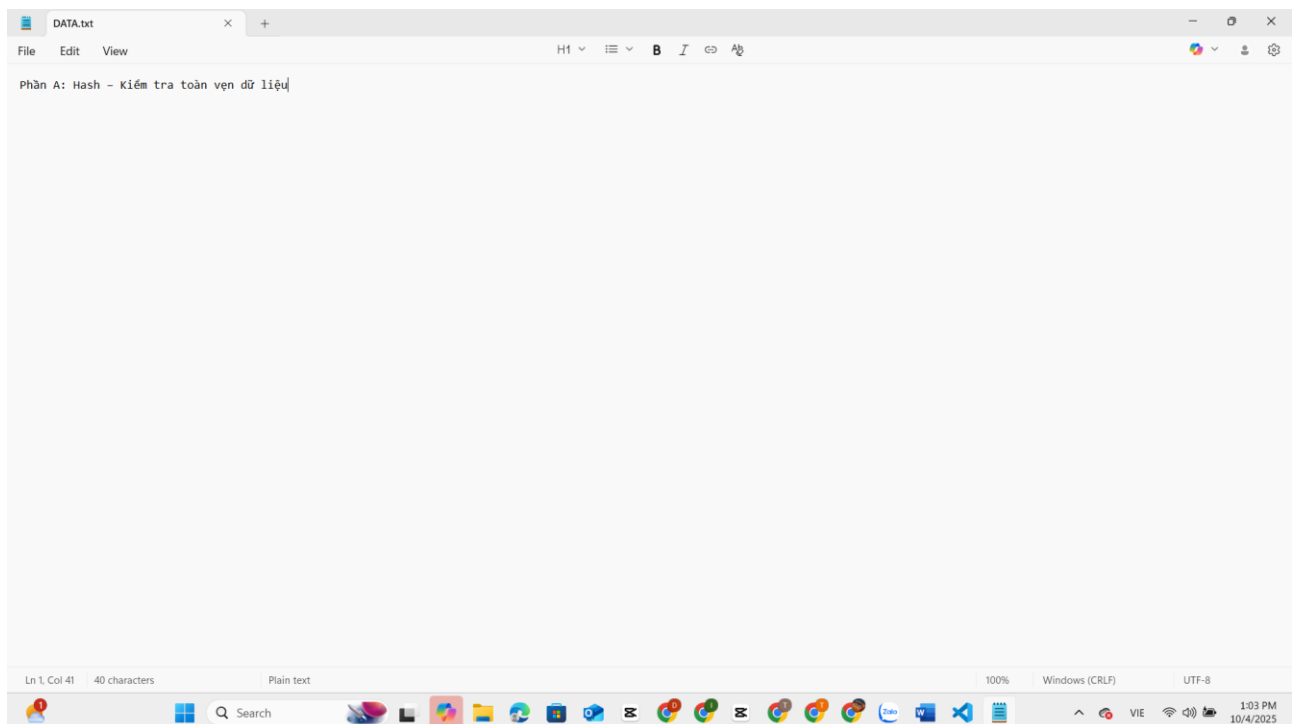
## Phần A: Hash – Kiểm tra toàn vẹn dữ liệu

### 1. Mục tiêu

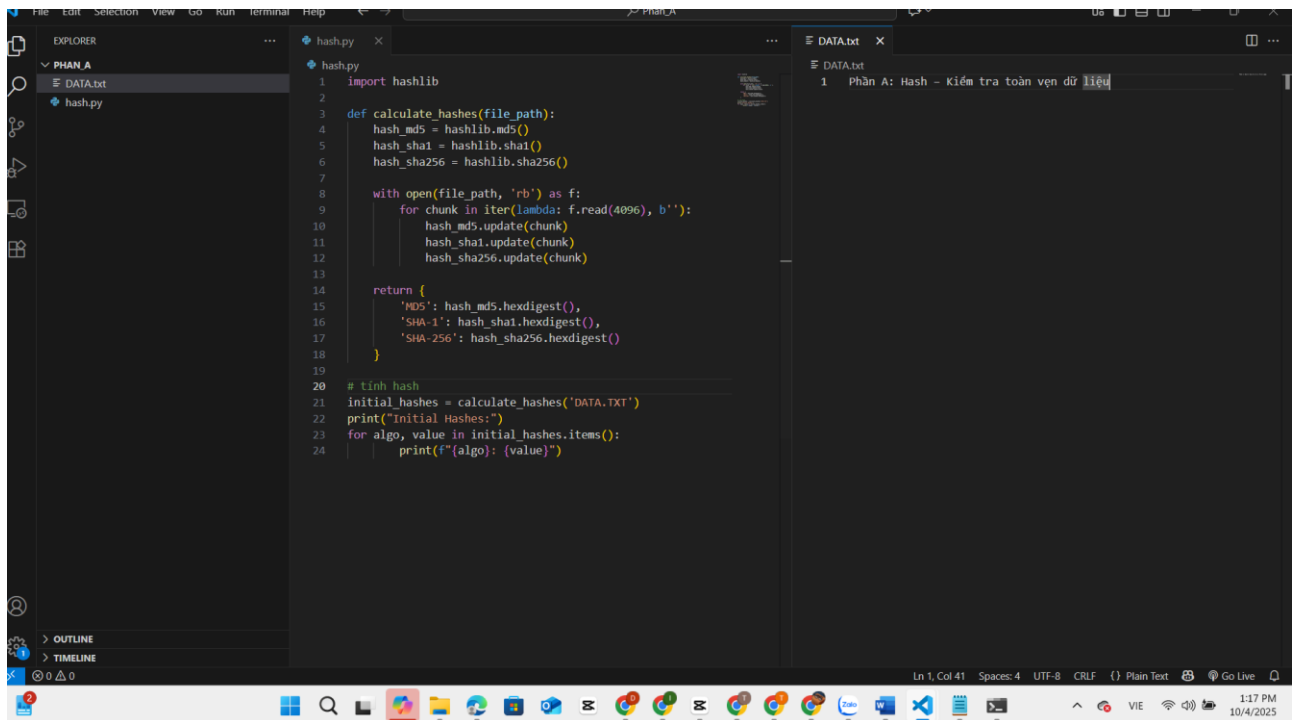
- Thực hành tính toán giá trị hash (MD5, SHA-1, SHA-256).
- Quan sát sự thay đổi hash khi dữ liệu bị sửa đổi.

### 2. Yêu cầu

#### 1. Tạo một file văn bản **DATA.TXT**



#### 2. Tính hash file bằng 3 thuật toán: **MD5, SHA-1, SHA-256** .

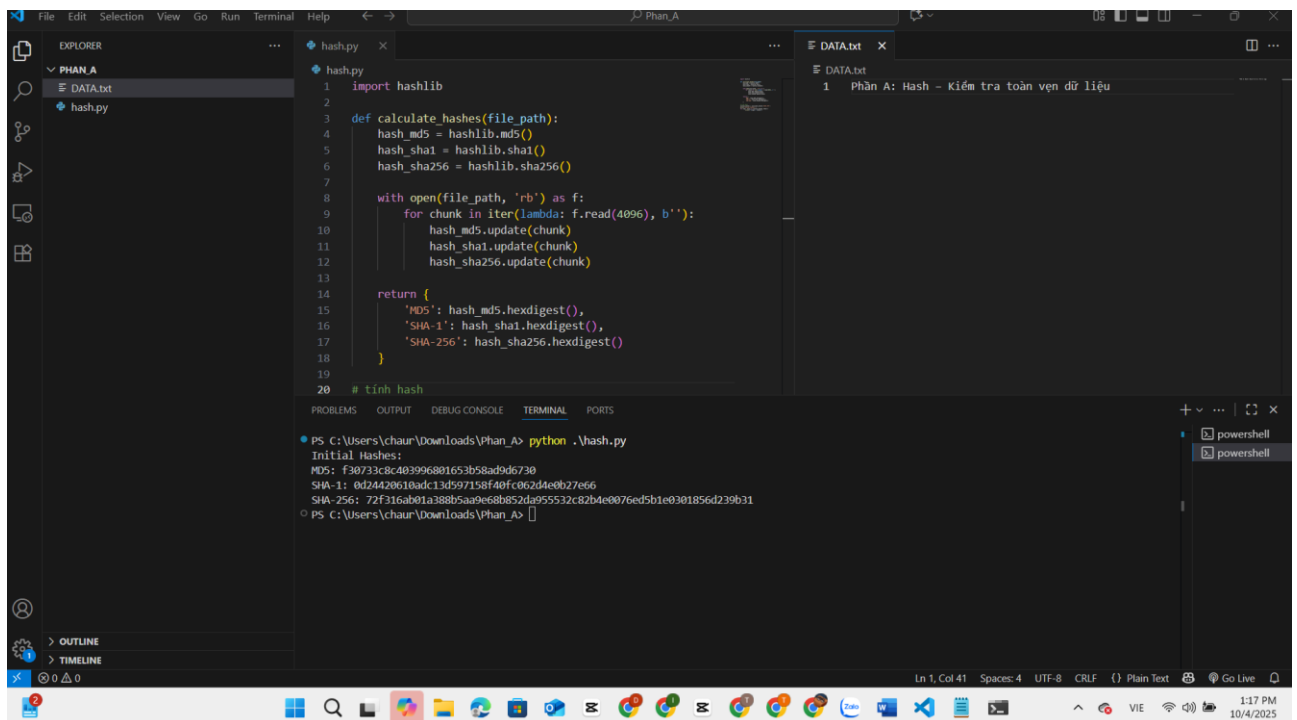


The screenshot shows a Visual Studio Code editor with two files open: `hash.py` and `DATA.txt`. The `hash.py` script is designed to calculate MD5, SHA-1, and SHA-256 hashes for a given file. It uses the `hashlib` module and processes the file in chunks of 4096 bytes. The `DATA.txt` file contains the text: "Phần A: Hash - Kiểm tra toàn vẹn dữ liệu".

```

1 import hashlib
2
3 def calculate_hashes(file_path):
4     hash_md5 = hashlib.md5()
5     hash_sha1 = hashlib.sha1()
6     hash_sha256 = hashlib.sha256()
7
8     with open(file_path, 'rb') as f:
9         for chunk in iter(lambda: f.read(4096), b''):
10             hash_md5.update(chunk)
11             hash_sha1.update(chunk)
12             hash_sha256.update(chunk)
13
14     return {
15         'MD5': hash_md5.hexdigest(),
16         'SHA-1': hash_sha1.hexdigest(),
17         'SHA-256': hash_sha256.hexdigest()
18     }
19
20 # tính hash
21 initial_hashes = calculate_hashes('DATA.TXT')
22 print("Initial Hashes:")
23 for algo, value in initial_hashes.items():
24     print(f"{algo}: {value}")
    
```

### 3. Ghi nhận kết quả hash.



This screenshot shows the same VS Code editor after running the `hash.py` script. The terminal window at the bottom displays the output of the script, which prints the initial hashes for MD5, SHA-1, and SHA-256. The `DATA.txt` file remains unchanged.

```

PS C:\Users\chaun\Downloads\Phan_A> python .\hash.py
Initial Hashes:
MD5: f30733cbc403996801653b58ad9d6730
SHA-1: 0d24420610adc13d597158f40fc062d4e0b27e66
SHA-256: 72f316ab01a388b5aa9e68b852da955532c82b4e0076ed5b1e0301856d239b31
    
```

### 4. Thay đổi nội dung file (ví dụ sửa một ký tự).

```
DATA.txt
1 Phần A: Hash - Kiểm tra toàn vẹn dữ liệu
2
```

Thêm 1 ký từ: “1”

5. Tính lại hash và so sánh với kết quả ban đầu.

```
hash.py
1 import hashlib
2
3 def calculate_hashes(file_path):
4     hash_md5 = hashlib.md5()
5     hash_sha1 = hashlib.sha1()
6     hash_sha256 = hashlib.sha256()
7
8     with open(file_path, 'rb') as f:
9         for chunk in iter(lambda: f.read(4096), b''):
10             hash_md5.update(chunk)
11             hash_sha1.update(chunk)
12             hash_sha256.update(chunk)
13
14     return {
15         'MD5': hash_md5.hexdigest(),
16         'SHA-1': hash_sha1.hexdigest(),
17         'SHA-256': hash_sha256.hexdigest()
18     }
19
20 # Tính hash
```

```
PS C:\Users\chaur\Downloads\Phan_A> python .\hash.py
Initial Hashes:
MD5: 152a6b9c0bc2f12e87c149d70ea66add
SHA-1: bcf444cd2186a841c5f99ac3361c14616c4445c7
SHA-256: bc0d81c310bf6892802a18c3af73b839d2cb8da0cc2ee8a8735790dfe460fc77
PS C:\Users\chaur\Downloads\Phan_A> python .\hash.py
Initial Hashes:
MD5: 8fed5f6eb9e4ab36c7e1b88cd384f0eb
SHA-1: 1c36addcccc404b46ea387e3b20e7c552237bdf1
SHA-256: 4917c62c9c7069ff512cf56eb43c4e8a7b43fb54cbcd49d3c51ef82138f4a2
PS C:\Users\chaur\Downloads\Phan_A>
```

Thuật toán	Hash trước khi sửa đổi	Hash sau khi sửa đổi	Có thay đổi?
MD5	152a6b9c0bc2f12e87c149d70ea66add	8fed5f6eb9e4ab36c7e1b88cd384f0eb	Có
SHA-1	bcf444cd2186a841c5f99ac3361c14616c4445c7	1c36addcccc404b46ea387e3b20e7c552237bdf1	Có
SHA-256	bc0d81c310bf6892802a18c3af73b839d2cb8da0cc2ee8a8735790dfe460fc77	4917c62c9c7069ff512cf56eb43c4e8a7b43fb54cbcd49d3c51ef82138f4a2	Có

#### ➤ Nhận xét:

- Tất cả các giá trị hash (MD5, SHA-1, SHA-256) đều **khác nhau** giữa hai lần chạy.
- Điều này cho thấy **nội dung file DATA.TXT đã thay đổi** giữa hai lần chạy code.

### 3. Báo cáo

**Bảng so sánh giá trị hash trước và sau khi sửa file.**

Chúng em đã thực hiện tính toán giá trị hash của file DATA.TXT bằng các thuật toán **MD5**, **SHA-1**, và **SHA-256** trước và sau khi sửa đổi nội dung file. Nội dung file ban đầu là "Phần A: Hash – Kiểm tra toàn vẹn dữ liệu". Sau đó, file được sửa thành "Phần A: Hash – Kiểm tra toàn vẹn dữ liệu1" (thêm ký tự "1" vào cuối).

Thuật toán	Hash trước khi sửa đổi	Hash sau khi sửa đổi
MD5	152a6b9c0bc2f12e87c149d70ea66add	8fed5f6eb9e4ab36c7e1b88cd384f0eb
SHA-1	bcf444cd2186a841c5f99ac3361c14616c4445c7	1c36addcccc404b46ea387e3b20e7c552237bdf1
SHA-256	bc0d81c310bf6892802a18c3af73b839d2cb8da0cc2ee8a8735790dfe460fc77	4917c62c9c7069ff512cf56eb43c4e8

**Nhận xét:**

- Tất cả giá trị hash thay đổi hoàn toàn sau khi sửa nội dung file.
- Sự khác biệt này chứng minh rằng các thuật toán hash rất nhạy với bất kỳ thay đổi nào trong dữ liệu.

**Giải thích tại sao chỉ thay đổi 1 ký tự mà hash thay đổi hoàn toàn.**

Hàm hash (như MD5, SHA-1, SHA-256) có tính chất (**avalanche effect**), nghĩa là một thay đổi nhỏ trong dữ liệu đầu vào sẽ tạo ra một giá trị hash hoàn toàn khác. Trong trường hợp này, việc thêm ký tự "1" vào nội dung file DATA.TXT (từ "Phần A: Hash – Kiểm tra toàn vẹn dữ liệu" thành "Phần A: Hash – Kiểm tra toàn vẹn dữ liệu1") dẫn đến sự thay đổi hoàn toàn của các giá trị hash. Lý do:

- **Cơ chế hoạt động của hàm hash:**
  - Hàm hash chuyển đổi dữ liệu đầu vào thành một chuỗi có độ dài cố định (ví dụ: 32 ký tự cho MD5, 64 ký tự cho SHA-256) thông qua các phép toán phức tạp như XOR, dịch bit, và phép toán mô-đun.
  - Một thay đổi nhỏ (như thêm ký tự "1") làm thay đổi toàn bộ chuỗi phép toán, dẫn đến giá trị hash mới không có bất kỳ điểm tương đồng nào với giá trị hash cũ.
- **Mục đích của avalanche effect:**

- Đảm bảo tính **toàn vẹn dữ liệu**: Nếu hash không thay đổi khi dữ liệu bị sửa, hàm hash sẽ không thể phát hiện được sự thay đổi, làm mất đi ý nghĩa kiểm tra toàn vẹn.
- Tăng tính **bảo mật**: Sự thay đổi hoàn toàn của hash khiến kẻ tấn công khó dự đoán hoặc tạo ra dữ liệu giả mạo có cùng giá trị hash.
- **Ví dụ minh họa**:
  - Ban đầu: "Phần A: Hash – Kiểm tra toàn vẹn dữ liệu" → Hash SHA-256: bc0d81c310bf6892802a18c3af73b839d2cb8da0cc2ee8a8735790dfe460fc77.
  - Sau khi sửa: "Phần A: Hash – Kiểm tra toàn vẹn dữ liệu1" → Hash SHA-256: 4917c62c9c7069ff512cf56eb43c4e87a7b43fb54bcebd49d3c51ef82138f4a2.
  - Mặc dù chỉ thêm một ký tự "1", giá trị hash SHA-256 (64 ký tự) thay đổi hoàn toàn, không có bất kỳ điểm tương đồng nào với hash ban đầu.

**Kết luận:** Sự thay đổi hoàn toàn của giá trị hash khi chỉ thêm một ký tự là đặc điểm cốt lõi của các hàm hash. Điều này đảm bảo rằng bất kỳ sửa đổi nào trong dữ liệu, dù nhỏ nhất, cũng sẽ được phát hiện, từ đó duy trì tính toàn vẹn của dữ liệu.

## Phần B: Digital Signature – Toàn vẹn và Xác thực dữ liệu

### 1. Mục tiêu

- Hiểu cơ chế chữ ký số với khóa công khai – khóa bí mật.
- Thực hành ký và xác minh dữ liệu.

### 2. Yêu cầu

1. Sinh cặp khóa RSA (Private key: dùng để ký dữ liệu; Public key: dùng để xác minh chữ ký)
2. Với file DATA.TXT: Tính hash của file; ký hash bằng **private key** để tạo **digital signature**.
3. Người nhận kiểm tra: Dùng **public key** để xác minh chữ ký - Nếu file bị thay đổi → chữ ký không hợp lệ.

### 3. Báo cáo

- Lưu lại chữ ký số đã sinh.
- Trình bày kết quả xác minh chữ ký **khi file gốc còn nguyên** và **khi file bị sửa đổi**.
- So sánh sự khác nhau giữa việc chỉ dùng Hash và dùng Digital Signature.

#### 3.1. Sinh khóa

- Độ dài khóa: 3072-bit

- Lưu đường dẫn tới private\_key.pem, public\_key.pem

### 3.2. Tính hash và Ký

Sinh khóa:

```
python src/signature_demo.py gen-keys --out-dir keys --bits 3072
```

```
PS D:\Course\NT212.ANTT\lab1 - Hash và Digital Signature trong Data Integrity\Phần B Digital Signature - Toàn vẹn và Xác thực dữ liệu> python src/signature_demo.py gen-keys --out-dir keys --bits 3072
Đã sinh khóa RSA 3072 bit:
- Private key: keys\private_key.pem
- Public key : keys\public_key.pem
```

Tính hash (lưu hex và nhị phân):

```
python src/signature_demo.py hash DATA.TXT
```

```
PS D:\Course\NT212.ANTT\lab1 - Hash và Digital Signature trong Data Integrity\Phần B Digital Signature - Toàn vẹn và Xác thực dữ liệu> python src/signature_demo.py hash DATA.TXT
SHA-256(DATA.TXT) = 4917c62c9c7069ff512cf56eb43c4e87a7b43fb54bcebd49d3c51ef82138f4a2
- Lưu hash nhị phân: DATA.TXT.sha256.bin
- Lưu hash hex      : DATA.TXT.sha256.txt
```

Ký HASH của file bằng private key (RSA-PSS + SHA-256, Prehashed):

```
python src/signature_demo.py sign DATA.TXT --private-key keys/private_key.pem
```

```
PS D:\Course\NT212.ANTT\lab1 - Hash và Digital Signature trong Data Integrity\Phần B Digital Signature - Toàn vẹn và Xác thực dữ liệu> python src/signature_demo.py sign DATA.TXT --private-key keys/private_key.pem
Đã ký hash của DATA.TXT
- Hash (hex): 4917c62c9c7069ff512cf56eb43c4e87a7b43fb54bcebd49d3c51ef82138f4a2
- Chữ ký   : DATA.TXT.sig (nhị phân), DATA.TXT.sig.b64 (Base64)
```

Xác minh chữ ký với public key:

```
python src/signature_demo.py verify DATA.TXT --public-key keys/public_key.pem --signature DATA.TXT.sig
```

```
PS D:\Course\NT212.ANTT\lab1 - Hash và Digital Signature trong Data Integrity\Phần B Digital Signature - Toàn vẹn và Xác thực dữ liệu> python src/signature_demo.py verify DATA.TXT --public-key keys/public_key.pem --signature DATA.TXT.sig
XÁC MINH: HỢP LỆ (file chưa bị thay đổi, chữ ký đúng).
```

Thử sửa đổi file rồi xác minh lại:

```
NT212.ANTT > lab1 - Hash và Digital Signature trong Data Integrity > Phần B Digital Signature – Toàn vẹn và Xác thực dữ liệu
1 1 Phần A: Hash – Kiểm tra toàn vẹn dữ liệu gregregregre
2 2
```

```
cp DATA.TXT DATA_MOD.TXT
```

```
echo "tamper" >> DATA_MOD.TXT
```

```
python src/signature_demo.py verify DATA.TXT --public-key keys/public_key.pem --signature DATA.TXT.sig
```

```
PS D:\Course\NT212.ANTT\lab1 - Hash và Digital Signature trong Data Integrity\Phần B Digital Signature – Toàn vẹn và Xác thực dữ liệu> python src/signature_demo.py verify DATA.TXT --public-key keys/public_key.pem --signature DATA.TXT.sig
XÁC MINH: KHÔNG HỢP LỆ (file bị thay đổi hoặc chữ ký/ khóa sai).
PS D:\Course\NT212.ANTT\lab1 - Hash và Digital Signature trong Data Integrity\Phần B Digital Signature – Toàn vẹn và Xác thực dữ liệu>
```

## Câu hỏi mở rộng

### 1. Điểm mạnh và điểm yếu của Hash trong đảm bảo toàn vẹn dữ liệu là gì?

**Hash** là một hàm toán học chuyển dữ liệu (như nội dung file, văn bản) thành một chuỗi ký tự ngắn có độ dài cố định, gọi là **giá trị hash** (ví dụ: một chuỗi 32 ký tự với MD5 hoặc 64 ký tự với SHA-256). Hash thường được dùng để kiểm tra xem dữ liệu có bị thay đổi hay không.

#### Điểm mạnh của Hash

##### 1. Nhanh và dễ tính toán:

- Hàm hash được thiết kế để xử lý nhanh, ngay cả với dữ liệu lớn (như file hàng GB). Ví dụ: Khi bạn tải file từ internet, tính hash chỉ mất vài giây.
- Code để tính hash rất đơn giản, không cần cấu hình phức tạp. Chỉ cần vài dòng code trong Python (thư viện hashlib) là có thể tính được.

##### 2. Phát hiện mọi thay đổi trong dữ liệu:

- Hash có **hiệu ứng tuyết lở** (avalanche effect): chỉ cần thay đổi một ký tự nhỏ (ví dụ: đổi chữ "A" thành "B"), giá trị hash sẽ thay đổi hoàn toàn. Điều này giúp dễ dàng phát hiện bất kỳ sửa đổi nào, dù nhỏ nhất, trong dữ liệu.
- Ví dụ: Nếu bạn tải file và hash của file khớp với hash được công bố, bạn có thể chắc chắn file không bị lỗi hoặc bị sửa.

##### 3. Đơn giản, không cần quản lý khóa:

- Hash không yêu cầu hệ thống khóa (như khóa công khai, khóa bí mật). Bạn chỉ cần chạy hàm hash trên dữ liệu và so sánh kết quả.



- Điều này làm cho hash dễ dùng trong các trường hợp đơn giản như kiểm tra file tải về từ internet.

### Điểm yếu của Hash

#### 1. Không xác thực được nguồn gốc dữ liệu:

- Ai cũng có thể tính hash của dữ liệu, nên hash không chứng minh được dữ liệu đến từ ai. Ví dụ: Nếu bạn nhận được file và giá trị hash, bạn không biết liệu file đó có thực sự từ nguồn đáng tin cậy hay không.
- Kẻ tấn công có thể giả mạo file và cung cấp một giá trị hash mới khớp với file giả mạo.

#### 2. Một số thuật toán dễ bị tấn công collision:

- **Collision** là khi hai dữ liệu khác nhau tạo ra cùng một giá trị hash. Các thuật toán cũ như **MD5** và **SHA-1** có nguy cơ bị tấn công này, nghĩa là kẻ xấu có thể tạo file giả mạo có cùng hash với file gốc.
- Ví dụ: MD5 hiện nay gần như không được dùng trong bảo mật vì đã bị phá vỡ. SHA-256 an toàn hơn, nhưng vẫn có hạn chế về xác thực nguồn gốc.

#### 3. Không ngăn chặn giả mạo:

- Nếu kẻ tấn công thay đổi dữ liệu và tính toán lại hash, họ có thể gửi cả dữ liệu giả và hash mới cho bạn. Bạn không có cách nào biết liệu hash đó có đáng tin hay không, vì hash không gắn với danh tính người gửi.

## 2. Digital Signature khắc phục những nhược điểm nào của Hash?

**Chữ ký số (Digital Signature)** là một kỹ thuật kết hợp **hash** với **mã hóa khóa công khai - khóa bí mật** (như RSA) để vừa đảm bảo **toàn vẹn dữ liệu** vừa **xác thực nguồn gốc**. Dưới đây là cách chữ ký số khắc phục các hạn chế của hash:

#### 1. Xác thực nguồn gốc:

- Với chữ ký số, người gửi dùng **private key** (khóa bí mật) để ký lên hash của dữ liệu, tạo ra một chữ ký độc nhất. Chỉ người có private key mới tạo được chữ ký này.
- Người nhận dùng **public key** (khóa công khai) để kiểm tra chữ ký. Nếu chữ ký hợp lệ, người nhận biết chắc chắn dữ liệu đến từ người sở hữu private key.
- Ví dụ: Nếu bạn nhận email có chữ ký số từ "[abc@x.com](#)", bạn có thể chắc chắn email đến từ người sở hữu private key của "[abc@x.com](#)", không phải kẻ giả mạo.

#### 2. Ngăn giả mạo:

- Nếu dữ liệu bị thay đổi (dù chỉ một ký tự), hash của dữ liệu sẽ thay đổi, dẫn đến chữ ký không còn hợp lệ khi kiểm tra bằng public key. Kẻ tấn công không thể giả mạo chữ ký vì họ không có private key.

- Điều này khắc phục hạn chế của hash: kẻ xấu không thể gửi dữ liệu giả kèm hash mới để lừa bạn, vì họ không thể tạo chữ ký hợp lệ.

### 3. Tính không thể phủ nhận:

- Vì chữ ký số chỉ có thể được tạo bởi người sở hữu private key, người gửi không thể chối rằng họ không gửi dữ liệu. Điều này rất quan trọng trong các ứng dụng như hợp đồng điện tử hoặc blockchain.
- Ví dụ: Trong blockchain, chữ ký số đảm bảo rằng một giao dịch được tạo bởi đúng người sở hữu ví (wallet), và họ không thể phủ nhận giao dịch đó.

### 3. Tại sao người ta thường dùng Hash để kiểm tra file tải về; còn Digital Signature trong email, chứng chỉ số, blockchain?

#### Hash cho file tải về

- **Mục đích chính:** Kiểm tra **toàn vẹn dữ liệu**, tức là đảm bảo file bạn tải về không bị lỗi hoặc bị sửa đổi trong quá trình truyền tải (do lỗi mạng, virus, hoặc cố ý).
- **Tại sao dùng hash:**
  - **Đơn giản:** Chỉ cần tính hash của file và so sánh với hash được cung cấp trên trang web. Không cần hệ thống khóa phức tạp.
  - **Phù hợp với tình huống công khai:** Các trang web cung cấp phần mềm (như Ubuntu, Python) thường công bố hash (MD5, SHA-256) để người dùng kiểm tra file tải về. Vì file là công khai, không cần xác minh ai gửi.
  - **Nhanh chóng:** Hash dễ tính, dễ kiểm tra, phù hợp với nhu cầu kiểm tra file lớn.
- **Ví dụ:**
  - Bạn tải file cài đặt Python từ python.org. Trang web cung cấp giá trị SHA-256. Bạn tính hash của file tải về và so sánh. Nếu khớp, file an toàn. Nếu không, file có thể bị lỗi hoặc bị sửa.

#### Digital Signature trong email, chứng chỉ số, blockchain

- **Mục đích chính:** Không chỉ kiểm tra **toàn vẹn dữ liệu** mà còn **xác thực nguồn gốc** và đảm bảo **tính không thể phủ nhận**. Những hệ thống này yêu cầu độ bảo mật cao hơn.
- **Tại sao dùng Digital Signature:**
  - **Email:**
    - Cần đảm bảo email đến từ đúng người gửi và nội dung không bị sửa đổi. Chữ ký số giúp bạn biết email thực sự từ "[abc@x.com](#)" và không bị ai đó giả mạo hoặc chỉnh sửa.
    - Ví dụ: Email ngân hàng có chữ ký số để đảm bảo thông báo đến từ ngân hàng thật.
  - **Chứng chỉ số:**

- Dùng để xác minh danh tính của website hoặc phần mềm. Ví dụ: Khi bạn truy cập <https://google.com>, trình duyệt kiểm tra chứng chỉ số (có chữ ký số) để đảm bảo website là Google thật, không phải trang giả mạo.
- Chữ ký số từ các tổ chức đáng tin cậy (như Let's Encrypt) đảm bảo phần mềm hoặc website không bị giả mạo.
- **Blockchain:**
  - Trong blockchain (như Bitcoin, Ethereum), chữ ký số đảm bảo giao dịch được tạo bởi đúng người sở hữu ví và không bị sửa đổi. Điều này giúp ngăn chặn gian lận và đảm bảo tính bảo mật.
  - Ví dụ: Khi bạn gửi 1 Bitcoin, bạn ký giao dịch bằng private key. Mạng blockchain dùng public key để kiểm tra, đảm bảo giao dịch là từ bạn.
- **Ưu điểm của Digital Signature:**
  - Cung cấp bảo mật cao hơn hash, phù hợp với các hệ thống cần xác thực danh tính và ngăn giả mạo.
  - Đảm bảo tính không thể phủ nhận, quan trọng trong các giao dịch pháp lý hoặc tài chính.