Implementasi Penambahan Salt Dalam Algoritma SHA-256 Pada AirDrop

Alexander Delvin Widjaja - 18220064
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail : delvin.alexander67@gmail.com

***Abstract*—Pertukaran data melalui AirDrop sudah menjadi bagian dari kehidupan sehari-hari banyak orang. Penggunaan AirDrop sangatlah membantu proses pertukaran dan pengiriman data, karena pertukaran data melalui AirDrop sangat praktis dan bisa dilakukan secara *offline* tanpa perlu media pengantara antar gawai yang bertukar atau mengirimkan data. Namun kenyamanan ini tentunya bukan tanpa isu. AirDrop memiliki isu keamanan pada sisi *hash* informasi *email* dan nomor telepon pemakainya. Isu ini muncul sebab hasil hash dapat dengan mudah di *brute force* untuk menampilkan *preimage*-nya. Pada makalah ini akan dibahas konsep penambahan *salt* pada SHA-256 yang dapat membantu memperlambat dan mempersulit proses *brute force hash* yang bertujuan untuk pengambilan data *emai* dan nomor telepon**

Keywords*—AirDrop; SHA-256; *Salt; Hash, Brute Force;

I. PENDAHULUAN

Di zaman yang serba maju ini, bertukar data melalui gawai merupakan hal yang sangat lazim dilakukan. Media seperti bluetooth, Wi-Fi, BLE, drive, quickshare, dan AirDrop sudah menjadi bagian yang tidak lagi dapat dipisahkan dari kegiatan dan kehidupan sehari-hari. Munculnya media ini memang sangat memudahkan proses pertukaran data. Proses yang dulunya membutuhkan waktu yang cukup lama dan memerlukan alat maupun konfigurasi tertentu menjadi sangat cepat dan mudah untuk dilakukan, bahkan kebanyakan dari media ini memungkinkan pertukaran tanpa perlu berpindah gawai. Namun mudahnya pertukaran data tentunya datang dengan permasalahnnya sendiri. Contohnya adalah permasalahan keamanan pada AirDrop.

AirDrop merukapan layanan berbagi *file* yang disediakan oleh Apple Inc. Layanan ini dapat digunakan oleh gawai dengan dengan OS diatas iOS 7 atau gawai dengan OS diatas OS X 10.11. AirDrop menggunakan BLE (Bluetooth Low Energy) untuk membentuk jalur Wi-Fi *peer-to-peer* yang digunakan dalam proses mengirim *file*. Layanan AirDrop dapat bekerja secara *ofline* karena jalur atau jaringan yang dibentuk merupakan jaringan *direct*/langsung [1].

Permasalahan keamanan pada AirDrop muncul ketika penyerang dapat menggunakan kelemahan dalam AirDrop untuk mendapatkan data nomor telepon dan email dari pengguna AirDrop lain yang sedang berada dalam jangkauan AirDrop penyerang. Isu ini disebabkan dari penggunaan fungsi *hash* sebagai upaya melindungi atau menyembunyikan data nomor telepon dan data *email*. Sayangnya *hash* yang digunakan adalah SHA-256 tanpa proses *"salting"* sehingga memungkinkan penyerang untuk mendapat data dari nilai *hash* (preimage) dengan cara melakukan *brute force* (mencocokan data hasil *hash* dengan tabel berisi semua hasil *hash* dan *input*-nya)[2].

Pada makalah ini akan dilakukan experimen sederhana untuk melakukan implementasi penambahan *Salt* pada algoritma SHA-256 dengan *input* berupa email dan password serta *Salt* yang digenerasi secara acak sepanjang 32 byte.

II. DASAR TEORI

A. *Hash*

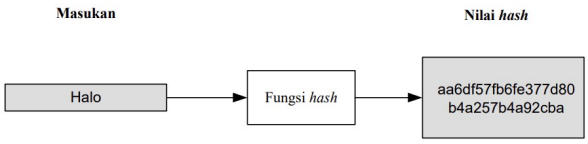
Hash merupakan sebuah fungsi yang memberikan output dengan panjang yang tetap apapun inputnya. Baik input maupun output dari fungsi hash adalah string. Input dari hash disebut juga *message* sementara itu output dari hash disebut *message digest / hash-value* . Fungsi ini merupakan fungsi satu arah sehingga kita tidak dapat mengembalikan nilai hasil hash (hash-value) menjadi pesan semula (*message*) [3].

Fungsi *Hash* :

h = H(m)

h adalah nilai *hash* (*hash-value*), H merupakan fungsi *hash* yang digunakan, dan m merupakan (*message)* pesan yang dimasukan kesdalam fungsi *hash*.

Cara kerja fungsi *hash*.



Gambar 1. Skema Fungsi Hash. (Sumber : Bahan kuliah II4031 Kriptografi dan Koding [3])

B. *SHA-256*

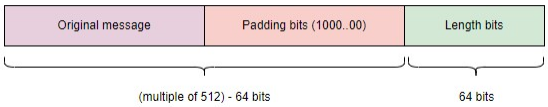
SHA-256 (*Secure Hash Algorithm* 256) merupakan salah satu anggota dari keluarga SHA-2. SHA-2 sendiri merupakan

suksesor dari SHA 1. Angka 256 digunakan karena, tidak perduli berapapun ukuran *plainteks*-nya hasil *hash* dari SHA-256 (*hash value / message digest*) berukuran 256 bit [4,5].

Proses SHA-256

- Mengubah *message* menjadi biner (nilai 0 atau 1)
- Menambahkan *padding* (satu buah 1 diikuti 0) hingga jumlah bit pesan merupakan kelipatan 512 dikurangi 64

  Jumlah bit = ((n x 512) - 64)
- Tambahkan 64 bit sisanya. 64 bit ini mewakili panjang bit dari input



Gambar 2. *Input* SHA. (Sumber : https://www.educative.io/answers/what-are-the-different-steps-in-sha-256)

- Inisialisasi *hash value* (8 variabel nilainya konstan) 8 nilai konstan ini didapat dari mengambil nilai pecahan dari akar pangkat dua 8 bilangan prima pertama yang diubah kedalam nilai biner hingga didapat nilai biner sepanjang 32 bit. Terakhir nilai biner diubah menjadi nilai hex [8].

  Menghitung nilai A:

  Bilangan prima pertama adalah dua, maka cari terlebih dahulu nilai dari akar dua

  $$2 \wedge (0.5) = 1.41421356237\ldots$$

  Kemudian ambil nilai pecahan (nilai dibelakang koma) dan jadikan biner ambil hingga 32 bit pertama

  $$0.41421356237\ldots$$
  $$=$$
  $$0.01101010000010011110011001100111$$

  Maka nilai biner adalah

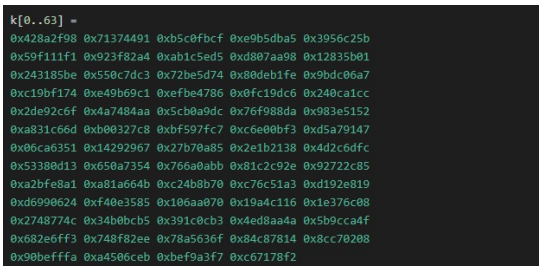
  0110 1010 0000 1001 1110 0110 0110 0111

  Ubah nilai biner ke nilai hex

Gambar 3. Pengubahan nilai biner menjadi nilai hex (Sumber : https://www.youtube.com/watch?v=orIgy2MjqrA&ab_channel=RedBlockBlue)

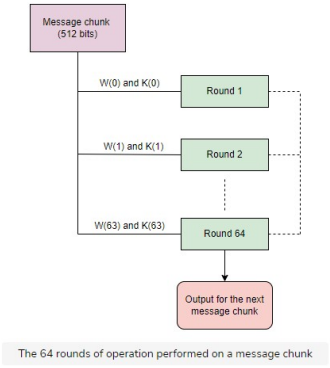
Gambar 4. Inisialisasi hash-value. (Sumber : https://www.educative.io/answers/what-are-the-different-steps-in-sha-256)

- Inisialisasi *round constants* (K (*Array* dengan isi 64 variabel)). Nilai ini didapat dari mengambil nilai pecahan dari akar pangkat tiga 64 bilangan prima pertama yang diubah kedalam nilai biner hingga didapat 32 bit dan terakhir diubah ke nilai hex



Gambar 5. Inisialisasi round constans (Sumber : https://www.educative.io/answers/what-are-the-different-steps-in-sha-256)

- Lakukan pemisahan untuk setiap 512 bit (1 *chunk*) utnuk setiap *chunk* lakukan putaran 64 kali kemudian hitung nilai W untuk tiap putaran dan hitung nilai Ch, Ma, A, E, serta perbaharui nilai a,b,c,d,e,f,g,h (Gambar 8)

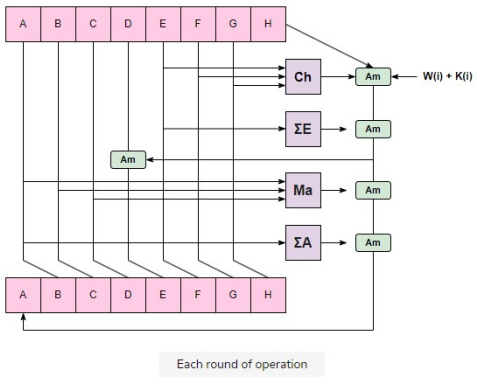


Gambar 6. Skema Fungsi *Hash* SHA-256 per *chunk*. (Sumber : https://www.educative.io/answers/what-are-the-different-steps-in-sha-256)

Untuk mendapatkan 16 nilai W pertama diambil dari 512 bit yang dibagi menjadi 16 bagian sehingga masing masing W berukuran 32 bit. Untuk nilai W mulai selanjutnya sampai nilai W ke-64 dihitung dengan rumus (Gambar 7)

$$s^0 = (W[i-15] \text{ right rotate } 7) \text{ XOR } (W[i-15] \text{ right rotate } 18) \text{ XOR } (W[i-15] \text{ right shift } 3)$$
$$s^1 = (W[i-2] \text{ right rotate } 17) \text{ XOR } (W[i-2] \text{ right rotate } 19) \text{ XOR } (W[i-2] \text{ right shift } 10)$$
$$W(i) = W[i-16] + s^0 + W[i-7] + s^1$$

Gambar 7. Rumus Menghitung nilai W (Sumber : https://www.educative.io/answers/what-are-the-different-steps-in-sha-256)



Each round carries out the following set of computations:

- $Ch = (E \text{ AND } F) \text{XOR } ((\text{NOT } E) \text{ AND } G)$
- $Ma = (A \text{ AND } B) \text{XOR } (A \text{AND } C) \text{ XOR } (B \text{ AND } C)$
- $\sum A = (A >>> 2) \text{XOR } (A >>> 13) \text{XOR } (A >>> 22)$
- $\sum E = (E >>> 6) \text{XOR } (E >>> 11) \text{XOR } (E >>> 25)$
- $Am = (\text{addition}) \bmod (2^{32})$

Gambar 8. Skema perputaran fungsi *hash* SHA-256 (Sumber : https://www.educative.io/answers/what-are-the-different-steps-in-sha-256)

C. *Brute Force*

Brute force attack merupakan metode *hacking* dengan cara melakukan melakukan *trial and error* untuk memecahkan *password* atau informasi lainnya.

Ada 5 jenis *Brute Force Attack*:

- *Simple Brute Force Attack*

  Merupakan *Brute Force* yang terjadi ketika *hacker* mencoba memasukan *password* dan *username* yang sering digunakan secara manual. (menebak dari *username* dan *password* yang lazim digunakan)

- *Dictionary Attack*

  Merupakan *Brute force* dengan cara memilih satu target kemudian mencoba semua *password* yang memungkinkan terhadap *username* dari target

- *Hybrid Brute Force Attack*

  Merupakan gabungan dari *simple bruteforce attack* dan *dictionary attack*. *Brute force attack* ini dimulai dengan mengetahui *Username* kemudian mencoba *password* dengan cara menggabungkan kata kata yang biasa digunakan seperti tahun , kata kata populer, dan angka yang sering digunakan sehingga akan didapat pasangan *username* dan *password*

- *Reverse Brute Force Attack*

  Merupakan metode *brute force attack* yang terlebih dahulu mengetahu *password* kemudian melakukan pencocokan terhadap *username*

- *Credential Stuffing*

  Menggunakan kombinasi password dan *username* yang telah berhasil dimiliki oleh *hacker* kemudain hacker mencoba kombinasi pada *website* sehingga jika pemilik memiliki *password* yang sama antar akun pada *website* berbeda maka akun pada i lainnya juga ikut terbuka [6].

D. *Salt*

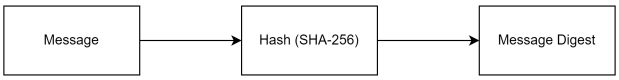
Proses menambahkan nilai yang unik pada *message* sebelum proses *hashing* dilakukan. *Salt* ditambahakn demi menambah keamanan dari *password*. dengan adanya *salt* ketika ada *password* yang sama hasil *hash* tidak akan sama karena variabel *salt* berbeda. Dengan menambahkan *salt* seolah-oleh menyembunyikan *hash value.* Penambahan/ penggunaan *salt* dapat mengurangi resiko serangan secara *brute force* dan *rainbow table*. Proses penambahan *salt* dapat dilakukan si bagian depan maupun dibagian belakang [7].

Gambar 9. Penggunaann Salt (Sumber : https://www.educative.io/answers/what-are-the-different-steps-in-sha-256)

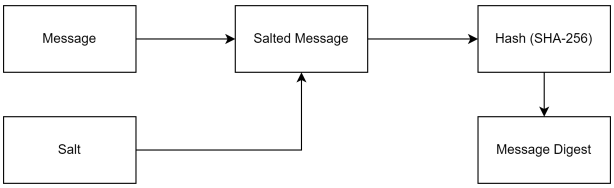
III. RANCANGAN

Rancangan yang akan dibuat terbilang sederhana karena hanya menambahkan proses salting pada rancangan yang telah ada (Gambar 10). Untuk proses generasi *salt* sendiri akan dilakukan dengan mengambil nilai acak berupa sebesar 32 byte. Setelah generasi *salt* selesai maka dilakukan penambahan nilai *salt* ke *message* sebelum dilakukan proses *hash*. Kemudian proses *hash* menggunakan SHA-256 akan dilakukan seperti pada skema awal. Pada perancangan kali ini hanya berfokus pada proses *hashing* dan generasi *salt* saja. Berikut merupakan bagan proses *hashing* awal (Gambar 10), bagan proses *hashing* dengan menambahkan *salt* (Gambar 11), dan

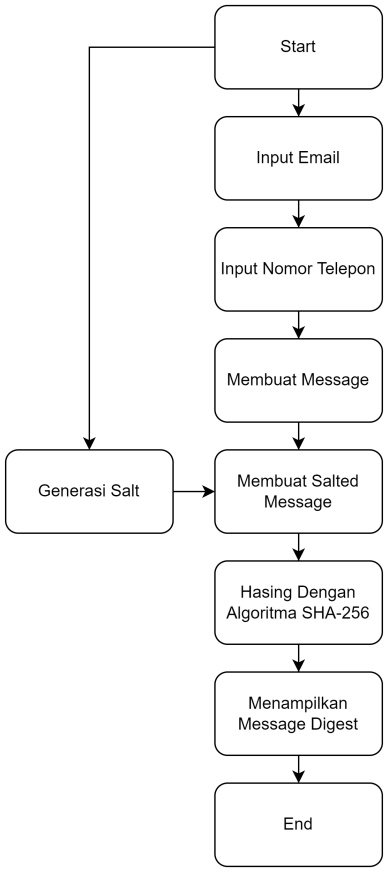
diagram alir dari proses pada program yang akan dibuat (Gamabr 12)



Gambar 10. Proses *Hashing* Awal



Gambar 11. Rancangan dengan penambahan *salt*



Gambar 12. Diagram alir program

IV. IMPLEMENTASI

Untuk melihat perbandingan hasil dari proses *salted* dan prosees biasa maka akan dibuat dua program. Program pertama diimplementasi sesuai dengan diagram alir namun tidak memiliki fungsi generasi *salt* dan pembuatan *salted message*. Sementara program kedua dibuat sesuai dengan diagram alir pada rancangan dengan menerapkan generasi *salt* dan pembuatan *salted message*. Pembuatan program dilakukan dengan menggunakan bahasa Python. Berikut merupakan kode program yang dibuat

A. Kode program tanpa *salt*

```
import hashlib

# menerima input email
email = str(input("Masukkan Email : "))

# menerima inpit nomor telepon
telp = str(input("Masukkan Nomor Telepon : "))

#membuat message dengan cara mengabungkan email dan telepon
message = email+telp
print("Message :",message)


#proses hasing
digest = hashlib.sha256(message.encode())
digest_hex = digest.hexdigest()
print("Hasil Hash :",digest_hex)
```

B. Kode program dengan *salt*

```
    import hashlib
    import random
    import string

    # menerima input email
    email = str(input("Masukkan Email : "))

    # menerima inpit nomor telepon
    telp = str(input("Masukkan Nomor Telepon : "))

    #membuat message dengan cara mengabungkan email dan telepon
    message = email+telp
    print("Message :",message)

    #fungsi generasi salt
    def generate_salt(length):
        #Pilihan berupa semua huruf dan angka
        letters = string.ascii_letters + string.digits
        result_str = ''.join(random.choice(letters) for i in range(length))
        return result_str
```

```
    #generasi salt
    # string diasumsikan memiliki ukuran 4 byte sehingga digenerasi salt dengan panjang 8 string sehingga terbentuk salt berukuran 32 byte
    salt = generate_salt(8)
    print("Salt :",salt)

    #pembuiatan salted message
    smessage = message + salt
    print("Salted Message :",smessage)

    #proses hasing
    digest = hashlib.sha256(smessage.encode())
    digest_hex = digest.hexdigest()
    print("Hasil Hash :",digest_hex)
```

```
pData/Local/Programs/Python/Python39/python.exe" "d:/Delvin/ITB/Semester 6/Kripto/Salted SHA-256/sHA256.py"
Masukkan Email : delvin.alexander67@gmail.com
Masukkan Nomor Telepon : 012345678910
Message : delvin.alexander67@gmail.com012345678910
Hasil Hash : 7d8aa6cf28a931d2e311776d6f199c8f3a43cb8cf1f0f1edbc84801a6178b435
PS D:\Delvin\ITB\Semester 6\Kripto\Salted SHA-256>
PS D:\Delvin\ITB\Semester 6\Kripto\Salted SHA-256> & "C:/Users/hon gaming PC/AppData/Local/Programs/Python/Python39/python.exe" "d:/Delvin/ITB/Semester 6/Kripto/Salted SHA-256/sHA256.py"
Masukkan Email : delvin.alexander67@gmail.com
Masukkan Nomor Telepon : 012345678910
Message : delvin.alexander67@gmail.com012345678910
Hasil Hash : 7d8aa6cf28a931d2e311776d6f199c8f3a43cb8cf1f0f1edbc84801a6178b435
PS D:\Delvin\ITB\Semester 6\Kripto\Salted SHA-256>
PS D:\Delvin\ITB\Semester 6\Kripto\Salted SHA-256> & "C:/Users/hon gaming PC/AppData/Local/Programs/Python/Python39/python.exe" "d:/Delvin/ITB/Semester 6/Kripto/Salted SHA-256/sHA256.py"
Masukkan Email : delvin.alexander67@gmail.com
Masukkan Nomor Telepon : 012345678910
Message : delvin.alexander67@gmail.com012345678910
Hasil Hash : 7d8aa6cf28a931d2e311776d6f199c8f3a43cb8cf1f0f1edbc84801a6178b435
```

Gambar 13. Pengujian program tanpa *salt*

```
PS D:\Delvin\ITB\Semester 6\Kripto\Salted SHA-256> & "C:/Users/hon gaming PC/AppData/Local/Programs/Python/Python39/python.exe" "d:/Delvin/ITB/Semester 6/Kripto/Salted SHA-256/Salted SHA256.py"
Masukkan Email : delvin.alexander67@gmail.com
Masukkan Nomor Telepon : 012345678910
Message : delvin.alexander67@gmail.com012345678910
Salt : 48fG8UH7
Salted Message : delvin.alexander67@gmail.com012345678910148fG8UH7
Hasil Hash : 64ab2057df079b786f04ce1baf36d3f7293916478453cc3da90f8f2692b48e22
PS D:\Delvin\ITB\Semester 6\Kripto\Salted SHA-256>
PS D:\Delvin\ITB\Semester 6\Kripto\Salted SHA-256> & "C:/Users/hon gaming PC/AppData/Local/Programs/Python/Python39/python.exe" "d:/Delvin/ITB/Semester 6/Kripto/Salted SHA-256/Salted SHA256.py"
Masukkan Email : delvin.alexander67@gmail.com
Masukkan Nomor Telepon : 012345678910
Message : delvin.alexander67@gmail.com012345678910
Salt : aKXJE63g
Salted Message : delvin.alexander67@gmail.com012345678910aKXJE63g
Hasil Hash : f247e7098150f84e1fa22101cad2ee7778606a1482da32b2fab7a80d29dbaf74
PS D:\Delvin\ITB\Semester 6\Kripto\Salted SHA-256>
PS D:\Delvin\ITB\Semester 6\Kripto\Salted SHA-256> & "C:/Users/hon gaming PC/AppData/Local/Programs/Python/Python39/python.exe" "d:/Delvin/ITB/Semester 6/Kripto/Salted SHA-256/Salted SHA256.py"
Masukkan Email : delvin.alexander67@gmail.com
Masukkan Nomor Telepon : 012345678910
Message : delvin.alexander67@gmail.com012345678910
Salt : hmyrjTfy
Salted Message : delvin.alexander67@gmail.com012345678910hmyrjTfy
Hasil Hash : e2c8342df89ae687a599236f0a150d98055a42b761096fd3b806f60d973bde0c
```

Gambar 14. Pengujian program dengan *salt*

V. Pengujian dan Hasil

Pengujian program dilakukan dengan cara menjalan program melalui terminal sebanyak tiga kali untuk masing-masing program. Setiap kali program berjalan dilakukan *input* email dan *input* nomor telepon yang sama untuk setiap program. Kemudian program akan bekerja dan memberikan hasil beruma *hash value / message digest*.

A. Pengujian program tanpa *salt*

Untuk pengujian digunakan *input*:
email : delvin.alexander67@gmail.com dan,
nomor telepon : 012345678910

B. Pengujian program dengan *salt*

Untuk pengujian digunakan *input*:
email : delvin.alexander67@gmail.com dan,
nomor telepon : 012345678910

C. Hasil pengujian

Dari pengujian didapat hasil *hash* yang sama untuk setiap pengujian program tanpa *salt* yaitu :

- 7d8aa6cf28a931d2e311776d6f199c8f3a43cb8cf1f0f1edbc84801a6178b435.

Sementara untuk program dengn *salt* menghasilkan hasil yang berbeda untuk setiap pengujian yaitu :

- 64ab2057df079b786f04ce1baf36d3f7293916478453cc3da90f8f2692b48e22
- f247e7098150f84e1fa22101cad2ee7778606a1482da32b2fab7a80d29dbaf74
- e2c8342df89ae687a599236f0a150d98055a42b761096fd3b806f60d973bde0c

Hasil pengujian ini membuktikan penggunaan *salt* dapat memperlambat proses *preimage* dengan menggunakan metode *brute force.* Hal ini dikarenakan adanya penambahan *salt* sehingga dibutuhnya lebih banyak komputasi untuk mendapatkan nilai *preimage*. Selain itu Penambahan *salt* juga mencegah serangan menggunakan *rainbow table.*

VI. Kesimpulan

Pada makalah ini dibahas mengenai penambahan *salt* sebagai cara untuk menangulangi isu kebocoran data pada proses AirDrop. Kebocoran data ini disebabkan mudahnya dilakukan *brute force* pada hasil hash data *email* dan nomor telepon. Dari hasil experimen sederhana dapat dilihat penambahan *salt* pada algoritma SHA-256 dapat menambah variasi dan mempersulit proses *brute force* hasil *hash*. Namun untuk penerapannya secara nyata masih harus dikembangkan lagi karena program yang diciptakan masih berupa penggambaran konsep dan belum siap pakai.

REFERENCES

[1] "AirDrop security," Apple Support, 2021. [Online]. Available: https://support.apple.com/en-lk/guide/security/sec2261183f4/web. [Accessed: May 17, 2023]

[2] "SiliconANGLE," SiliconANGLE, Apr. 26, 2021. [Online]. Available: https://siliconangle.com/2021/04/25/security-issues-apple-airdrop-allow-attackers-steal-personal-information/. [Accessed: May 19, 2023]

[3] R. Munir, "Bahan kuliah II4031 Kriptografi dan Koding," 2023 [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi-dan-Koding/2022-2023/13-Fungsi-hash-2023.pdf. [Accessed: May 21, 2023]

[4] M. Thakkar, "SHA 256 Algorithm Explained by a Cyber Security Consultant - InfoSec Insights," InfoSec Insights, Apr. 22, 2022. [Online]. Available: https://sectigostore.com/blog/sha-256-algorithm-explained-by-a-cyber-security-consultant/. [Accessed: May 21, 2023]

[5] Baivab Kumar Jena, "A Definitive Guide to Learn The SHA-256 (Secure Hash Algorithms)," Simplilearn.com, Jul. 13, 2021. [Online]. Available: https://www.simplilearn.com/tutorials/cyber-security-tutorial/sha-256-algorithm. [Accessed: May 21, 2023]

[6] "What is a Brute Force Attack? | Definition, Types & How It Works," Fortinet, 2017. [Online]. Available: https://www.fortinet.com/resources/cyberglossary/brute-force-attack#:~:text=A%20brute%20force%20attack%20is,and%20organizations'%20systems%20and%20networks.. [Accessed: May 21, 2023]

[7] P. Nohe, "The difference between Encryption, Hashing and Salting," Hashed Out by The SSL StoreTM, Dec. 19, 2018. [Online]. Available: https://www.thesslstore.com/blog/difference-encryption-hashing-salting/. [Accessed: May 21, 2023]

[8] RedBlockBlue, "SHA-256 | COMPLETE Step-By-Step Explanation (W/ Example)," YouTube. Mar. 17, 2022 [Online]. Available: https://www.youtube.com/watch?v=orIgy2MjqrA&ab_channel=RedBlockBlue. [Accessed: May 22, 2023]

[9] "What are the different steps in SHA-256?," Educative: Interactive Courses for Software Developers, 2015. [Online]. Available: https://www.educative.io/answers/what-are-the-different-steps-in-sha-256. [Accessed: May 22, 2023]

[10] "SHA in Python," GeeksforGeeks, Feb. 14, 2018. [Online]. Available: https://www.geeksforgeeks.org/sha-in-python/. [Accessed: May 22, 2023]

[11] "Python Generate random string of given length," GeeksforGeeks, May 21, 2019. [Online]. Available: https://www.geeksforgeeks.org/python-generate-random-string-of-given-length/. [Accessed: May 22, 2023]

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, 22 Mei 2023

Alexander Delvin Widjaja
Nim : 18220064