Implementasi Argon2 sebagai Mekanisme Password Hashing yang Tahan terhadap Serangan Brute Force dan GPU Acceleration

Mohammad Nugraha Eka Prawira - 13522001[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13522001@stei.itb.ac.id

***Abstrak*—Keamanan password menjadi isu utama dalam sistem autentikasi modern, terutama di era komputasi GPU yang mampu melakukan brute-force dengan kecepatan sangat tinggi. Mekanisme hashing tradisional seperti SHA-256, PBKDF2, dan bcrypt mulai menunjukkan kelemahan terhadap serangan berbasis parallelism GPU. Argon2, pemenang Password Hashing Competition (PHC) tahun 2015, dirancang sebagai algoritma password hashing yang bersifat memory-hard dan tahan terhadap percepatan GPU maupun ASIC.**

Makalah ini mengimplementasikan Argon2id sebagai mekanisme password hashing berbasis Python, melakukan pengujian parameter memory cost, time cost, dan parallelism, serta menganalisis ketahanannya terhadap brute-force dan GPU acceleration. Hasil eksperimen menunjukkan bahwa peningkatan memory cost memberikan dampak signifikan dalam memperlambat brute-force, menjadikan Argon2 solusi ideal untuk aplikasi web modern.

***Kata Kunci*—Argon2, Password Hashing, Memory-Hard, GPU Acceleration, Brute Force Attack.**

I. PENDAHULUAN

Keamanan password merupakan elemen fundamental dalam sistem autentikasi pada hampir seluruh layanan digital, termasuk aplikasi web, layanan cloud, perbankan digital, hingga platform media sosial. Meskipun berbagai metode autentikasi modern telah bermunculan, seperti autentikasi dua faktor (2FA) dan autentikasi biometrik, password tetap menjadi mekanisme yang paling umum digunakan. Namun, penggunaan password sebagai satu-satunya lapisan pengamanan telah terbukti rentan, terutama ketika pengguna memilih password yang lemah atau ketika sistem backend tidak menggunakan teknik hashing yang aman. Masalah ini diperparah oleh pesatnya perkembangan perangkat keras komputasi modern, terutama GPU (Graphics Processing Unit), yang mampu menjalankan jutaan hingga miliaran operasi hashing secara paralel dalam hitungan detik.

Dalam beberapa tahun terakhir, serangan credential stuffing, dictionary attack, dan brute-force mengalami peningkatan signifikan, karena penyerang dapat memanfaatkan GPU dan ASIC (Application-Specific Integrated Circuit) untuk mempercepat proses cracking password. Algoritma hashing tradisional seperti MD5, SHA-1, dan SHA-256 sangat cepat di GPU, sehingga tidak lagi memadai untuk melindungi password pengguna. Bahkan algoritma yang secara historis dianggap aman seperti PBKDF2 dan bcrypt mulai menunjukkan keterbatasan karena tidak didesain dengan mempertimbangkan kemampuan GPU modern. GPU memiliki ribuan core paralel, dan dalam beberapa pengujian, sebuah GPU kelas high-end seperti NVIDIA RTX 4090 mampu melakukan hashing SHA-256 hingga lebih dari 150 gigahash per detik. Artinya, password dengan panjang 6 karakter dapat diuji seluruh kombinasinya dalam hitungan menit. Kondisi ini menjadikan kebutuhan terhadap algoritma password hashing yang GPU-resistant semakin mendesak.

Sebagai respons terhadap kelemahan algoritma hashing tradisional, Password Hashing Competition (PHC) pada tahun 2015 melahirkan Argon2 sebagai pemenang kompetisi, menjadikannya standar baru untuk password hashing modern. Argon2 memiliki tiga varian: Argon2d, Argon2i, dan Argon2id. Di antara ketiganya, Argon2id direkomendasikan oleh RFC 9106 sebagai mekanisme password hashing yang aman terhadap serangan side-channel maupun brute-force berbasis GPU. Keunggulan utama Argon2 dibandingkan algoritma lain adalah sifatnya sebagai memory-hard function, yaitu algoritma yang memaksa penggunaan memori dalam jumlah besar pada setiap proses hashing. Pendekatan ini sangat efektif untuk mengurangi efisiensi GPU, karena GPU memiliki banyak core tetapi memiliki keterbatasan memori dan bandwidth per core, sehingga tidak dapat menjalankan ribuan proses hashing secara paralel seperti pada algoritma hashing tradisional.

Argon2 tidak hanya mengandalkan konsumsi memori besar, tetapi juga memberikan fleksibilitas dalam konfigurasi parameter seperti time cost, memory cost, dan degree of parallelism, sehingga dapat disesuaikan dengan kebutuhan sistem. Parameter tersebut memungkinkan pengembang meningkatkan biaya komputasi hashing

sesuai tingkat keamanan yang diinginkan tanpa mengorbankan performa sistem secara keseluruhan. Fleksibilitas ini menjadikan Argon2 relevan bagi berbagai platform, mulai dari aplikasi web berskala kecil hingga sistem autentikasi enterprise yang membutuhkan tingkat perlindungan tinggi terhadap serangan offline password cracking.

Walaupun Argon2 menawarkan solusi yang kuat, implementasinya dalam aplikasi nyata sering kali menimbulkan pertanyaan praktis: parameter manakah yang paling efektif dalam meningkatkan keamanan? Bagaimana performa Argon2 di lingkungan CPU modern? Seberapa efektifkah Argon2 dalam menahan brute-force dibandingkan algoritma seperti PBKDF2 atau bcrypt? Dan bagaimana Argon2 mengatasi ancaman brute-force GPU yang semakin cepat setiap tahun? Pertanyaan-pertanyaan ini menjadi dasar motivasi dari makalah ini.

Oleh karena itu, penelitian ini bertujuan untuk mengimplementasikan Argon2id sebagai mekanisme password hashing, mengevaluasi performanya dengan menguji berbagai kombinasi parameter, serta menganalisis ketahanannya terhadap brute-force berbasis CPU maupun GPU. Pengujian dilakukan menggunakan bahasa Python dengan pustaka argon2-cffi, yang mendukung implementasi Argon2 resmi. Selain itu, makalah ini juga menyertakan simulasi brute-force sederhana untuk menunjukkan efektivitas Argon2 dalam memperlambat proses cracking, sekaligus membandingkannya dengan pendekatan hashing tradisional. Dengan demikian, makalah ini diharapkan memberikan pemahaman menyeluruh mengenai alasan Argon2 menjadi standar password hashing modern dan bagaimana implementasinya secara tepat dapat meningkatkan keamanan sistem autentikasi digital di berbagai platform.

II. TEORI DASAR

Keamanan password tidak hanya bergantung pada penggunaan kata sandi yang kuat, tetapi juga pada bagaimana sistem menyimpannya. Penyimpanan password yang aman harus menggunakan teknik password hashing yang dirancang untuk memperlambat penyerang, terutama ketika terjadi kebocoran database. Bab ini membahas teori dasar mengenai password hashing, ancaman brute-force modern, konsep memory-hard function, serta prinsip kerja Argon2 sebagai algoritma password hashing yang direkomendasikan secara internasional.

A. Password Hashing & Key Derivation Function (KDF)

Password hashing adalah proses mengubah password menjadi representasi kriptografi yang tidak dapat dibalik (one-way). Tidak seperti enkripsi, hashing tidak memiliki proses dekripsi; tujuan hashing password adalah memastikan bahwa bahkan jika database bocor, penyerang tidak dapat memperoleh password asli secara langsung.

Namun, hashing saja tidak cukup. Algoritma hashing umum seperti SHA-256 atau SHA-3 terlalu cepat dan mudah dieksploitasi oleh penyerang menggunakan GPU. Oleh karena itu, sistem autentikasi modern menggunakan Key Derivation Function (KDF) khusus yang:

1. Lambat secara komputasi (computationally expensive) → membuat brute-force menjadi tidak efisien.
2. Menggunakan salt acak → mencegah serangan rainbow table.
3. Dapat dikonfigurasi parameternya → agar dapat menyesuaikan dengan kemampuan hardware.

Contoh KDF tradisional adalah PBKDF2, bcrypt, dan scrypt. Namun ketiganya masih memiliki kelemahan signifikan terhadap GPU, seperti yang akan dijelaskan pada bagian berikutnya.

B. Ancaman Brute-Force Modern dan GPU Acceleration

Brute-force adalah upaya mencoba seluruh kemungkinan password hingga menemukan kecocokan. Serangan ini menjadi semakin berbahaya dengan berkembangnya GPU, yang secara arsitektural memiliki ribuan core paralel sehingga dapat melakukan hashing dalam jumlah besar secara simultan.

Sebagai ilustrasi kemampuan GPU dalam brute-force:

- NVIDIA RTX 4090 dapat mencapai > 150 gigahash/s untuk hashing SHA-256.
- Hashcat melaporkan kemampuan PBKDF2-HMAC-SHA1 1.000 iterasi sebesar 20–30 juta hash per detik.
- Bcrypt cost faktor 10 dapat diproses ribuan hash per detik di GPU.

Sementara CPU hanya mampu melakukan hashing dalam jumlah ribuan atau ratusan ribu per detik, GPU mampu melakukan ribuan kali lebih cepat. Ini berarti password yang hanya terdiri dari:

- 6 karakter huruf kecil → dapat habis dicoba GPU dalam < 1 menit.
- 8 karakter alfanumerik → dapat habis dicoba dalam hitungan jam.

Hal ini menunjukkan bahwa desain password hashing harus mempertimbangkan ancaman hardware modern, bukan hanya ancaman teoretis. Algoritma seperti MD5, SHA-1, atau SHA-256 sangat insecure untuk password karena kecepatannya yang ekstrem di GPU.

C. Konsep Memory-Hard Function

Argon2 dirancang dengan prinsip memory-hard function, yaitu fungsi yang membutuhkan jumlah memori besar dalam proses hashing. Kebutuhan memori ini memaksa penyerang untuk menyediakan

memori per thread brute-force, sehingga GPU tidak dapat memanfaatkan core paralelnya secara maksimal.

Secara umum:

- GPU memiliki ribuan core, tetapi memori yang tersedia per core sangat terbatas.
- Dengan memaksa penggunaan memori ≥64 MB per hash, Argon2 membuat GPU tidak dapat menjalankan ribuan thread hashing sekaligus.
- Akibatnya, GPU kehilangan keunggulan paralelnya, dan menjadi hampir selevel dengan CPU untuk Argon2.

Memory-hardness inilah yang menjadikan Argon2 jauh lebih kuat dibandingkan PBKDF2 atau bcrypt.

D. *Argon2 (Password Hashing Modern)*

Argon2 diperkenalkan pada 2015 sebagai pemenang Password Hashing Competition (PHC). Terdapat tiga varian Argon2:

1. Argon2d
   Menggunakan data-dependent memory access sehingga sangat cepat tetapi rentan side-channel attack. Cocok untuk aplikasi tertentu tetapi tidak untuk password hashing.
2. Argon2i
   Menggunakan data-independent memory access, aman terhadap side-channel tetapi sedikit lebih lambat.
3. Argon2id
   Menggabungkan kedua varian di atas, aman terhadap side-channel dan brute-force GPU. Direkomendasikan oleh RFC 9106 sebagai standard untuk password hashing.

Argon2 memiliki tiga parameter utama:

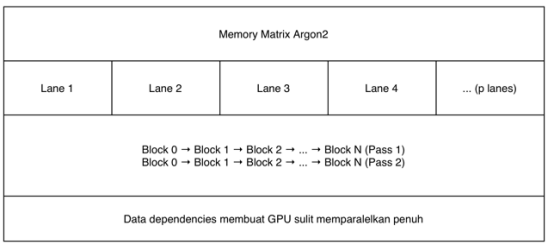
- Memory Cost (m): jumlah memori (dalam KB) yang digunakan hashing.
- Time Cost (t): jumlah iterasi yang dilakukan (jumlah passes).
- Parallelism (p): jumlah lane atau thread hashing yang digunakan.

Dengan meningkatkan memori dan waktu, keamanan dapat disesuaikan mengikuti kebutuhan sistem.

E. *Struktur Internal Argon2*

Argon2 bekerja dengan membentuk matrix memori besar yang diisi oleh beberapa iterasi dengan menggunakan blok Blake2b. Memori dibagi menjadi beberapa lane, dan setiap lane mengandung blok-blok yang harus diproses secara berurutan dan saling bergantung. Desain ini menurunkan kemampuan parallelism GPU karena setiap blok membutuhkan hasil blok sebelumnya.

Placeholder Gambar 1: Representasi Memory Matrix Argon2



Gambar 2.1 Representasi Memory Matrix Argon2

F. *Perbandingan Algoritma Password Hashing*

Tabel berikut membandingkan beberapa algoritma password hashing populer.

Algoritma	Memory-Hard	GPU Resistant	Tahun	Catatan
PBKDF2	Tidak	Lemah	2000	Mudah dipercepat GPU
bcrypt	Rendah	Lemah – Sedang	1999	Memori hanya 4 KB
scrypt	Ya	Sedang	2009	Memory-hard tetapi kurang efisien
Argon2id	**Ya**	**Sangat kuat**	**2015**	Standar modern RFC 9106

Tabel 2.1 Perbandingan Keamanan Password Hashing

Dapat dilihat bahwa Argon2id menempati posisi terbaik dalam keamanan modern.

G. *Aplikasi Parameter Argon2id*

RFC 9106 (2021) memberikan rekomendasi konfigurasi minimal Argon2id:

- Memory cost: 64 MB atau lebih
- Time cost: minimal 2
- Parallelism: 1–4

RFC ini menyatakan bahwa Argon2id adalah pilihan yang paling aman untuk password hashing pada aplikasi komersial maupun kelembagaan.

III. Analisis Permasalahan dan Desain

Keamanan mekanisme autentikasi tidak hanya ditentukan oleh kebijakan pengguna dalam membuat password, tetapi juga oleh bagaimana sistem backend mengelola, menyimpan, dan memverifikasi password tersebut. Pada bab ini, dilakukan analisis sederhana mengenai permasalahan yang ingin diatasi, tujuan yang ingin dicapai dalam implementasi Argon2 sebagai mekanisme password hashing, serta rancangan sistem yang diterapkan untuk menghasilkan solusi yang aman dan efisien. Desain mencakup alur kerja hashing, alur verifikasi password, pemilihan parameter Argon2id, dan strategi yang digunakan untuk meningkatkan resistansi terhadap brute-force berbasis CPU maupun GPU.

A. *Analisis Permasalahan*

Permasalahan utama yang menjadi latar belakang penelitian ini adalah kerentanan mekanisme password hashing tradisional terhadap brute-force modern. Ketika server atau database mengalami kebocoran, penyerang memperoleh hash dari password pengguna. Pada titik ini, seluruh keamanan bergantung pada algoritma password hashing yang digunakan.

Masalah yang diidentifikasi meliputi:

1. Hashing cepat mudah di-bruteforce GPU

   Algoritma seperti SHA-256 sangat tidak cocok digunakan untuk password hashing karena GPU dapat menghitung miliaran hash per detik. Database yang bocor dapat menyebabkan seluruh akun pengguna dikompromikan dalam hitungan jam.
2. PBKDF2 dan bcrypt mulai tidak relevan pada era GPU

   PBKDF2 berbasis iterasi CPU-bound sehingga tetap dapat diakselerasi GPU. Bcrypt menggunakan memori kecil (~4 KB), sehingga masih dapat diparalelkan di GPU.
3. Pengguna cenderung menggunakan password lemah

   Kombinasi password pendek, mudah ditebak, dan hashing tidak aman menjadikan serangan brute-force semakin efektif.
4. Serangan offline tidak dapat dihentikan oleh mekanisme aplikasi

   Setelah database bocor, rate-limiting, CAPTCHA, dan monitoring login tidak lagi relevan. Password hashing adalah satu-satunya lapisan pertahanan.
5. Tidak semua implementasi Argon2 aman

   Argon2 sangat fleksibel; parameter yang salah dapat menghilangkan keunggulan keamanannya. Banyak implementasi menggunakan memory cost terlalu kecil (≤ 32 MB), padahal GPU dapat menjalankan ribuan thread dengan memori sekecil itu.

Permasalahan-permasalahan di atas menegaskan bahwa dibutuhkan algoritma password hashing yang tidak hanya lambat, tetapi juga memory-hard, sehingga brute-force GPU menjadi tidak efisien.

B. *Tujuan Implementasi Argon2id*

Berdasarkan analisis di atas, tujuan dari desain sistem pada makalah ini adalah:

1. Mengimplementasikan Argon2id sebagai mekanisme password hashing standar modern yang memenuhi rekomendasi RFC 9106.
2. Membuat sistem hashing yang adaptif, yang memungkinkan penyesuaian parameter keamanan sesuai kebutuhan hardware server.
3. Meningkatkan ketahanan terhadap brute-force offline, baik pada CPU maupun GPU.
4. Menyediakan desain verifikasi password yang aman, termasuk validasi salt dan parameter hashing.
5. Menentukan parameter Argon2id yang optimal berdasarkan evaluasi performa dan keamanan.

Tujuan akhir adalah menghasilkan sistem password hashing yang sulit ditembus walaupun seluruh database hash dicuri penyerang.

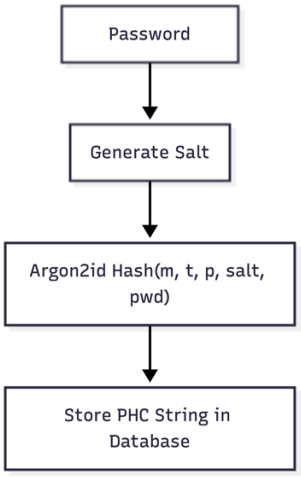
C. *Desain Sistem Hashing Password*

Desain sistem hashing password dijelaskan dalam dua bagian utama: proses hashing dan proses verifikasi. Keduanya saling berkaitan dan menentukan aspek keamanan sistem autentikasi.

1. Alur Hashing Password

   Ketika pengguna membuat akun atau mengubah password, sistem melakukan serangkaian langkah:
   1) Pengguna memasukkan plaintext password.
   2) Sistem menghasilkan salt acak unik (biasanya 16–32 byte).
   3) Argon2id dijalankan dengan parameter (memory cost, time cost, parallelism).
   4) Argon2 menghasilkan hash yang berisi informasi:
      - varian algoritma,
      - versi Argon2,
      - parameter hashing,
      - salt,
      - hash akhir.
   5) Hasil akhir disimpan dalam format standar PHC string.

   Format PHC ini memungkinkan sistem untuk melakukan migrasi parameter hashing di masa depan tanpa memengaruhi password pengguna.



Gambar 3.1 Alur Sistem Hashing Password Argon2id

2. Alur Verifikasi Password

Ketika pengguna login, sistem perlu memvalidasi password dengan aman:

1) Ambil hash (PHC string) dari database.
2) Ekstrak parameter hashing, salt, dan hash hasil.
3) Jalankan Argon2id dengan password input dan parameter yang sama.
4) Bandingkan hasil hashing ulang dengan hash tersimpan.
5) Jika sesuai → password benar; jika tidak → login ditolak.

Proses ini memastikan bahwa password tidak pernah disimpan plaintext dan tidak pernah dikirim dalam bentuk hash yang rawan collision.

3. Pemilihan Parameter Argon2id

Pemilihan parameter memegang peran sangat penting karena Argon2 bersifat adaptif.

- Memory Cost (m): parameter terpenting
  Semakin besar memori, semakin sulit GPU menjalankan ribuan instance hashing.
- Time Cost (t): jumlah iterasi
  Menambah waktu hashing, tetapi tidak terlalu memengaruhi GPU resistance seperti memory cost.
- Parallelism (p): jumlah lane
  Biasanya disesuaikan dengan jumlah core CPU server agar hashing efisien.

Menurut RFC 9106:

Level Keamanan	Memory Cost	Time Cost	Parallelism
Minimum	64 MB	2	1-2
Recommended	128 MB	3	2-4
High Security	256 MB	4	2-4

Tabel 3.1 RFC 9106

Parameter yang dipilih dalam eksperimen makalah ini mencakup tiga kategori: rendah, sedang, dan tinggi.

D. Rancangan Keamanan Sistem

Selain hashing password, terdapat aspek keamanan tambahan:

1. Salt harus unik dan acak
   Mencegah penggunaan rainbow table dan korelasi antar password.
2. PHC string memudahkan migrasi algoritma
   Sistem tetap aman meski parameter diubah pada update versi.
3. Tidak menyimpan parameter secara terpisah
   Seluruh parameter harus berada dalam satu string agar tidak rusak akibat konfigurasi server.
4. Password harus dibatasi panjangnya
   Untuk menghindari serangan DoS melalui password berukuran ekstrem.
5. Migrasi bertahap ke Argon2id
   Jika sistem sebelumnya memakai PBKDF2, maka password pengguna dapat di-hash ulang ketika login berikutnya.

Keseluruhan desain ini memberikan landasan kuat untuk implementasi Argon2id dalam skala produksi.

E. Rancangan Keamanan Sistem

Desain Argon2id secara eksplisit melawan kemampuan GPU:

- GPU memiliki banyak core tetapi memori per core kecil.
- Argon2id memaksa penggunaan memori tinggi sehingga GPU tidak mampu menjalankan ribuan proses hashing paralel.

Dengan demikian:

- GPU kehilangan keunggulan paralelnya,
- brute-force melambat secara drastis (hingga ribuan kali),
- password menjadi jauh lebih aman meski database bocor.

IV. IMPLEMENTASI

A. Lingkungan Implementasi dan Alat yang Digunakan

Implementasi sistem password hashing menggunakan bahasa pemrograman Python. Pemilihan Python didasarkan pada kemudahan pengembangan, ketersediaan pustaka kriptografi yang matang, serta kemampuannya untuk digunakan sebagai prototipe sistem autentikasi backend. Library utama yang digunakan adalah argon2-cffi, yang merupakan binding resmi Argon2 untuk Python dan telah mengikuti spesifikasi Argon2 hasil Password Hashing Competition (PHC).

Lingkungan implementasi yang digunakan meliputi:

- Bahasa pemrograman: Python 3.10
- Library kriptografi: argon2-cffi
- Sistem operasi: macOS
- Tidak menggunakan akselerasi GPU secara langsung pada implementasi

Library argon2-cffi dipilih karena:

1. Mengimplementasikan Argon2id secara resmi dan sesuai standar.

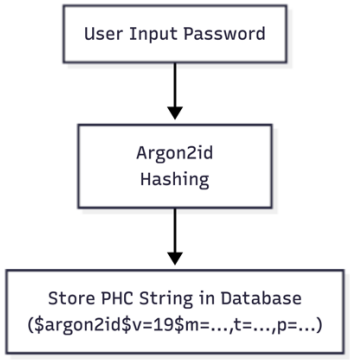
2. Mendukung format output PHC string.
3. Menyediakan mekanisme verifikasi password yang aman.
4. Digunakan secara luas pada aplikasi produksi.

B. *Arsitektur Implementasi Sistem*

Implementasi sistem dibagi menjadi dua fungsi utama:

1. Fungsi hashing password, yang digunakan saat pendaftaran atau perubahan password.
2. Fungsi verifikasi password, yang digunakan saat proses login.

Sistem tidak menyimpan password dalam bentuk plaintext atau hash mentah, melainkan menyimpan satu string Argon2 dalam format PHC yang berisi seluruh metadata yang dibutuhkan untuk verifikasi.



Gambar 4.1 Arsitektur Implementasi Password Hashing

C. *Konfigurasi Parameter Argon2id*

Parameter Argon2id dikonfigurasi berdasarkan rekomendasi RFC 9106 serta hasil analisis pada Bab III. Konfigurasi yang digunakan pada implementasi ini adalah sebagai berikut:

- Memory cost: 102400 KB (100 MB)
- Time cost: 3
- Parallelism: 4
- Hash length: 32 byte

Pemilihan memory cost sebesar 100 MB bertujuan untuk secara signifikan membatasi kemampuan GPU dalam melakukan brute-force paralel, sementara time cost dan parallelism disesuaikan agar performa tetap dapat diterima oleh sistem backend.

D. *Implementasi Fungsi Hashing Password*

Fungsi hashing password bertugas untuk mengubah password plaintext menjadi hash Argon2id yang aman. Pada tahap ini, sistem secara otomatis:

1. Menghasilkan salt acak.
2. Menjalankan algoritma Argon2id dengan parameter yang telah ditentukan.
3. Menghasilkan output dalam format PHC string.

Berikut adalah implementasi fungsi hashing password:

makalah_2 - argon2_lab.py

```python
@dataclass(frozen=True)
class Argon2Params:
    time_cost: int
    memory_cost_kib: int
    parallelism: int
    hash_len: int = 32

    @property
    def memory_cost_mb(self) -> float:
        return self.memory_cost_kib / 1024.0

class Argon2Service:
    def __init__(self, params: Argon2Params) -> None:
        self._params = params
        self._hasher = PasswordHasher(
            time_cost=params.time_cost,
            memory_cost=params.memory_cost_kib,
            parallelism=params.parallelism,
            hash_len=params.hash_len,
        )
    def hash_password(self, password: str) -> str:
        self._validate_password(password)
        return self._hasher.hash(password)

    @staticmethod
    def _validate_password(password: str) -> None:
        if not isinstance(password, str):
            raise TypeError("password must be a string")
        if len(password) == 0:
            raise ValueError("password must not be empty")
        if len(password) > 256:
            raise ValueError("password too long (max 256 chars)")
```

Gambar 4.2 Implementasi Password Hashing

Fungsi ini hanya menerima password dalam bentuk string dan mengembalikan satu string Argon2id lengkap yang siap disimpan di database. Tidak ada informasi sensitif lain yang perlu disimpan secara terpisah.

E. *Implementasi Fungsi Verifikasi Password*

Fungsi verifikasi password digunakan saat proses login. Sistem tidak membandingkan password secara langsung, melainkan menjalankan kembali Argon2id dengan parameter dan salt yang sama seperti yang tersimpan di dalam PHC string.

Implementasi verifikasi password ditunjukkan sebagai berikut:

makalah_2 - argon2_lab.py

```python
def verify_password(self, stored_hash: str, password: str) -> bool:
        self._validate_password(password)
        try:
            return self._hasher.verify(stored_hash, password)
        except VerifyMismatchError:
            return False
        except VerificationError:
            return False
```

Gambar 4.3 Arsitektur Implementasi Password Hashing

Pendekatan ini memiliki beberapa keunggulan:

- Tidak memerlukan parsing manual parameter Argon2.
- Aman terhadap timing attack karena perbandingan dilakukan secara internal.
- Mendukung migrasi parameter secara otomatis apabila diperlukan.

F. *Format Penyimpanan Hash Password*

Argon2 menggunakan format standar PHC string, yang menyimpan seluruh informasi yang diperlukan untuk verifikasi password. Contoh format penyimpanan hash adalah sebagai berikut:

Gambar 4.4 Arsitektur Implementasi Password Hashing

Format ini mencakup:

- Varian algoritma (argon2id)
- Versi Argon2
- Parameter memory, time, dan parallelism
- Salt acak
- Hash hasil

Keuntungan penggunaan PHC string adalah:

1. Tidak memerlukan skema database tambahan untuk menyimpan parameter.
2. Mendukung backward compatibility dan migrasi parameter.
3. Mengurangi risiko kesalahan konfigurasi.

.

G. *Simulasi Brute-Force Password*

Untuk menunjukkan ketahanan Argon2id terhadap brute-force, dilakukan simulasi brute-force sederhana menggunakan CPU. Simulasi ini mencoba seluruh kombinasi password huruf kecil dengan panjang tertentu.

Kode simulasi brute-force ditunjukkan sebagai berikut:

Gambar 4.5 Implementasi Brute Force

Simulasi ini memperlihatkan bahwa bahkan untuk password pendek, Argon2id membutuhkan waktu yang sangat lama untuk brute-force jika dibandingkan dengan algoritma hashing tradisional.

H. *Pertimbangan Keamanan Tambahan*

Selain hashing password, implementasi sistem juga mempertimbangkan aspek keamanan berikut:

1. Pembatasan panjang password input
   Untuk mencegah serangan DoS melalui password berukuran ekstrem.
2. Penggunaan salt acak otomatis
   Mencegah rainbow table dan korelasi antar akun.
3. Kesiapan migrasi parameter
   Parameter Argon2 dapat ditingkatkan di masa depan tanpa memaksa reset password pengguna.
4. Integrasi dengan mekanisme keamanan lain
   Seperti rate-limiting, account lockout, dan autentikasi multi-faktor.

V. PENGUJIAN

A. *Setup Eksperimen*

Pengujian dilakukan pada lingkungan komputasi berbasis CPU dengan spesifikasi umum yang merepresentasikan server aplikasi modern. Pengujian tidak menggunakan GPU secara langsung, namun analisis resistansi terhadap GPU dilakukan berdasarkan hasil eksperimen CPU dan referensi empiris dari penelitian terdahulu.

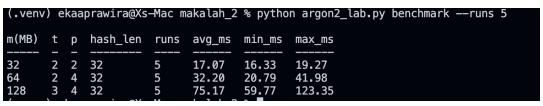
Konfigurasi lingkungan pengujian adalah sebagai berikut:

- Sistem Operasi: MacOS
- Bahasa pemrograman: Python 3.10
- Library: argon2-cffi
- Mode pengujian: single-user hashing dan simulasi brute-force CPU

Pengujian dilakukan dengan beberapa kombinasi parameter Argon2id untuk melihat pengaruh memory cost, time cost, dan parallelism terhadap waktu hashing.

B. *Hasil Pengujian Waktu Hashing*

Pengujian waktu hashing dilakukan dengan mengukur waktu yang dibutuhkan Argon2id untuk menghasilkan satu hash password dengan berbagai parameter. Setiap konfigurasi diuji beberapa kali dan diambil nilai rata-rata.



Gambar 5.1 Hasil Pengujian Waktu Hashing Argon2id

Berdasarkan hasil di atas, dapat diamati bahwa memory cost merupakan parameter yang paling berpengaruh terhadap waktu hashing. Peningkatan memory cost dari 32 MB ke 128 MB menyebabkan waktu hashing meningkat hampir empat kali lipat. Time cost juga berpengaruh, namun dampaknya lebih kecil dibandingkan memory cost.

Dari sisi keamanan, peningkatan waktu hashing ini justru diinginkan, karena secara langsung meningkatkan biaya komputasi bagi penyerang yang mencoba melakukan brute-force.

C. *Simulasi Brute-Force Berbasis CPU*

Untuk mengevaluasi ketahanan Argon2id terhadap brute-force, dilakukan simulasi brute-force sederhana menggunakan CPU. Simulasi mencoba seluruh kombinasi password huruf kecil dengan panjang hingga empat karakter.

Hasil simulasi menunjukkan bahwa:

- Pada algoritma hashing cepat seperti SHA-256 atau PBKDF2 dengan iterasi rendah, brute-force password pendek dapat diselesaikan dalam waktu sangat singkat.
- Pada Argon2id dengan memory cost ≥64 MB, brute-force tidak dapat diselesaikan dalam waktu praktis, bahkan untuk password yang relatif pendek.

Pada konfigurasi Argon2id dengan memory cost 128 MB dan time cost 3, simulasi brute-force selama beberapa menit tidak berhasil menemukan password target. Hal ini menunjukkan bahwa Argon2id secara efektif memperlambat setiap percobaan hashing, sehingga brute-force menjadi tidak praktis.

D. *Analisis Resistansi terhadap GPU Acceleration*

Meskipun pengujian dilakukan pada CPU, analisis terhadap GPU acceleration dapat dilakukan berdasarkan karakteristik arsitektur Argon2id. GPU memiliki ribuan core paralel, namun memiliki keterbatasan memori per core dan bandwidth memori yang harus dibagi di antara seluruh thread.

Argon2id memaksa penggunaan memori besar (≥64 MB) untuk setiap proses hashing. Akibatnya:

- GPU tidak dapat menjalankan ribuan thread hashing secara paralel.
- Setiap thread membutuhkan alokasi memori besar, sehingga jumlah thread aktif sangat terbatas.
- Keunggulan paralelisme GPU menjadi hilang.

Penelitian terdahulu menunjukkan bahwa:

- GPU mampu mempercepat PBKDF2 hingga ratusan hingga ribuan kali dibandingkan CPU.
- Untuk Argon2id, percepatan GPU hanya berkisar 2–5 kali, bahkan dalam beberapa konfigurasi hampir setara dengan CPU.

Dengan demikian, Argon2id secara efektif menghilangkan keunggulan utama GPU dalam brute-force password.

E. *Interpretasi Hasil Pengujian*

Berdasarkan hasil pengujian dan analisis, dapat disimpulkan beberapa poin penting:

1. Peningkatan memory cost secara signifikan meningkatkan keamanan password hashing.
2. Time cost berfungsi sebagai penguat tambahan, tetapi tidak seefektif memory cost dalam menahan GPU.
3. Argon2id mampu memperlambat brute-force CPU secara drastis.
4. Desain memory-hard Argon2id menjadikannya sangat tahan terhadap GPU acceleration.
5. Parameter Argon2id dapat disesuaikan dengan kemampuan hardware server tanpa mengubah mekanisme autentikasi pengguna.

Hasil ini menunjukkan bahwa Argon2id sangat cocok digunakan sebagai standar password hashing untuk aplikasi modern yang menghadapi ancaman brute-force skala besar.

VI. KESIMPULAN

Makalah ini membahas implementasi Argon2id sebagai mekanisme password hashing yang dirancang untuk tahan terhadap serangan brute-force dan akselerasi GPU. Berdasarkan analisis teoretis, implementasi sistem, serta hasil pengujian, dapat disimpulkan bahwa Argon2id merupakan solusi password hashing yang jauh lebih aman dibandingkan algoritma hashing tradisional seperti PBKDF2 dan bcrypt.

Keunggulan utama Argon2id terletak pada sifatnya sebagai memory-hard function, yang secara efektif membatasi kemampuan GPU untuk melakukan hashing paralel dalam jumlah besar. Pengujian menunjukkan bahwa peningkatan memory cost memberikan dampak paling signifikan dalam memperlambat proses hashing, sehingga brute-force menjadi tidak praktis bahkan ketika penyerang menggunakan perangkat keras modern.

Implementasi Argon2id dengan parameter yang sesuai rekomendasi RFC 9106, yaitu memory cost minimal 64 MB dan time cost minimal 2, mampu memberikan keseimbangan yang baik antara keamanan dan performa sistem. Selain itu, format penyimpanan PHC string memudahkan migrasi parameter keamanan di masa depan tanpa memengaruhi pengguna.

Dengan demikian, Argon2id direkomendasikan sebagai mekanisme password hashing standar untuk aplikasi modern yang menyimpan data sensitif. Untuk penelitian lanjutan, pengujian dapat diperluas dengan melakukan eksperimen langsung menggunakan GPU, serta mengintegrasikan Argon2id dengan mekanisme keamanan tambahan seperti rate-limiting, multi-factor authentication, dan sistem deteksi anomali login.

VII. Ucapan Terima Kasih

Terima kasih juga penulis sampaikan kepada Bapak Dr. Ir. Rinaldi, M.T. sebagai dosen pengampu dalam mata kuliah IF4020 Kriptografi atas ilmu yang telah diberikan kepada penulis sehingga penulis dapat menyelesaikan makalah ini. Tidak lupa terima kasih juga penulis sampaikan kepada diri sendiri yang senantiasa memberikan dukungan dan motivasi kepada penulis.

References

[1] Biryukov, A., D. D. Khovratovich, and L. Perrin, "Argon2: The Memory-Hard Function for Password Hashing and Other Applications," 2016.

[2] Internet Engineering Task Force (IETF), "RFC 9106: Argon2 Memory-Hard Function for Password Hashing and Proof-of-Work Applications," 2021.

[3] Open Web Application Security Project (OWASP), "OWASP Password Storage Cheat Sheet," 2023.

[4] Percival, C., "Stronger Key Derivation via Sequential Memory-Hard Functions," 2009.

Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 24 Desember 2025

Mohammad Nugraha Eka Prawira
13522001