Eksplorasi Skema Enkripsi Data Sederhana Menggunakan Jarak Edit Levenshtein

Richard Christian - 13523024
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: richsukandar1@gmail.com , 13523024@std.stei.itb.ac.id

***Abstract*—Penelitian ini mengeksplorasi alternatif penggunaaan Jarak Edit Levenshtein sebagai metode alternatif untuk enkripsi data eksperimental sederhana. Informasi disisipkan dengan memanfaatkan kemiripan struktural antara dua string, dengan menggunakan bantuan kunci string simetris.**

Hasil awal menunjukkan keberhasilan dalam memenuhi aspek keamanan dasar kriptografi, seperti kerahasiaan, diffusion, confusion, dan sifat deterministik dan reversibel. Akan tetapi, tetap ada kelemahan signifikan pada efisiensi enkripsi dan kerentanan kriptanalisis. Untuk mengatasi kerentanan ini, diterapkan implementasi Sliding Key Window, dimana bentuk kunci menjadi lebih dinamis, disertai tambahan minimum edit distance. Tambahan lapisan keamanan ini cukup sukses untuk mengingkatkan keamanan kriptanalisis, tetapi efisiensi enkripsi khususnya pada ukuran hasil enkripsi masih menjadi tantangan untuk aplikasi enkripsi berbasis Jarak Edit Levenshtein.

Keywords—Jarak Edit Levenshtein, Enkripsi, Kunci Simetris, Eksperimental

I. Pendahuluan

Dalam bidang informasi dan komunikasi, algoritma transformasi data memiliki peran yang penting untuk dapat menjaga integritas dan kerahasiaan pada transmisi dan penyimpanan data. Salah satu bentuk transformasi data yang paling umum digunakan untuk keamanan adalah enkripsi data, dimana informasi diubah menjadi bentuk yang tidak dapat dibaca atau dipahami tanpa menggunakan kunci tertentu. Enkripsi modern seperti RSA dan AES dirancang dengan prinsip kriptografi yang kuat dan memiliki banyak penggunaan dalam bidang keamanan data.

Selain enkripsi modern, terdapat juga berbagai algoritma kriptografi yang lebih sederhana seperti Caesar Cipher, substitution Cipher, dan XOR Cipher. Teknik-teknik ini jauh lebih sederhana dibandingkan metode enkripsi modern, tetapi memiliki banyak *vulnerability* sehingga kini sudah kurang relevan. Walaupun begitu, teknik-teknik ini masing sering digunakan sebagai media pembelajaran dan eksplorasi algoritma kriptografi.

Eksperimen ini berfokus pada eksplorasi enkripsi data ringan menggunakan algoritma *Levenshtein Distance,* yang mengukur seberapa mirip suatu data string dengan string lainnya. Algoritma ini biasanya digunakan untuk melakukan *string matching,* yaitu untuk mengukur kemiripan antara 2 buah string. Informasi ini lalu biasanya digunakan lagi untuk apliklasi algoritma seperti dalam *autocorrect* maupun fuzzy search.

Pada eksperimen ini, sifat *Levenshtein Distance* akan dimanfaatkan sebagai alternatif dalam enkripsi, dimana informasi disimpan dalam bentuk kemiripan antara 2 ciphertext dan sebuah kata sebagai kunci. Perlu ditekankan bahwa hasil enkripsi pada eksperimen ini tidak ditujukan untuk digunakan sebagai metode enkripsi modern, melainkan sebagai eksplorasi salah satu alternatif enkripsi unik, dan masih memiliki berbagai kelemahan dalam penerapannya.

II. Dasar Teori

A. Enkripsi Data

Enkripsi adalah proses transformasi data dari bentuk semula yang dapat dibaca menjadi bentuk ciphertext untuk menyembunyikan informasi. Proses pembuatan ciphertext atau enkripsi biasanya dilakukan menggunakan suatu algoritma tertentu, dengan bantuan suatu kunci. Hasil ciphertext adalah hasil data yang memerlukan kunci untuk melakukan dekripsi data, yaitu menerjemahkan hasil enkripsi kembali ke data yang dapat dibaca. Tujuan dari proses ini adalah untuk menjaga kerahasiaan informasi dalam penyimpanan maupun dalam proses transmisi data.

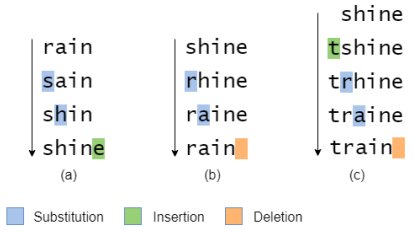
Dalam enkripsi data, terdapat 2 jenis enkripsi utama yang dibedakan melalui bentuk kunci yang digunakan, yaitu:

- *Symmetric Encryption*, dimana pengirim dan penerima akan menggunakan kunci yang sama untuk melakukan proses enrkipsi dan dekripsi data.
- *Assymmetric Encryption*, dimana ada 2 kunci berbeda, yaitu kunci publik yang digunakan untuk melakukan enkripsi, dan kunci privat untuk melakukan dekripsi[2].

B. Levenshtein Distance

Levenshtein Distance adalah cara mengukur kemiripan antara dua buah string, dengan menghitung jumlah operasi yang diperlukan untuk mengubah salah satu string menjadi string yang diinginkan. Terdapat 3 buah operasi yang dapat

diaplikasikan pada Levenshtein Distance, yaitu *Insertion, Deletion,* dan *Substitution.* Levenshtein distance banyak digunakan untuk aplikasi string matching, seperti pada koreksi otomatis, *fuzzy search*, dan pemrosesan data [1].



Gambar 1. Contoh Levenshtein Distance

(Sumber: https://needjarvis.tistory.com/801)

C. *Keamanan Data dalam Kriptografi*

Keamanan data menjadi aspek utama yang perlu diprioritaskan dalam kriptografi. Untuk memenuhi nilai ini, ada beberapa poin-poin penting yang perlu dipenuhi oleh suatu algoritma untuk dapat dianggap sebagai algoritma enkripsi yang baik, beberapa diantaranya yaitu [3,4,5]:

- Kerahasiaan : Data tidak dapat ditebak dengan mudah walaupun algoritma enkripsi yang digunakan sudah diketahui, sehingga memerlukan kunci untuk dapat mengakses data.
- *Diffusion and Confusion*: Difusi dalam sebuah enkripsi berarti data dari enkripsi tersebar pada hasil cipherteks, sehingga mempersulit analisis secara statistik. *Confusion* berarti hubungan antara kunci dan cipherteks juga tidak terkait secara langsung, sehingga hubungannya sulit untuk dianalisis.
- Ketahanan Kriptanalisis: Algoritma enkripsi yang baik harus tahan terhadap serangan-serangan kriptografi yang umum, seperti dengan *brute force* dan *frequency analysis.*
- Efisiensi enkripsi dan algoritma: Algoritma enkripsi yang baik dapat diaplikasikan secara efisien sesuai kebutuhannya, dan hasil ciphertext juga tidak jauh lebih besar dari data aslinya.
- Deterministik dan reversibel: Proses enkripsi dapat memberikan hasil yang sama bila input data dan kunci yang digunakan sama, dan prosesnya juga harus dapat reversibel sesuai kunci yang digunakan.

III. IMPLEMENTASI

A. *Program dan Library*

Pembuatan program utama menggunakan Python3. Python dipilih karena kemudahan penggunaannya dan bentuk program yang cukup sederhana. Python juga mendukung library rapidfuzz yang akan digunakan untuk melakukan perhitungan *Levenshtein Distance* pada string. Library ini dipilih karena perhitungan menggunakan algoritma Myers' yang lebih efisien dibandingkan dengan perhitungan *Levenshtein* konvensional.

B. *Batasan implementasi*

Untuk mempermudah implementasi program dan pengecekan hasil enkripsi, program yang diimplementasikan diberikan beberapa batasan tertentu. Pembatasan ini diperlukan dan sesuai dengan konteks implementasi yang berbentuk eksperimental dan masih *proof-of-concept*. Beberapa batasan yang diaplikasikan yaitu:

- Proses enkripsi data sederhana dengan membandingkan jarak levenshtein distance secara langsung, sehingga hasil enkripsi bukan merupakan representasi penyisipan data enkripsi paling optimal. Bentuk enkripsi ini memaksimalkan kemudahan evaluasi program, pemahaman hasil program, dan ilustrasi kasus.
- Set karakter yang digunakan untuk enkripsi dan dekripsi dibatasi menjadi *printable characters* pada ASCII saja, dengan pengecualian beberapa karakter yang jarang digunakan pada sistem modern untuk membatasi jumlah set karakter dibawah 100 karakter unik. Untuk proses enkripsi lebih terbatas dengan tidak menggunakan whitespace sama sekali agar penanganan output lebih mudah.

```
CHARSET_ENCODE = string.printable.strip() # no whitespace

original_charset = string.printable
excluded = {'\x0b', '\x0c'}
CHARSET = ''.join(ch for ch in original_charset if ch not in excluded)
char_to_index = {ch: i for i, ch in enumerate(CHARSET)}
index_to_char = {i: ch for ch, i in char_to_index.items()}
```

Gambar 2. Set Karakter Program

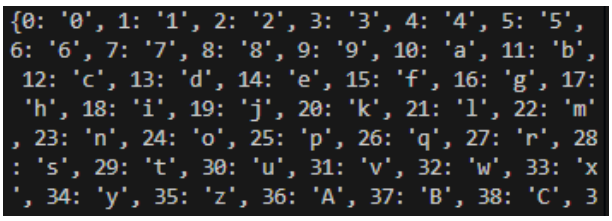
(Sumber: Dokumentasi Pribadi)

C. *Konsep Program*

Implementasi enkripsi fokus pada aplikasi *Levenshtein Distance* dengan menyimpan data melalui perbedaan jarak antara ciphertext yang dihasilkan algoritma menggunakan kunci. Pengguna program akan dapat memasukkan input kunci yang ingin digunakan, kemudian teks yang ingin dienkripsi oleh program, dengan kriteria keduanya termasuk dalam batasan set karakter yang dijelaskan sebelumnya.

Set karakter sengaja dibatasi dengan ukuran kurang dari 100, sehingga setiap kemungkinan karakter dapat direpresentasikan sebagai gabungan antara Levenshtein Distance dari 2 kata berbeda hasil enkripsi kunci, dengan kata pertama sebagai puluhan dan kata kedua sebagai satuan. Berikut adalah contoh ide hasil enkripsi program:

Potongan set karakter:



Gambar 3. Hasil Indeks Set Karakter Program

(Sumber: Dokumentasi Pribadi)

Untuk merepresentasikan kata "abc", maka program enkripsi akan membuat 6 buah kata, dengan *levenshtein distance* terhadap kunci awal masing-masing 1, 0, 1, 1, 1, dan 2 secara berurutan. Bila kunci yang digunakan berupa "tes", maka program dapat memberikan output "te", "tes", "tess", "aes", "tee", dan "ta".

Program dekripsi lalu akan menerima kunci yang sama, kemudian menghitung levenshtein distance dari masing-masing kata untuk kembali mendapatkan indeks dari set karakter. Hasil penggabungan akan kembali menghasilkan distance yang sama sebelum proses enkripsi, yang dikelompokkan menjadi pasangan 10, 11, dan 12.

D. Program Enkripsi dan Dekripsi

```python
#key -> input string
#dist -> target Levenshtein distance
#return string result
def encode(key: str, dist: int):
    if dist == 0:
        return key

    result = key
    current_dist = 0
    failed_count = 0

    while current_dist < dist:
        possible_operations = []
        possible_operations.append('insert')

        if len(result) > 0:
            possible_operations.append('delete')
            possible_operations.append('substitute')

        operation = random.choice(possible_operations)
        new_result = None
        if operation == 'insert': ...

        elif operation == 'delete': ...

        elif operation == 'substitute': ...

        if new_result is not None: ...
        else:
            failed_count += 1 # fail edit

        # fallback for when edits fail
        if current_dist < dist and failed_count > 19: ...

    return result
```

Gambar 4. Fungsi Encode

(Sumber: Dokumentasi Pribadi)

Inti dari program enkripsi terdapat pada fungsi encode. Fungsi encode akan menerima input kunci yang diinginkan dan target jarak *Levenshtein Distance* dari kunci. Program encode yang diimplementasikan akan mengaplikasikan *insert, delete,* dan *substitute* pada kunci dengan probabilitas yang sama.

Program bekerja dengan menambahkan edit pada kunci satu demi satu hingga jarak yang diinginkan tercapai. Hasil algoritma ini dapat terjamin memiliki jarak yang sesuai dengan mengecek apakah setiap operasi edit berhasil menambahkan jarak *Levenshtein Distance* pada teks sebelum mengaplikasikannya dan menambahkan variabel counter jarak. Ditambahkan alur fallback untuk membantu membatasi iterasi program saat edit yang diaplikasikan terus gagal untuk menambahkan edit distance.

Terdapat juga beberapa fungsi bantuan seperti char_to_distance, yang digunakan untuk mengkonversikan indeks pada charset menjadi 2 satuan jarak yang akan diterima oleh fungsi encode. Fungsi encode_message lalu menggabungkan kedua fungsi ini untuk menghasilkan serangkaian kata ciphertext, dengan setiap pasangan kata dipisah dengan newline dan antara pasangan kata dipisah dengan dua buah newline.

```python
def char_to_distances(char: str):
    index = char_to_index[char]
    tens = index // 10
    units = index % 10
    return (tens, units)

def encode_message(message: str, key: str):
    encoded_pairs = []

    for char in message:
        tens, units = char_to_distances(char)
        word1 = encode(key, tens)
        word2 = encode(key, units)
        encoded_pairs.append(f"{word1}\n{word2}")
    return "\n\n".join(encoded_pairs)
```

Gambar 5. Fungsi Bantuan Encode

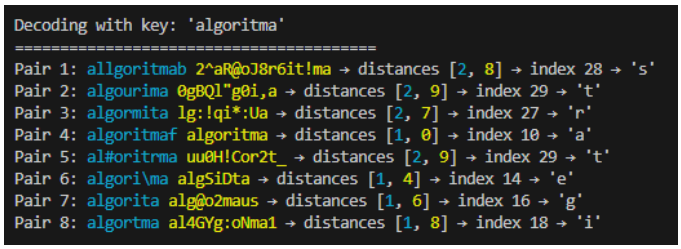
(Sumber: Dokumentasi Pribadi)

Proses pada program dekripsi jauh lebih sederhana, karena hanya perlu untuk menginterpretasi hasil output dari program enkripsi. Program dekripsi akan menerima kunci yang digunakan untuk enkripsi, kemudian ciphertext yang ingin didekripsi. Program dekripsi akan secara otomatis menangani hasil output dari program enkripsi, dan memisahkan setiap pasangan kata. Program dekripsi lalu menghitung kembali Levenshtein Distance setiap kata dengan kunci, dan mencocokkannya kembali dengan set karakter yang sudah didefinisikan sebelumnya.

IV. HASIL DAN PEMBAHASAN

A. Hasil Awal

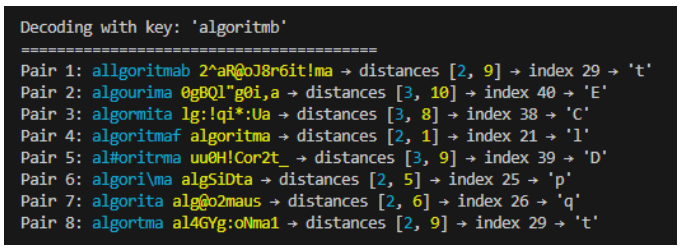
Percobaan 1:
Key = algoritma
Message = strategi



Gambar 5. Hasil Dekripsi 1

(Sumber: Dokumentasi Pribadi)

Key salah = algoritmb



Gambar 6. Hasil Dekripsi 1, Kunci Salah

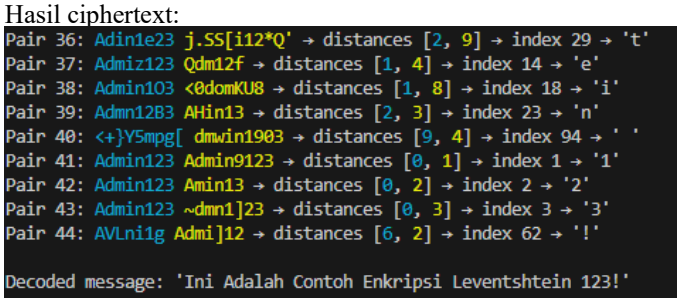
(Sumber: Dokumentasi Pribadi)

Percobaan 2:

Key = Admin123

Message = Ini Adalah Contoh Enkripsi Leventshtein 123!

Hasil ciphertext:



Gambar 7. Hasil Dekripsi 2

(Sumber: Dokumentasi Pribadi)

B. Pembahasan

Berdasarkan hasil percobaan dari ciphertext 1 dan 2, ciphertext dengan menggunakan *Levenshtein Distance* cukup berhasil untuk dapat mengenkripsi data. Isi data yang terenkripsi cukup sulit untuk dapat dipecahkan, khususnya saat distance dari kunci cukup tinggi. Hasil enkripsi juga sesuai dengan aspek-aspek keamanan data kriptografi, yaitu pada poin kerahasiaan, *diffusion and confusion*, dan sifat deterministik dan reversibel. Akan tetapi, percobaan juga menunjukkan 2 kelemahan besar dari sistem enkripsi ini, yaitu:

- Efisiensi enkripsi

  Program membutuhkan overhead pemrosesan yang cukup tinggi dibandingkan sistem enkripsi sederhana lainnya, seperti XOR Cipher, dan hasil ciphertext yang dihasilkan juga jauh lebih besar dari data asli yang berhasil dienkripsi. Ini membuat enkripsi dengan Levenshtein Distance sangat tidak praktis untuk penggunaan nyata.

- Keamanan Kriptanalisis

  Penggunaan kunci yang sama, dan penerapan algoritma generator Levenshtein Distance, khususnya dalam kasus jarak yang rendah, menyebabkan kunci rentan untuk dapat diekstraksi dari hasil-hasil enkripsi. Dapat terlihat bahwa penggunaan kunci seperti 'algoritma' sangat terlihat saat jarak yang di-generate kecil. Ini menyebabkan skema enkripsi ini menjadi sangat rentan terhadap kriptanalisis, bahkan kuncinya dapat ditebak hanya dengan memperkirakan pola kata yang sering muncul.

Untuk masalah efisiensi enkripsi, memang menjadi kelemahan terbesar dari aplikasi enkripsi berbasis kemiripan struktural antara string. Informasi yang dapat disisipkan melalui jarak levenshtein sangat terbatas. Bahkan jika implementasi dioptimalkan untuk menyimpan lebih banyak data, peningkatan efisiensi enkripsi akan tetap minimal. Metode seperti ini sudah pasti akan boros, karena ukuran string pebanding yang dihasilkan hampir pasti akan jauh lebih besar daripada data yang dapat dienkripsi.

Untuk kriptanalisis, khususnya dalam keamanan kunci, dapat diaplikasikan beberapa solusi yang dapat mengurangi risiko bocornya kunci dari hasil enkripsi. Detil implementasi akan diperjelas pada bagian selanjutnya.

V. Implementasi Tambahan

A. Konsep dan Implementasi Sliding Key Window

Sliding Key Window adalah metode dimana kunci yang digunakan berbentuk dinamis, untuk meningkatkan kompleksitas enkripsi sehingga meningkatkan keamanan, khususnya untuk kriptanalisis kunci pada kasus ini. Penerapan sliding key window pada implementasi ini akan mencakup:

- Penggunaan kunci parsial, dimana hanya sebagian dari kunci yang digunakan untuk mengenkripsi data
- Penerapan kunci dinamis, dimana *window* kunci akan bergeser secara otomatis, baik dari index kunci dan ukuran kunci sehingga mencegah penggunaan 2 kunci sama dalam data yang berdekatan
- Perubahan kunci yang bergantung pada data, dimana *window* kunci dipengaruhi oleh hasil dekripsi data yang diterima, sehingga menambahkan kompleksitas.

```python
def load_key_from_file(filepath):
    try:
        with open(filepath, 'r', encoding='utf-8') as file:
            content = file.read()
        return ''.join(char for char in content if not char.isspace())
    except Exception as e:
        print(f"Error loading key file: {e}")
        return None

def sliding_key(key_string, current_index, size, pair_result=None):
    if pair_result is not None:
        tens = pair_result // 10
        units = pair_result % 10
        step = tens * units
        current_index = (current_index + step) % len(key_string)

        size_adjustment = tens + units
        size = size + size_adjustment
        while size > 10:
            size -= 5

    if current_index + size <= len(key_string):
        key_segment = key_string[current_index:current_index+size]
    else: #wrapping to start of index if exceeds key length
        first_part = key_string[current_index:]
        remaining = size - len(first_part)
        key_segment = first_part + key_string[:remaining]

    return key_segment, current_index, size
```

Gambar 8. Implementasi Sliding Key Window

(Sumber: Dokumentasi Pribadi)

Fungsi utama yang dibuat untuk mengimplementasi *Sliding Key Window* adalah fungsi sliding_key, yang digunakan pada sender dan receiver. Dalam implementasi ini, kunci akan berubah secara dinamis berdasarkan data indeks yang diterima. Indeks awal kunci akan ditambahkan sebesar hasil dekripsi indeks puluhan dikali indeks satuan. Sebagai contoh, bila data yang terakhir dikirim adalah data dengan indeks 27, maka kunci akan bergeser 14 indeks ke kanan. Diterapkan juga penanganan wrap-around bila kunci sudah digunakan habis, agar enkripsi bisa terus berlanjut menggunakan key yang sama.

Ukuran key juga ditentukan oleh indeks puluhan dan satuan, tetapi dengan perbedaan jenis operasi yang digunakan, yaitu penambahan antara keduanya. Ukuran key juga dibatasi sebesar 5 sampai 10, karena jarak *Levenshtein Distance* yang sudah secara tidak langsung dibatasi menjadi 9 karena metode implementasi. Dalam kasus yang sama, bila awalnya window size adalah 5, dan ditambah ukuran 2+7, maka window size akhir akan sebesar 9 karena dibatasi oleh ukuran key maksimum (14-5). Diterapkan juga minimum edit distance sebesar window size, untuk mengatasi masalah munculnya isi key pada saat jarak edit yang dienkripsi rendah.

B. Hasil dengan Sliding Key Window

Percobaan 1:
Key:
0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLM
NOPQRSTUVWXYZ!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~

Message: strategi

```
Enter message to encode: strategi
Enter key file path (e.g. key.txt): key.txt

Loaded key from 'key.txt' (94 characters)
Encoding 's' (index 28) with key[0:5] = '01234'
 → 3&4&DK? q-1vWaaQ7.!y)?
Encoding 't' (index 29) with key[16:26] = 'ghijklmnop'
 → gm!tg^('lFg&ano]p ,$'-hU.j{dk}lYHomFDn)an3[
Encoding 'r' (index 27) with key[34:40] = 'yzABCD'
 → zD|YCU0w~ O5B`j@#7$:Y3*
Encoding 'a' (index 10) with key[48:58] = 'MNOPQRSTUV'
 → ?Osd9qO3R2U =MO%Pc.'TTVsE]
Encoding 't' (index 29) with key[48:54] = 'MNOPQR'
 → 3eRHN^OR> |8$wIS+~v?OZPR}27
Encoding 'e' (index 14) with key[66:73] = '%&'()*+'
 → cN,]y(U:* 3w,0"zi(*rg<
Encoding 'g' (index 16) with key[70:77] = ')*+,-./'
 → N*+`u"^i&./J L&,irc]{i/8'O8
Encoding 'i' (index 18) with key[76:85] = '/:;<=>?@['
 → A/=wp>[@w`&6 ;;=T*o`n%BH:EF}o>
```

Gambar 9. Hasil Enkripsi Sliding Key 1

(Sumber: Dokumentasi Pribadi)

Key salah, tidak ada 0 pada bagian awal key:
123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLM
NOPQRSTUVWXYZ!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~

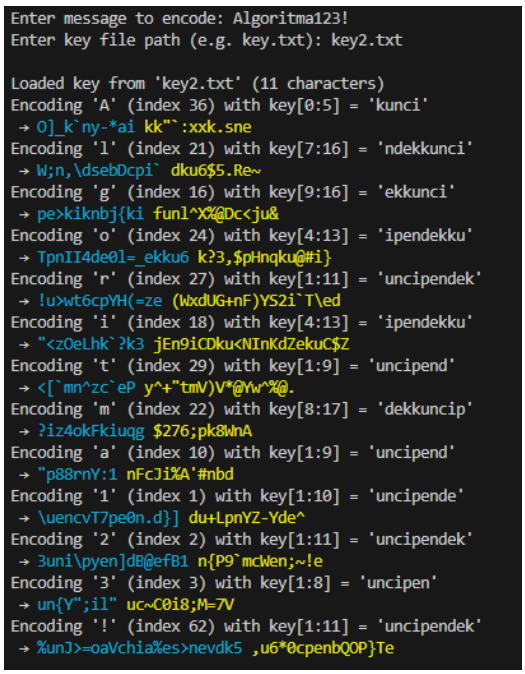
```
N*+`u"^i&./J
L&,irc]{i/8'O8

A/=wp>[@w`&6
;;=T*o`n%BH:EF}o>



Decoded message: 'sNeAk3VA'
```

Gambar 10. Hasil Dekripsi Sliding Key 1, Key Salah

(Sumber: Dokumentasi Pribadi)

Percobaan 2, pengujian wrapping kunci saat kunci habis:
Key: kuncipendek

Message: Algoritma123!



Gambar 11. Hasil Enkripsi Sliding Key 2

(Sumber: Dokumentasi Pribadi)

C. Pembahasan dengan Sliding Key Window

Hasil percobaan implementasi lanjutan menggunakan tambahan sliding key window cukup berhasil untuk meningkatkan kekuatan kriptanalisis dari algoritma enkripsi dengan cukup signifikan. Penambahan *minimum edit distance* juga memastikan bahwa kunci asli tidak lagi terlihat jelas pada hasil enkripsi, dan membuat panjang hasil key menjadi lebih variatif. Hasil implementasi ini juga tetap mempertahankan aspek keamanan yang sudah ada pada hasil enkripsi sebelumnya.

VI. KESIMPULAN

Eksperimen ini mencoba untuk mempelajari pemanfaatan alternatif dari *Levenshtein Distance* yang biasanya digunakan untuk pengolahan teks. Pada implementasi eksperimen ini, *Levenshtein Distance* digunakan untuk menyisipkan enkripsi data pada kemiripan struktur antara key dengan ciphertext yang dihasilkan. Implementasi diberikan beberapa batasan untuk dapat menyederhanakan implementasi sebagai bentuk *proof of concept*.

Hasil dari beberapa pengujian sederhana menunjukkan bahwa pengenkripsian data dengan metode ini cukup berhasil, dimana enkripsi memenuhi beberapa sifat-sifat umum dari enkripsi yang baik, seperti kerahasiaan, diffusion and confusion, dan sifat deterministik dan reversibel. Akan tetapi, implementasi sederhana ini juga menunjukan kelemahan pada efisiensi algoritma dan ukuran data hasil enkripsi, yang menjadi kelemahan terbesar untuk hasil enkripsi berbasis kemiripan struktur seperti pada implementasi ini.

Kelemahan lain yang ditemukan adalah kelemahan terhadap kriptanalisis, dimana key dapat direkonstruksi dari hasil-hasil enkripsi khususnya pada enkripsi dengan kemiripan struktur yang dekat. Untuk memitigasi hal ini, diimplementasikan *Sliding Key Window* agar kunci yang digunakan lebih dinamis sehingga lebih sulit untuk dapat menebak isi, posisi, dan ukuran kunci yang sedang digunakan. Penambahan minimum edit distance juga memastikan bahwa hasil enkripsi tidak akan mirip dengan key yang digunakan.

VII. LAMPIRAN

Kode implementasi enkripsi dapat diakses pada: https://github.com/doober22/makalah_enkripsi

REFERENSI

[1] M. Srivastava, "Introduction to Levenshtein Distance," *GeeksforGeeks*, [Tersedia]: https://www.geeksforgeeks.org/dsa/introduction-to-levenshtein-distance/. [Diakses: 23 Juni 2025]

[2] IBM, "What is encryption?" *IBM Think*, [Tersedia]: https://www.ibm.com/think/topics/encryption. [Diakses: 23 Juni 2025].

[3] A. Raj, "Encryption: Its Algorithms and Its Future," *GeeksforGeeks*, [Tersedia]: https://www.geeksforgeeks.org/encryption-its-algorithms-and-its-future/. [Diakses: 23 Juni 2025].

[4] "Making 'Good' Encryption Algorithms," *BrainKart*, [Tersedia]: https://www.brainkart.com/article/Making--Good--Encryption-Algorithms_9556/. [Diakses: 23 Juni 2025].

[5] C. Brook, "What Is Strong Encryption?" *eSecurityPlanet*, [Tersedia]: https://www.esecurityplanet.com/networks/strong-encryption/. [Diakses: 23 Juni 2025].

[6] D. Taneja, "Making Good Encryption Algorithms," *DrBTaneja.com*, [Tersedia]: https://drbtaneja.com/making-good-encryption-algorithms/. [Diakses: 23 Juni 2025].

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 Juni 2025

Richard Christian dan 13523024