HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA AN TOÀN THÔNG TIN

MÔN HỌC: MẬT MÃ HỌC CƠ SỞ

BÁO CÁO BÀI TẬP LỚN

TÌM HIỂU VỀ HÀM BĂM Argon2 VÀ ỨNG DỤNG TRONG XÁC THỰC MẬT KHẨU

Giảng viên: PGS.TS. Đỗ Xuân Chợ

Nhóm môn học: 01

Nhóm bài tập lớn: 6

Thành viên :

Ninh Chí Hướng – B20DCAT094	Hoàng Trung Kiên – B20DCAT098
Phan Minh Tiến – B20DCAT158	Vũ Ngọc Phương – B20DCAT192
Lê Văn Tráng – B20DCAT190	Lại Quốc Đạt – B20DCAT036

Hà Nội – 5/2023

Lời mở đầu

Ngày nay, cùng với sự phát triển liên tục và nhanh chóng của công nghệ thông tin, cả về phần cứng lẫn phần mềm. Hiệu năng xử lý của những chiếc máy tính hoặc siêu máy tính đang ngày càng tăng cao. Tuy nhiên, sự phát triển đó lại đem lại những mối lo ngại trong việc đảm bảo an toàn thông tin nói chung và đối với mã hóa nói riêng. Nhưng kéo theo sự phát triển đó, các giải thuật, phương pháp mã hóa mới vẫn đang liên tục được cải tiến và ra đời thêm nhiều loại khác nhau.

Trong đó Argon2 được coi là một loại hàm băm mạnh mẽ. Argon2 là một hàm băm được thiết kế để đảm bảo tính bảo mật cao trong việc lưu trữ mật khẩu và thông tin nhạy cảm khác. Hàm băm này sử dụng một cơ chế phức tạp để tạo ra các giá trị băm không thể đoán trước được và khó khăn trong việc tìm ra nguồn gốc của chúng. Argon2 được xem là một trong những hàm băm mạnh nhất hiện nay và được sử dụng rộng rãi trong các ứng dụng đòi hỏi tính bảo mật cao như đăng nhập, xác thực người dùng và lưu trữ mật khẩu. Trong đoạn văn này, chúng ta sẽ tìm hiểu thêm về cơ chế hoạt động của Argon2 và cách nó đảm bảo tính bảo mật cho các ứng dụng của bạn.

MỤC LỤC

I. GIỚI THIỆU VỀ HÀM BĂM Argon2

1.1 Giới thiệu, lịch sử hàm băm argon2

Hàm băm Argon2 là một giải thuật băm mật mã nhằm đảm bảo tính bảo mật cao trong quá trình lưu trữ mật khẩu hoặc thông tin nhạy cảm khác. Giải thuật này được phát triển bởi các nhà nghiên cứu bảo mật tại Đại học Luxembourg vào năm 2015 và được chấp nhận là tiêu chuẩn bảo mật mới nhất của các hiệp hội bảo mật như Password Hashing Competition. Argon2 hỗ trợ ba dạng chính là Argon2i và Argon2d, Argon2id, với Argon2i hỗ trợ chống tấn công xâm nhập, tránh được các lỗ hổng của phiên bản trước đó, còn Argon2d tối ưu hoá cho các khả năng chống lại cuộc bẻ khóa GPU, nó truy cập mảng bộ nhớ theo thứ tự phụ thuộc vào mật khẩu, giúp giảm khả năng xảy ra các cuộc tán công đánh đổi bộ nhớ theo thời gian (TMTO). Còn Argon2id là một phiên bản lai. Nó tuân theo cách tiếp cận Argon2i cho nửa đầu vượt qua bộ nhớ và cách tiếp cận Argon2d cho các lượt tiếp theo. Hàm băm Argon2 giúp tăng tính bảo mật của hệ thống và giúp tránh được những cuộc tấn công như tấn công từ điển hoặc rainbow table bởi cách tạo ra một băm mật mã mạnh và khó đoán. Argon2 được thiết kế để hiển thị những ưu điểm vượt trội so với các hệ thống băm mật khẩu truyền thống khác như bcrypt hay PBKDF2 khi đưa ra quyết định sử dụng.

Trong tổng quát, Argon2 là một hàm băm rất mạnh và an toàn, được thiết kế để chống lại các cuộc tấn công bằng cách sử dụng các kỹ thuật như chống tấn công đánh tráo (side-channel attacks) và chống tấn công bằng từ điển (dictionary attacks).

Hình 1.1: logo biểu tượng Argon2

II. CÁCH HOẠT ĐỘNG CỦA Argon2

Argon2 là một thuật toán băm mạnh mẽ và phổ biến để bảo vệ mật khẩu và dữ liệu trong các ứng dụng. Nó được thiết kế để chống lại các cuộc tấn công bằng lực

brute-force và các cuộc tấn công bằng từ điển. Argon2 sử dụng cấu trúc băm dựa trên mạng lưới (sponge-based) và tính toán song song (parallelism) để tăng hiệu

Argon2 cung cấp ba loại chế độ hoạt động khác nhau: Argon2d, Argon2i và Argon2id. Tất cả các chế độ hoạt động đều sử dụng cùng một lõi băm, nhưng khác nhau trong cách thức sử dụng và kết hợp các yếu tố khác để tạo ra giá trị băm cuối cùng.

Argon2 bao gồm hai giai đoạn: giai đoạn tiền xử lý (pre-processing) và giai đoạn băm chính (main hashing). Trong giai đoạn tiền xử lý, hàm băm sẽ xử lý mật khẩu và các thông tin khác để tạo ra các khối 128-bit, gọi là phiên (lane).

Sau khi tiền xử lý hoàn tất, giai đoạn băm chính sẽ bắt đầu. Trong giai đoạn này, các phiên được sử dụng để tạo ra giá trị băm cuối cùng. Việc tạo giá trị băm cuối cùng được thực hiện bằng cách sử dụng hàm băm Blake2b để kết hợp các phiên và các thông tin khác như muối, số lần lặp và độ dài của giá trị băm đầu ra.

Argon2 cũng có thể được định cấu hình để sử dụng các tham số khác nhau để tăng cường an toàn và hiệu suất. Các tham số này bao gồm số lượng phiên, kích thước khối, số lần lặp và độ dài của giá trị băm đầu ra.

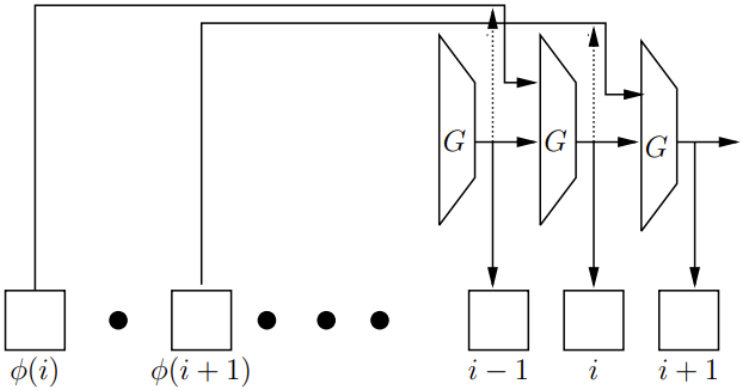
Thuật toán Argon2 nhận đầu vào là một chuỗi ký tự (password) và một số lượng các tham số để tùy chỉnh quá trình băm, bao gồm:

- **Password**: Đây là chuỗi ký tự đầu vào mà bạn muốn băm.
- **Salt**: Một giá trị ngẫu nhiên được sử dụng để đảm bảo rằng cùng một mật khẩu sẽ cho ra các mã băm khác nhau. Salt là một chuỗi ký tự duy nhất cho mỗi bản gốc của mật khẩu.
- **Memory Size**: Đây là một tham số quan trọng quyết định khối lượng bộ nhớ sử dụng bởi thuật toán. Nó giúp khó khăn hơn trong việc tấn công brute-force bằng cách yêu cầu nhiều bộ nhớ hơn để tính toán.
- **Iterations (t)**: Số lượng lần lặp lại (iterations) trong quá trình băm. Giá trị này cũng tăng độ khó cho các cuộc tấn công brute-force.
- **Parallelism (p)**: Số lượng luồng đồng thời được sử dụng trong quá trình băm. Nó tăng tốc độ tính toán trên các hệ thống đa nhân.
- **Output Size**: Kích thước mã băm đầu ra

Nguyên lý thuật toán : Argon2 sử dụng một hàm băm có tên là Blake2b, là một biến thể của hàm băm Blake2. Cấu trúc chính của thuật toán Argon2 bao gồm ba giai đoạn chính: khởi tạo, xử lý và trả về kết quả.

Dưới đây là mô tả cụ thể về từng giai đoạn:

- Khởi tạo: Trước hết, Argon2 chuyển đổi mật khẩu thành một chuỗi byte và tạo ra một muối (salt) ngẫu nhiên. Salt là một chuỗi ngẫu nhiên có kích thước cố định được sử dụng để ngăn chặn cuộc tấn công từ điển (dictionary attack) và tăng tính ngẫu nhiên của quá trình băm.



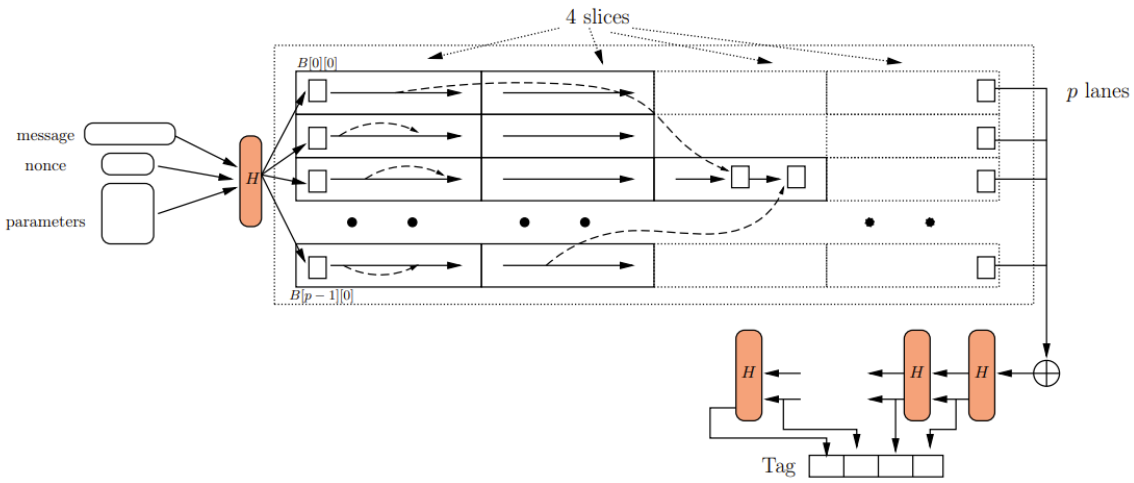
Hình 2.1 : Chế độ hoạt động argon2 không có sự song song

- Xử lý: Sau khi có mật khẩu và Salt, Argon2 bắt đầu giai đoạn xử lý chính. Nó sử dụng một số tham số cấu hình để điều chỉnh độ khó của thuật toán, bao gồm số lượng vòng lặp (iterations), kích thước bộ nhớ (memory size) và kích thước output (output size).

+Argon2 chia mật khẩu và salt thành các khối và tiến hành xử lý song song trên chúng.bh b

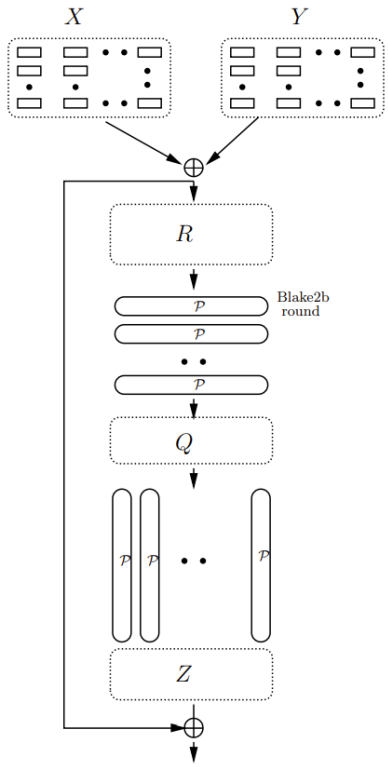
+Mỗi khối được truyền qua một hàm băm Blake2b để tạo ra các giá trị trung gian.

+Các giá trị trung gian được kết hợp với nhau và xử lý tiếp theo để tạo ra kết quả cuối cùng.



Hình 2.2: Argon2 chạy một lần với p lanes và 4 slices

- Trả về kết quả: Kết quả cuối cùng của thuật toán Argon2 là một chuỗi byte được gọi là băm mật khẩu (password hash). Băm mật khẩu này chứa thông tin về salt, các tham số cấu hình và các giá trị trung gian tính toán được sử dụng trong quá trình xử lý. Nó có thể được lưu trữ an toàn trong cơ sở dữ liệu hoặc truyền đi một cách an toàn.



Hình 2.3: Chức năng nén của hàm G

III. ƯU VÀ NHƯỢC ĐIỂM CỦA Argon2

Argon2 có nhiều ưu điểm và một số nhược điểm như sau:

3.1 Ưu điểm:

1. Khả năng chống lại các cuộc tấn công bằng từ điển (dictionary attack). Để chống lại cuộc tấn công này, Argon2 sử dụng một giá trị salt ngẫu nhiên, đảm bảo rằng các giá trị băm khác nhau được tạo ra cho cùng một mật khẩu, ngăn chặn kẻ tấn công sử dụng các bảng giá trị băm (rainbow table) để tấn công. Argon2 còn chống các cuộc tấn công bằng vét cạn (brute-force attack). Argon2 sử dụng các tham số Time Cost và Memory Cost để tăng độ khó của quá trình tính toán giá trị băm, làm giảm hiệu suất của tấn công brute-force.

2. Có khả năng điều chỉnh độ phức tạp tính toán của hàm băm, tùy thuộc vào yêu cầu của ứng dụng và cấu hình phần cứng.

3. Khả năng chống lại các cuộc tấn công bằng sử dụng đa luồng, nghĩa là nó có thể được tính toán trên nhiều bộ xử lý đồng thời, giúp tăng tốc độ tính toán.

4. Được thiết kế để tránh các cuộc tấn công bằng sử dụng bộ nhớ tạm (cache-timing attacks) và các cuộc tấn công bằng sử dụng bộ đệm (cache-based side-channel attacks).

5. Được coi là một trong những hàm băm mạnh nhất hiện nay và được sử dụng rộng rãi trong các ứng dụng đòi hỏi tính bảo mật cao.

3.2 Nhược điểm:

1. Tốc độ tính toán của Argon2 chậm hơn so với một số hàm băm khác như SHA-256 hoặc SHA-3.

2. Cấu hình của Argon2 khá phức tạp, đòi hỏi người dùng phải tìm hiểu và hiểu rõ để đảm bảo tính bảo mật của ứng dụng.

3. Argon2 không thể hoàn toàn ngăn chặn các cuộc tấn công bằng sử dụng botnet, nghĩa là các tấn công được thực hiện bởi hàng loạt các máy tính được kiểm soát từ xa.

IV. SO SÁNH Argon2 VÀ CÁC HÀM BĂM KHÁC

4.1 So sánh Argon2, Bcrypt và SHA256

	Argon2	Bcrypt	SHA-256
Thuật toán	Argon2 là một thuật toán băm mật khẩu dựa trên Blake2 và được thiết kế để làm chậm quá trình băm và tăng độ bảo mật bằng cách sử dụng một lượng lớn bộ nhớ.	Bcrypt sử dụng hàm băm Blowfish và được thiết kế để làm chậm quá trình băm mật khẩu và tăng độ bảo mật bằng cách sử dụng một lượng lớn bộ nhớ.	SHA-256 là một thuật toán băm không có trạng thái, dựa trên hàm băm đệ quy trên cơ sở của lý thuyết thông tin.
Tính bảo mật	Argon2 được thiết kế để chống lại các cuộc tấn công bằng từ điển, tấn công bằng lực brute và tấn công đa nhân.	Bcrypt được thiết kế để chống lại các cuộc tấn công bằng lực brute và các tấn công từ điển.	SHA-256 là một thuật toán băm mạnh và được sử dụng rộng rãi trong các ứng dụng bảo mật, nhưng nó dễ dàng bị tấn công bằng lực brute và tấn công từ điển so với Argon2 và Bcrypt.
Tốc độ băm	Argon2 được thiết kế để làm chậm quá trình băm bằng cách sử dụng một lượng lớn bộ nhớ. Chậm hơn so với SHA-256 và Bcrypt.	Bcrypt cũng được thiết kế để làm chậm quá trình băm bằng cách sử dụng phương pháp "phân tán" và một tham số gọi là "factors". Tuy nhiên, tốc độ băm của Bcrypt nhanh hơn so với Argon2 và chậm hơn so với SHA-256.	SHA-256 là một thuật toán băm nhanh và có thể băm một lượng lớn dữ liệu một cách nhanh chóng. Tốc độ băm của SHA-256 nhanh hơn so với Argon2 và Bcrypt.
Sử dụng bộ nhớ	Argon2 được thiết kế để sử dụng một lượng lớn bộ nhớ, để làm chậm quá trình băm và tăng độ bảo mật. Sử dụng kỹ thuật "phân tán bộ nhớ". Nên dùng nhiều bộ nhớ hơn SHA-256 và Bcrypt.	Bcrypt cũng sử dụng một vài lượng bộ nhớ để lưu trữ, tuy nhiên lượng bộ nhớ được sử dụng ít hơn so với Argon2 và nhiều hơn SHA-256.	SHA-256 không đòi hỏi một lượng lớn bộ nhớ để hoạt động. Sử dụng ít bộ nhớ hơn Argon2 và Bcrypt

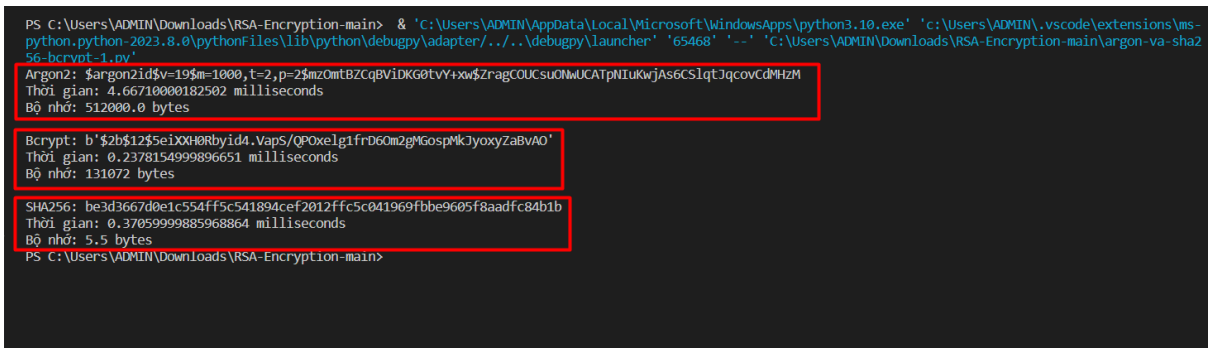
Hiệu suất	Argon2 có thể chậm hơn SHA-256 trong việc tính toán hash mật khẩu, do sử dụng các kỹ thuật bảo mật phức tạp hơn. Tuy nhiên, việc sử dụng các tham số hiệu suất của Argon2 cho phép điều chỉnh hiệu suất và độ an toàn của thuật toán.	Trong trường hợp thời gian hash dưới 1s, bcrypt có thể hoạt động nhanh hơn Argon2. Nếu tính an toàn là ưu tiên hàng đầu, Argon2 là lựa chọn tốt hơn. Tuy nhiên, nếu hiệu suất là ưu tiên hàng đầu và thời gian hash dưới 1s, Bcrypt có thể là lựa chọn tốt hơn.	SHA-256 có hiệu suất cao hơn so với Argon2 và Bcrypt, nhưng không có khả năng điều chỉnh độ an toàn và hiệu suất của thuật toán.
Kích thước đầu ra	Argon2 cho phép tạo ra các chuỗi băm (hash) với độ dài tùy ý.	Bcrypt luôn tạo ra các chuỗi băm có độ dài cố định là 192 bit.	SHA-256 luôn tạo ra các chuỗi băm có độ dài cố định là 256 bit.
Độ tin cậy	Argon2 được coi là một thuật toán hash mật khẩu đáng tin cậy hơn so với SHA-256 và Bcrypt.	Bcrypt được coi là một thuật toán hash mật khẩu đáng tin cậy. Tuy nhiên độ tin cậy của Bcrypt không bằng so với Argon2.	SHA-256 vẫn được coi là một thuật toán hash mật khẩu đáng tin cậy.

-Code so sánh Argon2 với Bcrypt và SHA-256

```
# Argon2
_argon = argon2.using(
    salt_len=16, # độ dài của muối
    time_cost=2, # số lần lặp
    memory_cost=1000, # bộ nhớ cần thiết
    parallelism=2, # số luồng thực hiện
    hash_len=32, # độ dài chuỗi đầu ra
)
start = timeit.default_timer()
_argon_hash = _argon.hash(password)
end = timeit.default_timer()
argon2_time = (end - start) * 1000
argon2_memory = len(_argon_hash)
argon2_memory = 1000 * 1024 * 2 /4
# Bcrypt
start = timeit.default_timer()
_bcrypt_hash = bcrypt.hashpw(password, bcrypt.gensalt(16)) #gensalt(): muối mặc định là 12 và có thể chỉnh độ dài của muối tùy ý "gensalt(16)"
end = timeit.default_timer()
bcrypt_time = (end - start)
bcrypt_memory = 2 ** ( 12+5)

# SHA256
#Tạo một chuỗi muối ngẫu nhiên bằng cách sử dụng os.urandom()
salt = os.urandom(16) #Lấy độ dài của muối là 16 bytes, có thể chỉnh tùy ý
salt_encoded = base64.b64encode(salt)
salted_password = salt + password

start = timeit.default_timer()
_sha256_hash = hashlib.sha256(salted_password).hexdigest()
end = timeit.default_timer()
sha256_time = (end - start) * 1000
sha256_memory = (len(password) / 64 ) * 32
```



⇨ Kết luận: Argon2, SHA-256 và Bcrypt đều là các thuật toán băm mật khẩu được sử dụng rộng rãi trong các ứng dụng bảo mật.

-SHA-256 là một thuật toán băm mật khẩu đơn giản và nhanh chóng, phù hợp để sử dụng trong các trường hợp đòi hỏi tính nhanh và sử dụng ít bộ nhớ nhưng không đảm bảo độ bảo mật cao so với Argon2 và Bcrypt.

-Bcrypt là một thuật toán băm mật khẩu, tốc độ băm của Bcrypt nhanh hơn so với Argon2 và chậm hơn so với SHA-256. Bcrypt cũng sử dụng ít bộ nhớ hơn so với Argon2.

-Argon2 là một thuật toán băm mật khẩu được coi là nâng cao hơn so với Bcrypt và SHA-256, vì nó sử dụng các kỹ thuật mới nhất để tăng độ bảo mật và khó khăn hơn cho các cuộc tấn công.

- Việc lựa chọn thuật toán băm mật khẩu phù hợp phụ thuộc vào yêu cầu của ứng dụng và mức độ bảo mật được yêu cầu. Trong các trường hợp đòi hỏi độ bảo mật cao hơn, Argon2 là lựa chọn tốt nhất. Tuy nhiên, trong những trường hợp đòi hỏi tính nhanh và độ bảo mật ở mức trung bình, Bcrypt hoặc SHA-256 có thể là những lựa chọn phù hợp hơn.

V. ỨNG DỤNG CỦA Argon2

Với khả năng chống tấn công tuyệt vời Argon2 được ứng dụng rộng rãi trong thực tế, đặc biệt là ứng dụng trong việc bảo vệ mật khẩu

1. Quản lý mật khẩu

Khi lưu trữ mật khẩu, vấn đề cần quan tâm bậc nhất là việc mật khẩu không thể bị tìm ra bằng 1 cách thức tấn công nào. Argon2 với khả năng bảo mật thuộc loại tốt nhất hiện nay hoàn toàn đáp ứng được yêu cầu này. Trong thực tế ta hoàn toàn có thể tự sử dụng argon2 để băm và lưu trữ mật khẩu phục vụ cho mục đích cá nhân hoặc tổ chức. Ta có thể dễ dàng bắt gặp các hệ thống quản lý mật khẩu đang sử dụng Argon2 như LastPass và KeePass

+ LastPass là 1 trình quản lý mật khẩu trực tuyến phổ biến nhất được tích hợp trong đa số trình duyệt hiện nay

+ KeePass là trình quản lý mật khẩu hỗ trợ chủ yếu cho Window, phục vụ cho việc lưu trữ và quản lý mật khẩu

2. Bảo vệ dữ liệu

Mặc dù thời gian băm và xác thực khá chậm so với các hàm băm hoặc các thuật toán mã hóa khác, Argon vẫn được sử dụng để bảo vệ những dữ liệu quan trọng nhằm chống lại các cuộc tấn công đánh cắp dữ liệu trên máy chủ hoặc các máy cá nhân. Việc băm dữ liệu bằng Argon2 có thể nói là bảo đảm an toàn gần như tuyệt đối, việc đọc hay ghi dữ liệu được mã hóa bằng Argon2 (được cấu hình chính xác) là chưa được ghi nhận.

3. Xác thực

Đôi khi Argon2 còn được sử dụng như hỗ trợ cho việc xác thực người dùng. Thông tin của người dùng trong hệ thống được lưu lại và băm bằng Argon2. Việc này làm giảm tối đa tấn công chiếm quyền điều khiển ngăn chặn hacker truy cập vào hệ thống khi chưa được cấp phép. Một số giao thức bảo mật đang sử dụng Argon2 có thể kể đến như OAuth và OpenID Connect

4. Blockchain

Công nghệ chuỗi khối Blockchain không còn quá xa lạ với chúng ta trong thời điểm hiện tại. Argon2 được sử dụng để mã hóa khối trong Blockchain đảm bảo tính toàn vẹn của hệ thống chống tấn công sửa đổi ở mức độ cao. Tiêu biểu có thể kể đến trong dự án Moreno hoặc ít phổ biến hơn là Wownero và Koto. Do tính năng khó bị tấn công, có thể trong tương lai sẽ có nhiều dự án Blockchain sử dụng Argon2 để mã hóa các khối

VI. Demo

Sử dụng Argon2 trong việc xác thực mật khẩu và đăng nhập

```
// password_hash('somepassword', PASSWORD_ARGON2I, ['memory_cost' => 2048, 'time_cost' => 4, 'threads' => 3]);
$hash = password_hash($password, PASSWORD_ARGON2I);
```

Đoạn code có chức năng băm mật khẩu bằng hàm băm Agron2i (các tham số đầu vào có thể thay đổi như đoạn code trong chú thích)

Giá trị băm của mật khẩu sẽ lưu trong CSDL

Để xác minh mật khẩu do người dùng từ xa cung cấp, bạn cần sử dụng hàm password_verify()

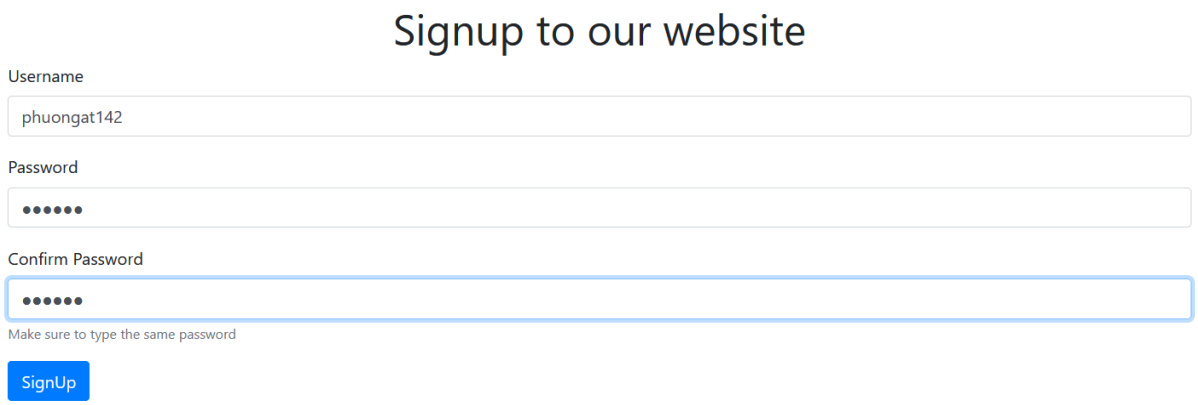
password_verify() nhận hai đối số:

- mật khẩu bạn cần xác minh, làm đối số đầu tiên
- hàm băm từ *password_hash()* của mật khẩu ban đầu, làm đối số thứ hai

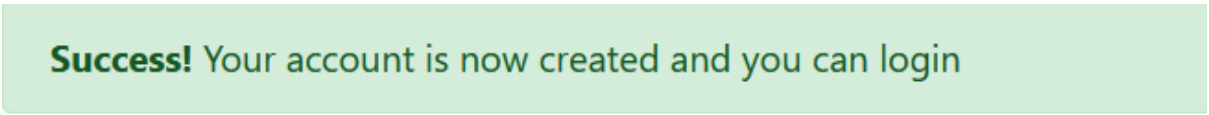
Nếu mật khẩu đúng, *password_verify()* trả về **true**

```
if (password_verify($password, $row['password'])){
    $login = true;
```

Bây giờ ta sẽ tạo tài khoản người dùng:



→ hệ thống thông báo đăng kí thành công



→ Giá trị băm được lưu trong CSDL

0	tienat	$argon2i$v=19$m=65536,t=4,p=1$MXE3DFGc1FJWU1yTFBmQw$y6ZE1gVSCdnyr8YMNOlmKc2ZmDra1g7yGVMC2PmY5FA	2023-05-11 07:49:05
0	phuongat142	$argon2i$v=19$m=65536,t=4,p=1$SG5pZkloRE9rci5xL3RVTg$S0xNCj+bV2uZfpFJ9S2xrQon3+hRy5xKGYvlmiZjoWg	2023-05-17 20:56:08

Sau khi người dùng đăng nhập và nhập mật khẩu, giá trị salt được truy xuất và sử dụng để tái tạo lại giá trị băm từ mật khẩu đăng nhập. Sau khi giá trị băm được tái tạo, nó được so sánh với giá trị băm được lưu trữ trong cơ sở dữ liệu. Nếu hai giá trị băm khớp nhau, mật khẩu được coi là đúng và người dùng được cho phép truy cập vào hệ thống.

Tài liệu tham khảo

1. Argon2: the memory-hard function for password hashing and other applications (https://github.com/P-H-C/phc-winner-argon2/blob/master/argon2-specs.pdf)

2. phc-winner-argon2 (https://github.com/P-H-C/phc-winner-argon2)

3. Cómo hacer Secure Password Hashing - Argon2 - SHA256 NO es seguro (https://youtu.be/Ku0Hh5W8z44)