# Langkah Praktikum

## Hashing

1. Pastikan **md5sum** telah terinstall di perangkat yang digunakan.

|  |
| --- |
| Penjelasan :  Output dari perintah which md5sum adalah /usr/bin/md5sum, berarti md5sum sudah terinstal dan siap digunakan.  Fungsi utama dari md5sum adalah untuk menghasilkan hash nilai MD5 dari sebuah berkas atau teks. Hash nilai MD5 digunakan untuk memverifikasi integritas berkas, karena perubahan sedikit pun pada berkas akan menghasilkan hash nilai yang berbeda. |

1. Download file pada tautan berikut (kamu bisa menggunakan command wget) dan ekstrak.
   1. message1.bin & message2.bin

|  |
| --- |
| wget <https://github.com/isfahany/infosec-module-downloadable-file/raw/master/collision-example/file.zip> |

|  |
| --- |
| Penjelasan:  wget: perintah untuk mengunduh file dari web  Perintah unzip akan digunakan untuk mengekstrak. File didalamnya berupa message1.bin dan message2.bin |

1. Jalankan command berikut pada file terkait.
   1. sha1sum

|  |
| --- |
| sha1sum message1.bin message2.bin |

|  |
| --- |
| Penjelasan : Output sha1sum berupa SHA-1 dari setiap berkas. SHA-1 adalah nilai hash yang dihasilkan dari algoritma fungsi hash SHA-1 (Secure Hash Algorithm 1).  Setiap berkas memiliki hash nilai SHA-1 yang berbeda, yang menunjukkan data di dalam berkas tersebut berbeda. Dapat dibandingkan apakah kedua berkas memiliki konten (isi) yang sama atau berbeda.   * Hash nilai SHA-1 dari message1.bin adalah : c6b384c4968b28812b676b49d40c09f8af4ed4cc. * Hash nilai SHA-1 dari message2.bin adalah : c728d8d93091e9c7b87b43d9e33829379231d7ca.   Kedua hash nilai nya berbeda, artinya kedua berkas memiliki konten yang berbeda. |

* 1. sha256sum

|  |
| --- |
| sha256sum message1.bin message2.bin |
| Penjelasan :  Output dari sha256sum adalah hash nilai SHA-256 dari setiap berkas. Hash nilai SHA-256 adalah nilai hash yang dihasilkan dari algoritma fungsi hash SHA-256 (Secure Hash Algorithm 256-bit).   * Hash nilai SHA-256 dari message1.bin adalah : 54bcb9a4fda31e4f254303e3959acd5e420ad18a80949d56a3000c3716fbd1a0. * Hash nilai SHA-256 dari message2.bin adalah : 90774a6455a2bdb7d106e533923ecbefe81392ca55bed0ce81cfab2c1a7f0afe.   Kedua hash nilai berbeda, artinya kedua berkas tersebut memiliki konten yang berbeda. | | |

* 1. md5sum

|  |
| --- |
| md5sum message1.bin message2.bin |

|  |
| --- |
| Penjelasan :  Output md5sum adalah hash nilai MD5 dari setiap berkas yang diberikan. Hash nilai MD5 adalah nilai hash yang dihasilkan dari algoritma fungsi hash MD5 (Message-Digest Algorithm 5).   * Hash nilai MD5 dari message1.bin adalah 008ee33a9d58b51cfeb425b0959121c9. * Hash nilai MD5 dari message2.bin juga adalah 008ee33a9d58b51cfeb425b0959121c9.   Faktor kekurangan md5sum adalah memiliki kekurangan dalam menangani hash collision, di mana dua input berbeda dapat menghasilkan nilai hash yang sama. Hal ini menyebabkan mengapa nilai hash md5sum untuk message1.bin dan message2.bin sama. |

1. Analisislah output dari nomor tiga. Menurutmu, mengapa hal tersebut bisa terjadi? Hash function mana yang lebih aman digunakan?

|  |
| --- |
| Pada output md5sum, nilai hash untuk message1.bin dan message2.bin sama, yaitu 008ee33a9d58b51cfeb425b0959121c9. Ini dikarenakan algoritma md5sum memiliki kelemahan yang disebut **hash collision**.  Hash collision terjadi ketika dua input yang berbeda menghasilkan nilai hash yang sama. Dalam kasus ini, message1.bin dan message2.bin memiliki konten yang berbeda, namun menghasilkan nilai md5sum yang sama. Ini terjadi karena algoritma md5sum tidak memiliki distribusi nilai hash yang merata.  Kelemahan hash collision pada md5sum membuatnya rentan terhadap serangan. Contohnya, seorang penyerang dapat membuat dua file yang berbeda dengan nilai md5sum yang sama. Salah satu file dapat dibuat sebagai file yang berbahaya, dan file lainnya dibuat sebagai file yang tampaknya sah. Penyerang kemudian dapat mendistribusikan file yang berbahaya dengan nama file yang sah, menipu pengguna untuk mengunduhnya dan menjalankannya.  Alasan md5sum Kurang Aman:  Hash collision: Kelemahan utama md5sum adalah kerentanannya terhadap hash collision.  Preimage attack: Preimage attack memungkinkan penyerang untuk menemukan input yang menghasilkan nilai hash md5sum tertentu. Hal ini dapat digunakan untuk memalsukan tanda tangan digital atau memverifikasi file yang berbahaya.  Second preimage attack: Second preimage attack memungkinkan penyerang untuk menemukan input lain yang menghasilkan nilai hash md5sum yang sama dengan input yang dipilih. Hal ini dapat digunakan untuk menyembunyikan file berbahaya dalam file lain.  Hash function yang lebih aman:  Karena kelemahan md5sum, disarankan untuk menggunakan hash function yang lebih aman untuk integritas data dan keamanan yang kuat, yaitu :   * SHA-256: Algoritma hash yang populer dan aman dengan output 256 bit. SHA-256 lebih tahan terhadap hash collision dan serangan lainnya dibandingkan md5sum. * SHA-3: Algoritma hash terbaru dari NIST (*National Institute of Standards and Technology*) dengan output 256 bit. SHA-3 menawarkan tingkat keamanan yang lebih tinggi dibandingkan SHA-256 dan direkomendasikan untuk aplikasi yang membutuhkan keamanan terbaik. |

## Encoding

1. Pastikan python telah terinstall di perangkat yang digunakan

|  |
| --- |
| python3 --version |

|  |
| --- |
|  |

1. Buka file dengan nama encode.py (file akan langsung terbuat), lalu copy-paste kode berikut:

|  |
| --- |
| import binascii |
| import base64 |
|  |
| text = "Road to Security Engineer" |
| text\_utf8 = text.encode("utf-8") |
| text\_b64 = base64.b64encode(text\_utf8) |
| text\_hex = binascii.hexlify(text\_utf8) |
|  |
| print("text = " + text) |
| print("utf8 encode = " + str(text\_utf8)) |
| print("base64 encode = " + str(text\_b64)) |
| print("hexadec encode = " + str(text\_hex)) |

Apa yang dilakukan kode tersebut ?

|  |
| --- |
| Penjelasan :  Tujuan kode tersebut adalah mengilustrasikan proses enkode teks menggunakan beberapa metode yang umum digunakan dalam pemrosesan data, yaitu UTF-8, base64, dan hexadecimal encoding.  Dengan menggunakan kode tersebut, kita dapat melihat bagaimana teks "Road to Security Engineer" diubah menjadi representasi yang berbeda-beda.   * Konversi ke UTF-8: Teks string diubah menjadi urutan byte UTF-8 menggunakan metode encode("utf-8"). * Base64 Encoding: Teks string yang diubah menjadi urutan byte UTF-8 dienkripsi menggunakan base64 encoding menggunakan fungsi base64.b64encode(). * Hexadecimal Encoding: Teks string yang diubah menjadi urutan byte UTF-8 dienkripsi ke dalam format hexadecimal menggunakan fungsi binascii.hexlify(). |

1. Tambahkan fungsionalitas untuk mengembalikan base64 dan hexadecimal encode pada source code python tersebut menjadi semula.

|  |
| --- |
| import binascii |
| import base64 |
|  |
| text = "Road to Security Engineer" |
| text\_utf8 = text.encode("utf-8") |
| text\_b64 = base64.b64encode(text\_utf8) |
| text\_hex = binascii.hexlify(text\_utf8) |
|  |
| # Decode base64 |
| decoded\_base64 = base64.b64decode(text\_b64).decode("utf-8") |
|  |
| # Decode hexadecimal |
| decoded\_hex = binascii.unhexlify(text\_hex).decode("utf-8") |
|  |
| print("text = " + text) |
| print("utf8 encode = " + str(text\_utf8)) |
| print("base64 encode = " + str(text\_b64)) |
| print("base64 decode = " + decoded\_base64) # Menambahkan base64 decode |
| print("hexadec encode = " + str(text\_hex)) |
| print("hexadec decode = " + decoded\_hex) # Menambahkan hexadecimal decode |

|  |
| --- |
|  |

## Encoding

1. Pastikan python telah terinstall di perangkat yang digunakan

|  |
| --- |
| python3 --version |

|  |
| --- |
|  |

1. Buka file dengan nama caesar.py (file akan langsung terbuat), lalu copy-paste kode

berikut:

|  |
| --- |
| #!/usr/bin/python3 |
|  |
| plaintext = "Infosec" |
| ciphertext = "" |
| for i in range(len(plaintext)): |
| if(plaintext[i].isupper()): |
| ciphertext += chr((ord(plaintext[i]) + 7 - 65) % 26 + 65) |
| else: |
| ciphertext += chr((ord(plaintext[i]) + 7 - 97) % 26 + 97) |
| print("plaintext = " + plaintext) |
| print("ciphertext = " + ciphertext) |

|  |
| --- |
|  |

1. Apa yang terjadi pada line 7 dan 9? Mengapa ada angka 65, 97, dan 26 disana?

|  |
| --- |
| Line 7 dan 9 pada kode melakukan operasi enkripsi Caesar pada huruf-huruf dalam plaintext.  **Line 7**  Jika karakter dalam plaintext adalah huruf kapital (uppercase), dilakukan operasi : ciphertext += chr((ord(plaintext[i]) + 7 - 65) % 26 + 65)  65 untuk mengubah posisi Unicode kembali ke posisi huruf A dalam ASCII (65 adalah nilai Unicode untuk huruf A dalam huruf kapital).  % 26 untuk memastikan bahwa nilai yang dihasilkan tetap dalam rentang 0 hingga 25 (0-25 mengelilingi A-Z dalam alfabet).  **Line 9**  Jika karakter dalam plaintext adalah huruf kecil (lowercase), dilakukan operasi :  ciphertext += chr((ord(plaintext[i]) + 7 - 97) % 26 + 97)  97 digunakan untuk mengubah posisi Unicode kembali ke posisi huruf a dalam ASCII (65 adalah nilai Unicode untuk huruf a).  % 26 untuk memastikan bahwa nilai yang dihasilkan tetap dalam rentang 0 hingga 25 (0-25 mengelilingi a-z dalam alfabet). |

1. Kriptografi apakah kode tersebut? Bagaimana caranya bekerja?

|  |
| --- |
| Kriptografi tersebut merupakan contoh sederhana dari **metode enkripsi Caesar**. Dinamakan demikian karena Julius Caesar, yang menurut Suetonius, menggunakannya dengan pergeseran tiga huruf (A menjadi D saat dienkripsi, dan D menjadi A saat didekripsi) untuk melindungi pesan-pesan penting militer. Meskipun Caesar adalah pengguna pertama yang tercatat menggunakan skema ini, sandi substitusi lain diketahui telah digunakan sebelumnya.  *“Jika ia memiliki suatu rahasia yang akan disampaikan, ia menuliskannya dalam sandi, dengan mengganti urutan abjad, sehingga tak satu kata pun dapat dimengerti. Jika ada yang ingin membaca pesan-pesan ini, ia harus mengganti huruf keempat dalam alfabet, yaitu D, untuk A, dan seterusnya untuk huruf-huruf lain.”* - Suetonius, Life of Julius Caesar  Metode Caesar bekerja dengan menggeser setiap huruf dalam teks plaintext sejumlah langkah tertentu dalam alfabet. Dalam contoh kode tersebut, setiap huruf digeser sebanyak 7 langkah ke depan. Proses ini menghasilkan teks ciphertext yang nantinya dapat dikirimkan secara aman.   * Jika karakter adalah **huruf kapital**, kode menambahkan karakter hasil enkripsi ke dalam ciphertext menggunakan rumus enkripsi Caesar untuk huruf kapital. * Jika karakter adalah **huruf kecil**, kode menambahkan karakter hasil enkripsi ke dalam ciphertext menggunakan rumus enkripsi Caesar untuk huruf kecil. * Jika karakter **bukan huruf**, kode tidak melakukan enkripsi dan langsung menambahkannya ke dalam ciphertext. |

1. Optimalkan cara kerja algoritma kriptografi tersebut! Ambil input dari user dan juga shift lompatan angka dari user (interactive) !

|  |
| --- |
| plaintext = input("Masukkan teks plaintext: ") |
| shift = int(input("Masukkan jumlah langkah geser: ")) |
|  |
| ciphertext = "" |
| for char in plaintext: |
| if char.isupper(): |
| ciphertext += chr((ord(char) + shift - 65) % 26 + 65) |
| elif char.islower(): |
| ciphertext += chr((ord(char) + shift - 97) % 26 + 97) |
| else: |
| ciphertext += char |
|  |
| print("plaintext = " + plaintext) |
| print("ciphertext = " + ciphertext) |

|  |
| --- |
| Penjelasan :  Kode ini, menggunakan fungsi input() untuk mengambil input dari user. User diminta untuk memasukkan teks plaintext dan jumlah langkah geser. Variabel shift akan berisi jumlah langkah geser yang dimasukkan oleh pengguna.  Kemudian, kode melakukan loop melalui setiap karakter dalam plaintext dan melakukan enkripsi sesuai dengan jumlah langkah geser yang ditentukan oleh user. Hasil enkripsi disimpan dalam variabel ciphertext.  Setelah loop selesai, teks plaintext dan ciphertext dicetak untuk ditampilkan kepada pengguna. Sehingga, algoritma kriptografi Caesar menjadi lebih interaktif dan lebih mudah digunakan user. |

## Tugas

1. Jelaskan perbedaan hash, enkripsi, dan encoding!

|  |
| --- |
| **Hashing:**   * Hashing adalah proses mengubah data (teks, file, dll.) menjadi nilai hash yang unik. * Tujuan utama hashing adalah untuk menghasilkan nilai hash yang konsisten untuk setiap input yang diberikan. * Hash function digunakan untuk menghasilkan nilai hash, yang biasanya memiliki panjang yang tetap, tidak peduli seberapa besar atau kecil data inputnya. * Contoh algoritma hashing termasuk MD5, SHA-1, SHA-256, dan sebagainya. * Hasil hashing tidak dapat diubah kembali menjadi data asli (tidak reversibel). * Hashing sering digunakan untuk mengamankan kata sandi, verifikasi integritas data, dan pembuatan identifikasi unik untuk data.   **Enkripsi:**   * Enkripsi adalah proses mengubah data asli menjadi bentuk yang tidak terbaca atau tidak dapat dimengerti tanpa kunci enkripsi yang sesuai. * Tujuan utama enkripsi adalah untuk menyembunyikan atau melindungi data dari akses yang tidak sah. * Enkripsi menggunakan algoritma enkripsi dan kunci enkripsi untuk mengubah data menjadi bentuk terenkripsi yang aman. * Hasil enkripsi dapat didekripsi kembali menjadi data asli menggunakan kunci yang tepat. * Enkripsi sering digunakan untuk melindungi informasi sensitif dalam penyimpanan atau transmisi, seperti data pengguna, pesan rahasia, dan sebagainya.   **Encoding:**   * Encoding adalah proses mengubah data dari satu format ke format lain, biasanya dalam konteks representasi teks. * Tujuan utama encoding adalah untuk memastikan bahwa data dapat disampaikan dan disimpan dengan benar tanpa kehilangan informasi. * Encoding tidak menyembunyikan data atau melindunginya dari akses yang tidak sah; hanya mengubah cara data direpresentasikan. * Beberapa jenis encoding umum termasuk Base64, ASCII, UTF-8, dan sebagainya. * Hasil encoding dapat didekode kembali ke bentuk aslinya tanpa kehilangan informasi. * Encoding sering digunakan dalam pertukaran data, komunikasi, dan penyimpanan data untuk memastikan konsistensi format dan interoperabilitas. |

1. Diberikan source code berikut, buatlah dekripsi dalam bahasa pemrograman python:

|  |
| --- |
| encrypt.py |
| #!usr/bin/python3 |
| def encrypt(plaintext): |
| plaintext = plaintext[::-1] |
| ciphertext = "" |
| for i in plaintext: |
| copy = "X" \* ((ord(i) ^ 0x50) + 9) |
| copy += "-" |
| ciphertext += copy |
| return ciphertext |
|  |
|  |
| print ("Infosec Module Encryptor") |
| plaintext = input("Masukkan string yang ingin di-encrypt: ") |
| print ("Result : ") |
| print (encrypt(plaintext)) |

|  |
| --- |
| decrypt.py |
| def decrypt(ciphertext): |
| ciphertexts = ciphertext.split('-') |
| decrypted = '' |
| for text in ciphertexts: |
| countx = 0 |
| for x in text: |
| countx += 1 |
| if (countx!=0): |
| decrypted += chr((countx-9^0x50)) |
| decrypted = decrypted[::-1] |
| return decrypted |
|  |
| print ("Infosec Module Decryptor") |
| ciphertext = input("Masukkan string yang ingin di-decrypt: ") |
| print ("Result :", decrypt(ciphertext)) |

|  |
| --- |
| **encrypt.py**        **decrypt.py** |

1. Jelaskan source code yang telah anda buat pada nomor 2

|  |
| --- |
| Source decrypt.py di atas adalah implementasi dari fungsi decrypt dalam bahasa Python. Tujuan dari source code diatas adalah untuk menyediakan fungsi yang dapat mendekripsi teks yang telah dienkripsi oleh fungsi. Dengan fungsi ini, kita dapat mengembalikan teks plaintext dari teks ciphertext yang dihasilkan oleh encrypt.py. |

1. Jelaskan perbedaan antara
   1. Asymmetric encryption dan symmetric encryption

|  |
| --- |
| 1. Symmetric Encryption:  * Dalam symmetric encryption, kunci yang sama digunakan untuk enkripsi dan dekripsi data. * Lebih cepat daripada enkripsi asimetris karena kunci yang digunakan lebih pendek. * Kekurangannya adalah kunci harus dibagikan secara aman antara pengirim dan penerima. * Contoh algoritma simetris termasuk AES (Advanced Encryption Standard) dan DES (Data Encryption Standard).  1. Asymmetric Encryption:  * Dalam asymmetric encryption, ada sepasang kunci: kunci publik dan kunci pribadi. Kunci publik digunakan untuk enkripsi dan dapat dibagikan secara publik, sementara kunci pribadi digunakan untuk dekripsi dan harus disimpan secara rahasia. * Enkripsi asimetris lebih aman karena tidak memerlukan pertukaran kunci rahasia. Namun, prosesnya lebih lambat daripada enkripsi simetris. * Contoh algoritma asimetris termasuk RSA (Rivest-Shamir-Adleman) dan ECC (Elliptic Curve Cryptography). |

* 1. Stream cipher dan block cipher

|  |
| --- |
| 1. Stream Cipher:  * Stream cipher mengenkripsi data bit demi bit atau byte demi byte. * Stream cipher menggunakan aliran kunci acak yang dihasilkan secara bersamaan dengan proses enkripsi untuk mengacak data. * Cocok untuk mengenkripsi data secara real-time seperti komunikasi VoIP atau video streaming. * Contoh stream cipher termasuk RC4 (Rivest Cipher 4) dan ChaCha20.  1. Block Cipher:  * Block cipher mengenkripsi blok data yang lebih besar, biasanya dalam ukuran tetap, seperti 64 bit atau 128 bit. * Data dipecah menjadi blok-blok yang sama ukurannya sebelum dienkripsi. * Cocok untuk mengenkripsi data yang disimpan, seperti file atau pesan yang akan dikirim. * Contoh block cipher termasuk AES (Advanced Encryption Standard) dan DES (Data Encryption Standard). |

**Kesimpulan**

Hashing:

* Hashing diawali dengan memastikan md5sum terinstal dan melakukan hashing terhadap dua file dengan algoritma SHA-1, SHA-256, dan MD5.
* Pada output md5sum, nilai hash untuk message1.bin dan message2.bin sama, yaitu 008ee33a9d58b51cfeb425b0959121c9. Hal ini dikarenakan algoritma md5sum memiliki kelemahan yang disebut hash collision.

Encoding:

* Kode Python encode.py mengonversi teks ke dalam format UTF-8, base64, dan hexadecimal.
* Fungsionalitas tambahan ditambahkan untuk mengembalikan teks dari base64 dan hexadecimal encode.

Enkripsi:

* Source code caesar.py melakukan enkripsi plaintext dengan menggunakan algoritma Caesar Cipher.

Dekripsi:

* decrypt.py ditambahkan untuk mendekripsi teks yang telah dienkripsi.

**Evaluasi**

* Rangkaian tugas praktikum kriptografi ini memberikan pemahaman yang baik tentang konsep dasar kriptografi, seperti hashing, enkripsi, dan encoding.
* Implementasi praktis dalam bahasa pemrograman Python membantu untuk memahami cara kerja algoritma dan teknik kriptografi.
* Evaluasi keseluruhan menunjukkan pemahaman yang kuat tentang konsep-konsep kriptografi yang diuji dalam praktikum BAB 5 Kriptografi.