

# **LAPORAN TUGAS BESAR III**

**IF2211 Strategi Algoritma**

**Pemanfaatan *Pattern Matching* dalam Membangun Sistem Deteksi Individu Berbasis Biometrik Melalui Citra Sidik Jari**



Disusun oleh:

**“PuntangPanting”**

Erdianti Wiga Putri Andini 13522053

Moh Fairuz Alauddin Yahya 13522057

Ellijah Darrellshane S 13522097

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**2024**

## DAFTAR ISI

<b>DAFTAR ISI.....</b>	<b>1</b>
<b>BAB I.....</b>	<b>3</b>
<b>DESKRIPSI TUGAS.....</b>	<b>3</b>
1.1 Deskripsi Tugas.....	3
1.2 Spesifikasi.....	3
<b>BAB II.....</b>	<b>5</b>
<b>LANDASAN TEORI.....</b>	<b>5</b>
2.1 Pattern Matching.....	5
2.2 Algoritma Boyer-Moore.....	5
2.3 Algoritma Knuth-Morris-Pratt.....	6
2.4 Regular Expression.....	7
2.5 Algoritma Levenshtein Distance.....	9
2.6 Penjelasan Singkat Aplikasi yang Dibuat.....	9
<b>BAB III.....</b>	<b>11</b>
<b>ANALISIS PEMECAHAN MASALAH.....</b>	<b>11</b>
3.1 Langkah Pemecahan Masalah secara Umum.....	11
3.2 Proses Pemilihan ASCII dan Evaluasi Pattern.....	11
3.3 Proses Penyelesaian Solusi dengan Algoritma KMP dan BM.....	12
3.4 Proses Pencarian Matching Biodata dengan Regex.....	13
3.5 Fitur Fungsional dan Arsitektur Aplikasi yang Dibangun.....	14
3.3.1 Database.....	14
3.3.3 AES Encryption.....	15
3.6 Ilustrasi Kasus.....	21
<b>BAB IV.....</b>	<b>23</b>
<b>IMPLEMENTASI DAN PENGUJIAN.....</b>	<b>23</b>
4.1 Spesifikasi Teknis Program.....	23
4.1.1 Kelas Ascii Converter.....	23

4.1.2 Kelas TextProcessing.....	24
4.1.3 Kelas CustomAes.....	24
4.1.4 Kelas Database.....	25
4.1.5 Kelas BM.....	25
4.1.6 Kelas KMP.....	25
4.1.7 Kelas Levenshtein.....	26
4.1.8 Kelas MainWindow.....	26
4.2 Hasil Pengujian.....	27
4.3 Analisis Hasil Pengujian.....	31
<b>BAB V.....</b>	<b>33</b>
<b>KESIMPULAN DAN SARAN.....</b>	<b>33</b>
5.1 Kesimpulan.....	33
5.2 Saran.....	33
<b>DAFTAR PUSTAKA.....</b>	<b>34</b>
<b>LAMPIRAN.....</b>	<b>35</b>

# BAB I

## DESKRIPSI TUGAS

### 1.1 Deskripsi Tugas

Keamanan data dan akses menjadi semakin penting. Salah satu metode autentikasi yang telah berkembang adalah penggunaan biometrik, terutama identifikasi sidik jari, yang menawarkan keunikan yang praktis dan sulit untuk dipalsukan. Teknologi ini memanfaatkan algoritma *pattern matching* seperti Boyer-Moore dan Knuth-Morris-Pratt untuk mencocokkan sidik jari yang terdeteksi dengan sidik jari yang sudah terdaftar dalam database. Dengan demikian, sistem dapat mengidentifikasi seseorang dengan cepat dan akurat meskipun sidik jari yang di-scan tidak sempurna.



**Gambar 1.** Ilustrasi *fingerprint recognition* pada deteksi berbasis biometrik.

(Sumber: <https://www.aratek.co/news/unlocking-the-secrets-of-fingerprint-recognition>)

### 1.2 Spesifikasi

Pada tugas besar ini, kami diminta untuk mengembangkan sebuah sistem identifikasi biometrik yang menggunakan sidik jari sebagai kunci akses. Sistem ini akan mengkonversi gambar sidik jari menjadi data biner dan ASCII untuk memudahkan pencocokan pattern. Proses ini terbagi menjadi beberapa tahapan dimana setiap pixel dari citra sidik jari diambil dan dikonversi menjadi format biner dan selanjutnya ke ASCII, yang kemudian digunakan untuk pencocokan menggunakan algoritma pencocokan string. Selain itu, sistem juga dihadapkan pada tantangan untuk menangani data yang mungkin mengalami korupsi dalam

bentuk bahasa korup, yang sering ditemukan dalam data nama pada database. Sistem harus mampu menggunakan Regular Expression untuk mengkonversi bahasa korup ke format standar sebelum melakukan pencocokan nama menggunakan algoritma yang sama.

Program ini dirancang untuk berjalan pada aplikasi desktop yang dikembangkan dengan bahasa C#, di mana pengguna dapat meng-upload citra sidik jari dan memilih algoritma pencocokan yang diinginkan yaitu Boyer-Moore atau Knuth-Morris-Pratt. Interface program akan menyajikan informasi detail dari sidik jari yang paling mirip yang terdapat dalam database, termasuk persentase kemiripan dan waktu eksekusi program.

## BAB II

### LANDASAN TEORI

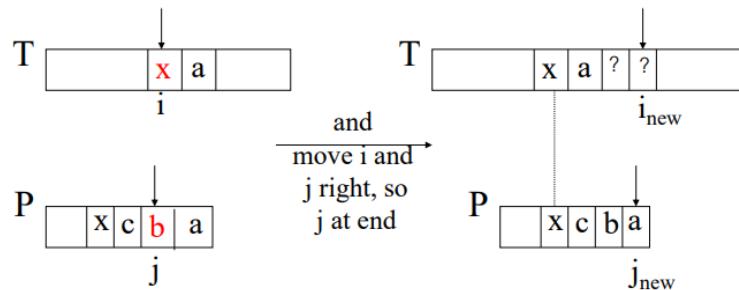
#### 2.1 Pattern Matching

*Pattern matching* adalah proses menemukan lokasi pertama di dalam sebuah teks T dengan panjang n karakter, di mana sebuah *pattern* P dengan panjang m karakter muncul sepenuhnya, dengan asumsi bahwa  $m <<< n$ .

#### 2.2 Algoritma Boyer-Moore

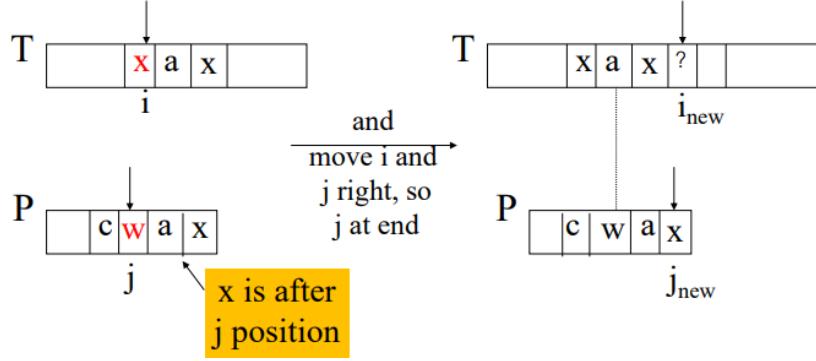
Algoritma pencocokan pola Boyer-Moore didasarkan pada dua teknik. Pertama, teknik cermin atau "*looking-glass technique*", yang mencari pola P dalam teks T dengan cara bergerak mundur melalui P, dimulai dari akhirnya. Kedua, teknik lompatan karakter atau "*character-jump technique*", yang digunakan ketika terjadi ketidakcocokan di  $T[i] = x$ , di mana karakter dalam pola  $P[j]$  tidak sama dengan  $T[i]$ . Terdapat tiga kemungkinan kasus sebagai berikut.

- Ketika pattern P mengandung x di suatu tempat, geser P ke kanan untuk menyejajarkan kemunculan terakhir x dalam P dengan T[i]



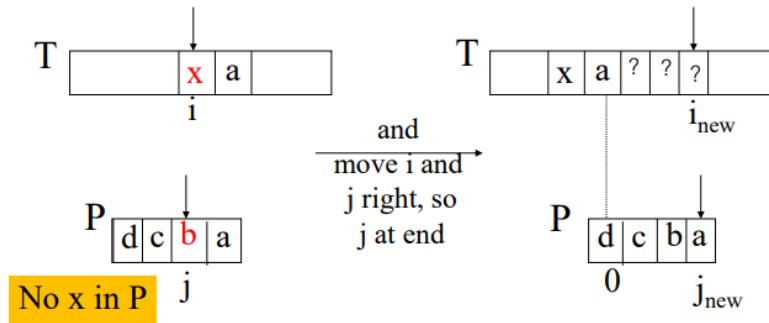
**Gambar 2.** Ilustrasi kasus 1 pencarian string dengan algoritma BM.

- Ketika pattern P mengandung x di suatu tempat dan pergeseran ke kanan ke kemunculan terakhir tidak mungkin, geser P sejauh 1 karakter ke  $T[i+1]$ .



**Gambar 3.** Ilustrasi kasus 2 pencarian string dengan algoritma BM.

- Jika keadaannya selain kedua kemungkinan sebelumnya, geser pattern P untuk menyajarkan  $P[0]$  dengan  $T[i+1]$ .



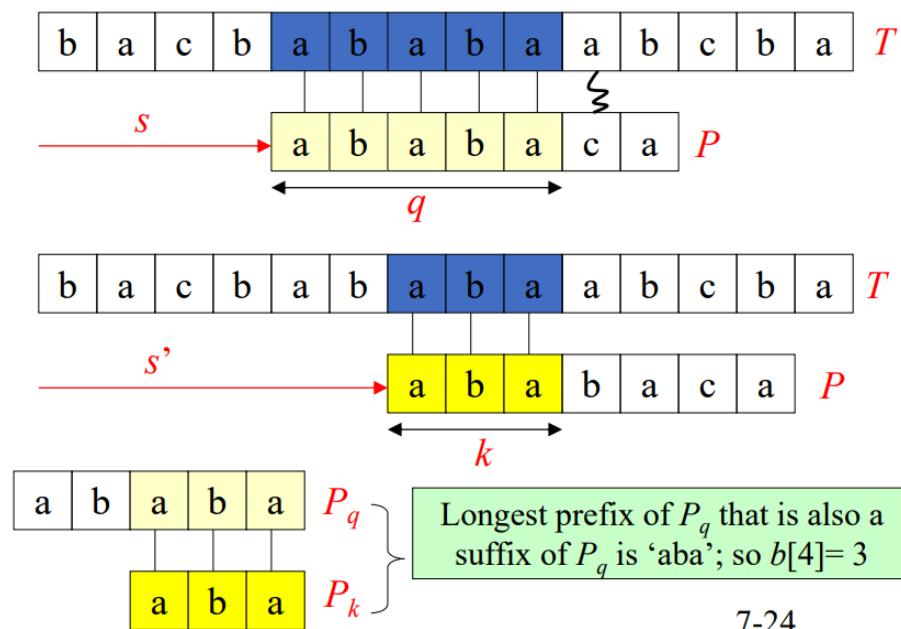
**Gambar 4.** Ilustrasi kasus 3 pencarian string dengan algoritma BM.

Fungsi untuk menghitung kemunculan terakhir dalam algoritma ini disebut *last occurrence function* (dinotasikan dengan  $L(x)$ ).  $L(x)$  didefinisikan sebagai index terbesar  $i$  sehingga  $P[i] == x$  atau  $-1$  jika indeks tersebut tidak ada.

### 2.3 Algoritma Knuth-Morris-Pratt

Algoritma Knuth-Morris-Pratt (KMP) mencari *pattern* dalam teks dengan urutan dari kiri ke kanan (seperti algoritma *brute force*). Namun, algoritma ini menggeser *pattern* dengan lebih baik dibandingkan dengan algoritma *brute force*. Saat terjadi ketidakcocokan antara teks dan pattern  $P$  di posisi  $P[j]$  ( $T[i] \neq P[j]$ ), *pattern* tersebut digeser sejauh panjang *pattern* dikurangi panjang prefiks terpanjang dari  $P[0..j-1]$  yang juga merupakan sufiks dari  $P[1..j-1]$ . Panjang prefiks ini disebut sebagai fungsi pinggiran KMP atau *KMP Border Function*, yang dinotasikan dengan  $b(k)$  dimana  $k = j-1$ . Algoritma ini memiliki efisiensi waktu  $O(m)$  untuk

menghitung fungsi pinggiran dan  $O(n)$  untuk proses pencarian string, sehingga total kompleksitas waktu algoritma KMP adalah  $O(m + n)$ .



**Gambar 5.** Ilustrasi pencarian string dengan algoritma KMP.

Kelebihan algoritma KMP adalah tidak memerlukan pergerakan mundur dalam teks masukan  $T$ . Hal ini sangat menguntungkan saat mengolah file berukuran besar yang dibaca dari perangkat eksternal atau melalui aliran jaringan. Namun, kekurangannya terlihat ketika ukuran alfabet meningkat, yang meningkatkan peluang terjadinya ketidakcocokan. Ketidakcocokan ini lebih sering terjadi di awal *pattern*, sementara KMP bekerja lebih efisien jika ketidakcocokan terjadi di bagian akhir *pattern*.

## 2.4 Regular Expression

Regex adalah sebuah filter yang memaparkan pola tertentu yang harus dipenuhi oleh serangkaian string. Dengan kata lain, regex dapat menerima sejumlah string tertentu sementara menolak yang lainnya. Penggunaan regex dalam satu baris dapat menggantikan banyak baris kode pemrograman dengan mudah, membuatnya menjadi alat yang sangat kuat dalam memanipulasi string. Pada program ini, regex digunakan untuk mengkonversi bahasa korup. Sebuah regex dapat terdiri dari beberapa notasi umum sebagai berikut.

**Tabel 1.** Notasi Umum *Regular Expression*

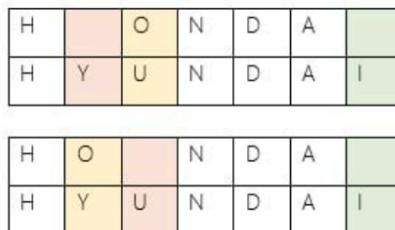
Notasi	Arti
+	Operator yang menunjukkan bahwa karakter sebelumnya harus muncul setidaknya satu kali atau lebih.
*	Operator yang menunjukkan bahwa karakter sebelumnya bisa muncul nol kali atau lebih.
?	Operator yang menunjukkan bahwa karakter sebelumnya bisa muncul nol atau satu kali.
	Operator OR yang memungkinkan pencocokan dengan salah satu dari beberapa pola.
.	Semua karakter kecuali <i>newline</i> .
^	Mencocokkan awal dari string atau baris.
\$	Mencocokkan akhir dari string atau baris.
\d, \w, \s	Shorthand character classes <ul style="list-style-type: none"> <li>- \d cocok dengan angka digit (0-9).</li> <li>- \w cocok dengan karakter word (angka, huruf, dan underscore).</li> <li>- \s cocok dengan whitespace (spasi, tab, baris baru).</li> </ul>
\D, \W, \S	Kebalikan dari shorthand character classes <ul style="list-style-type: none"> <li>- \D cocok dengan non-digit.</li> <li>- \W cocok dengan non-word characters.</li> <li>- \S cocok dengan non-whitespace.</li> </ul>
[a-z]	Character set yang cocok dengan karakter apapun dari 'a' hingga 'z'.

[abc]	Character set yang cocok dengan 'a', 'b', atau 'c'.
\., \\\, \*	Karakter titik, backslash, dan bintang.

## 2.5 Algoritma Levenshtein Distance

*Levenshtein Distance* adalah ukuran kemiripan antara dua string, yang memperhitungkan jumlah operasi penyisipan, penghapusan, dan substitusi yang diperlukan untuk mengubah satu string ke string lainnya. *Levenshtein Distance* yang semakin sedikit menandakan kemiripan antara kedua string yang semakin tinggi. Sebaliknya, *Levenshtein Distance* yang semakin tinggi menandakan kemiripan antara kedua string yang semakin rendah.

*Levenshtein distance between "HONDA" and "HYUNDAI" is 3.*



**Gambar 6.** Ilustrasi algoritma *Levenshtein Distance*.

## 2.6 Penjelasan Singkat Aplikasi yang Dibuat

Aplikasi yang dibangun dibuat dengan menggunakan bahasa pemrograman C# untuk backend maupun GUI. Backend dari program ini meliputi algoritma BM, KMP, dan *regular expression*. Basis data yang dikembangkan menggunakan MySQL dengan informasi yang disimpan adalah nama korup, nama asli, tempat lahir, tanggal lahir, jenis kelamin, alamat, status perkawinan, NIK, agama, kewarganegaraan, golongan darah, pekerjaan, path sidik jari, dan ASCII sidik jari.

Aplikasi memiliki hanya satu halaman dimana user akan mengupload gambar sidik jari yang ingin dicari informasinya lalu sidik jari yang cocok akan muncul beserta informasi yang sesuai. Pencocokan ini akan memanfaatkan tabel di database. Terdapat pula tombol switch yang memberikan pilihan pada user untuk melakukan pencarian dengan algoritma BM atau

KMP. Selain hasil informasi yang sesuai, ditampilkan pula persen kesesuaian sidik jari yang diunggah user dengan sidik jari yang paling cocok dari database.

## **BAB III**

### **ANALISIS PEMECAHAN MASALAH**

#### **3.1 Langkah Pemecahan Masalah secara Umum**

Permasalahan yang akan diselesaikan pada Tugas Besar kali ini adalah pencarian sidik jari dari database yang cocok dengan sidik jari yang diunggah oleh user. Urutan penyelesaian masalah yang dilakukan adalah sebagai berikut.

1. Program menerima gambar sidik jari yang diunggah oleh user.
2. Program melakukan evaluasi pattern dan mengkonversi sidik jari yang diunggah ke nilai ASCII yang sesuai.
3. Program mencari nilai ASCII pada database (tepatnya tabel sidik jari) yang cocok dengan nilai ASCII sidik jari yang diunggah.
4. Pada tabel sidik jari, akan mendapatkan nama korup yang sesuai dengan sidik jari yang ditemukan.
5. Program akan mencari ASCII dari pattern pada ASCII pada setiap gambar dengan menggunakan algoritma KMP dan BM, serta evaluasi dengan Levenstein.
6. Program akan mencari nama asli dari nama korup tersebut dengan menggunakan regex.
7. Nama asli yang dihasilkan akan dibandingkan dengan nama asli yang ada pada tabel biodata.
8. Program akan menampilkan biodata yang sesuai pada aplikasi beserta gambar sidik jarinya.

#### **3.2 Proses Pemilihan ASCII dan Evaluasi Pattern**

Dalam menentukan area ASCII yang akan diambil, ada beberapa keputusan yang harus dipertimbangkan dalam proses pemotongan pada setiap gambar:

1. Penanganan Padding

Padding dari suatu gambar akan dihilangkan dengan kriteria sebagai berikut, setidaknya harus terdapat 40 pixel dengan nilai keabuan (gray value) di bawah 128 dalam setiap baris untuk memastikan penghilangan padding putih lainnya. Pada kolom, hanya minimal 1 pixel dengan syarat yang sama. Penghapusan padding penting untuk

menghindari gangguan yang dapat mempengaruhi analisis sidik jari. Dengan mengambil area yang memenuhi kriteria nilai keabuan, area yang dipertahankan dalam proses pemotongan akan menjadi representasi yang lebih akurat dari gambar asli.

## 2. Konsistensi Lebar

Proses pemotongan juga memperhatikan konsistensi lebar pada setiap baris dengan memastikan lebarnya adalah kelipatan 8. Hal ini bertujuan untuk meningkatkan fleksibilitas dalam konversi ke format ASCII sehingga hasil konversi lebih terstruktur dan dapat memberikan representasi yang lebih baik dari gambar asli.

## 3. Pemilihan Pola

Dalam membandingkan pola dengan data ASCII input dari pola pada beberapa gambar, pola yang diambil adalah yang terletak di bagian tengah sidik jari. Pengamatan menunjukkan bahwa pengambilan pola dari bagian tengah menghasilkan hasil yang lebih konsisten daripada pola yang diambil dari daerah dengan konsentrasi tertinggi ataupun daerah lain. Proses pengambilan pola dilakukan dengan mengonversi 1 baris data berisi 80 piksel menjadi ASCII dengan interval kelipatan 8 untuk membentuk satu karakter untuk meningkatkan pertimbangan pattern.

### 3.3 Proses Penyelesaian Solusi dengan Algoritma KMP dan BM

Algoritma KMP dan BM pada program ini digunakan untuk mencocokkan ASCII sidik jari hasil unggahan dari user dengan kumpulan ASCII sidik jari yang terdaftar dalam database. Hal yang pertama dilakukan adalah mengonversi biner sidik jari ke ASCII.

Melanjutkan penjelasan pada poin 3.2, cara perbandingan kedua ASCII sidik jari ini dilakukan dengan beberapa tahap. Setelah pola ditentukan, ditemukan ASCII yang sesuai dengan sidik jari tersebut. Pencocokan antar ASCII sidik jari menggunakan dua algoritma yaitu Boyer-Moore atau Knuth-Morris-Pratt. Bila menggunakan algoritma KMP, karakter ASCII paling kiri sidik jari pertama akan dibandingkan dengan karakter ASCII paling kiri sidik jari kedua. Bila terjadi mismatch pada suatu karakter, peloncatan indeks akan mengikuti border function ASCII sidik jari pertama (sidik jari yang diupload).

Bila menggunakan algoritma BM, karakter ASCII paling kanan sidik jari pertama akan dibandingkan dengan karakter ASCII paling kiri sidik jari kedua. Bila terjadi mismatch pada suatu karakter, peloncatan indeks akan mengikuti last occurrence ASCII sidik jari pertama

(sidik jari yang diupload). Mismatch pada algoritma BM dibagi menjadi 3 kasus, jika last occurrence karakter mismatch pada sidik jari kedua lebih kiri daripada indeks sebenarnya karakter tersebut, maka indeks barunya adalah indeks sekarang + ukuran sidik jari pertama - (last occurrence + 1). Jika last occurrence karakter mismatch pada sidik jari kedua lebih kanan daripada indeks sebenarnya karakter tersebut, maka indeks barunya adalah indeks sidik jari kedua sekarang + ukuran sidik jari pertama - indeks karakter pada sidik jari pertama. Kasus terakhir, jika karakter yang mismatch tidak terdapat pada sidik jari pertama, maka indeks barunya akan menjadi indeks sekarang + ukuran sidik jari pertama.

Berdasarkan kedua algoritma tersebut, bila ditemukan string ASCII yang benar-benar sama, maka akan dikeluarkan persentase 100% antara kedua string tersebut. Bila tidak ada string yang benar-benar sama, maka akan dihitung persentasenya dengan menggunakan algoritma Levenshtein.

### **3.4 Proses Pencarian Matching Biodata dengan Regex**

Regular Expression digunakan untuk melakukan solving pada data name corrupt pada data table biodata, nama yang corrupt ini tidak bisa menjadi foreign key ke data table sehingga untuk menghubungkan entitas dari pemilik suatu sidik jari. Langkah-langkah yang diterapkan adalah sebagai berikut:

1. Kedua kalimat diubah menjadi lowercase dan dibersihkan dari karakter yang bukan huruf atau angka untuk memastikan huruf dan angka yang menjadi fokus perbandingan.
2. Kedua kalimat dijalani melalui proses normalisasi alay yang mengkonversi angka dalam kalimat menjadi karakter sesuai dengan aturan normalisasi alay yang telah ditetapkan menggunakan regex.
3. Kedua kalimat dipecah menjadi array of kata-kata, sehingga kita dapat mengiterasi melalui setiap kata dalam kalimat.
4. Untuk setiap kata dalam kalimat, dilakukan perbandingan sebagai berikut:
  - a. Menghilangkan huruf vokal dari kata dengan menggunakan regex.
  - b. Membandingkan apakah hasil huruf konsonan dari kedua kata tersebut sama. Jika tidak sama, artinya kata-kata tersebut tidak cocok.
  - c. Jika huruf konsonan sama, Selanjutnya perlu dilakukan pengecekan huruf vokal, dibuat sebuah pola (pattern) berdasarkan huruf vokal yang lebih banyak dari kedua

- kata tersebut. Pola ini akan mencocokkan huruf-huruf vokal dari kata yang lebih sedikit huruf vokalnya.
- d. Selanjutnya, setiap huruf vokal dari kata yang lebih sedikit huruf vokalnya dibandingkan dengan pola tersebut. Jika ada huruf konsonan yang tidak cocok dengan pola, artinya kata-kata tersebut tidak cocok.

### **3.5 Fitur Fungsional dan Arsitektur Aplikasi yang Dibangun**

#### **3.3.1 Database**

Kami memilih menggunakan MySQL sebagai DBMS karena MySQL adalah salah satu sistem manajemen basis data (DBMS) yang sangat populer dan sering digunakan dalam pengembangan aplikasi. Dalam konteks aplikasi .NET, penghubungan dengan MySQL dilakukan melalui MySQL Connector yang merupakan komponen khusus untuk menghubungkan aplikasi .NET dengan server MySQL. MySQL Connector menyediakan kelas-kelas dan metode-metode yang memungkinkan aplikasi .NET untuk berkomunikasi dengan database MySQL, seperti melakukan kueri (query) data, mengeksekusi perintah SQL, dan lain sebagainya.

Kami juga menggunakan library Bogus dalam rangka untuk generate dummy data yang lebih realistik dan sesuai dengan kebutuhan aplikasi atau pengujian yang dilakukan.

#### **3.3.2 GUI**

Dalam aplikasi ini, kami menggunakan Windows Presentation Foundation (WPF) sebagai framework untuk membangun antarmuka pengguna grafis (GUI). WPF memungkinkan pembuatan antarmuka yang kaya dan interaktif dengan menggunakan berbagai kontrol dan layout yang tersedia

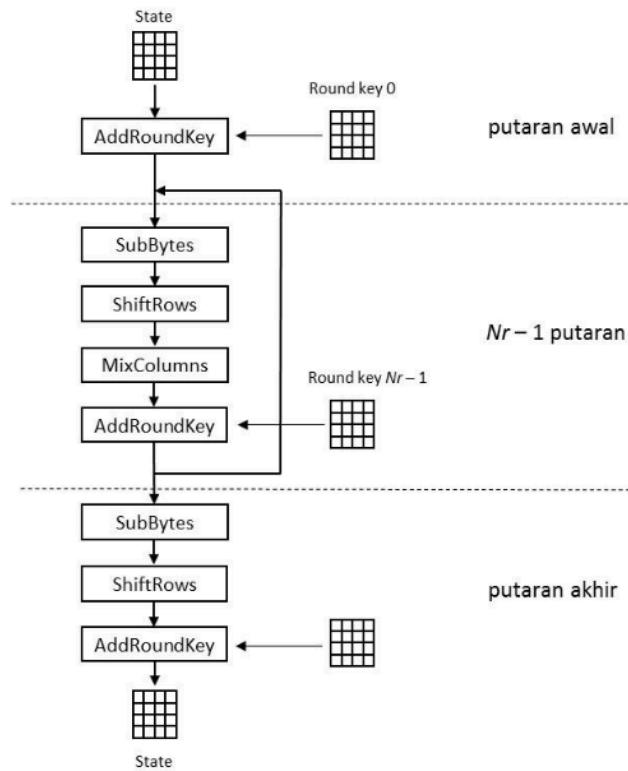
Beberapa komponen utama yang kami gunakan dalam antarmuka ini meliputi:

- Grid dan StackPanel: Grid mengatur tata letak utama dengan baris dan kolom, sementara StackPanel menyusun elemen-elemen secara vertikal atau horizontal.
- Custom Button: Tombol-tombol diberi gaya khusus untuk tampilan modern dan responsif, seperti tombol fullscreen dan tombol close.

- Custom Toggle Button: ToggleButton diberi gaya khusus untuk memungkinkan pengguna mengubah status antara aktif dan non-aktif dengan umpan balik visual yang jelas.
- TextBlock: Menampilkan teks seperti judul aplikasi dengan gaya yang konsisten dan menarik.
- Border: Memberikan bingkai dan padding pada elemen-elemen untuk meningkatkan estetika dan keterbacaan.
- ScrollViewer: Membuat area yang dapat digulir untuk melihat konten yang lebih besar dari ukuran tampilan.
- Toggle Button : Membuat interactivity terhadap data yang berbentuk boolean

### 3.3.3 AES Encryption

Advanced Encryption Standard (AES) atau Rijndael Algorithm adalah sebuah blok ciphertext simetrik yang merupakan lanjutan dari Data Encryption Standard (DES) yang masa berlakunya dianggap telah usai karena faktor keamanan. Dalam algoritma kriptografi AES-128, 1 blok plainteks berukuran 128 bit terlebih dahulu dikonversi menjadi matriks heksadesimal berukuran  $4 \times 4$  yang disebut state. Setiap elemen state berukuran 1 byte.

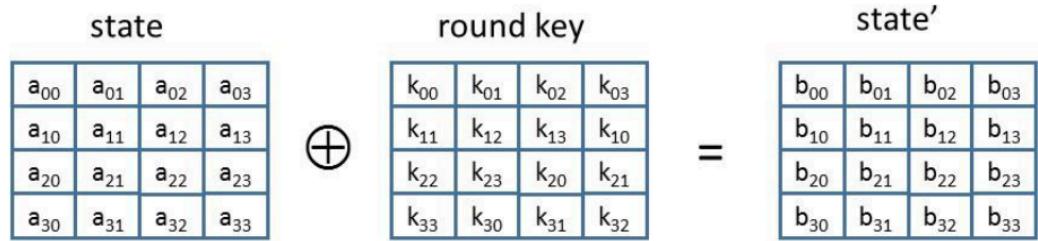


**Gambar 7.** Ilustrasi AES Encryption

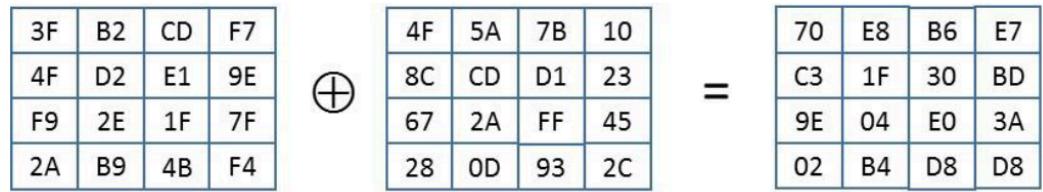
Algoritma Rijndael yang beroperasi pada blok 128-bit dengan kunci 128-bit beroperasi sebagai berikut (di luar proses pembangkitan round key):

1. AddRoundKey

Melakukan XOR antara state awal (plainteks) dengan cipher key. Tahap ini disebut juga initial round.



Contoh:



**Gambar 6.** Ilustrasi AES Encryption

2. Putaran sebanyak  $Nr - 1$  kali. Proses yang dilakukan pada setiap putaran adalah:

a. SubBytes

Substitusi byte dengan menggunakan tabel substitusi (S-box).

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Contoh:

state				state'			
23 A2 BC 4A				26 3A 65 D6			
D4	03	97	F3	48	7B	88	0D
16	48	CD	50	47	52	BD	53
FF	DA	10	64	16	57	CA	43

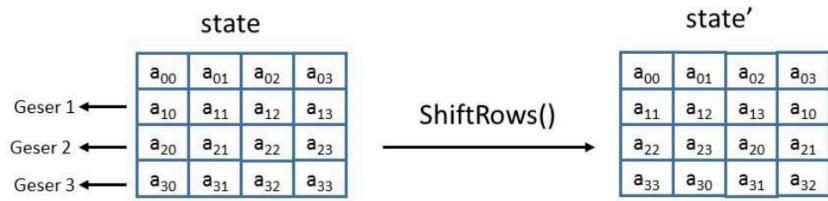
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Gambar 8. Tabel SBox

b. ShiftRows

Melakukan operasi permutasi dengan pergeseran secara wrapping (siklik) pada 3 baris terakhir array state. Jumlah pergeseran bergantung pada nilai

baris (r). Baris  $r = 1$  digeser sejauh 1 byte, baris  $r = 2$  sejauh 2 byte, dan baris  $r = 3$  sejauh 3 byte. Baris  $r = 0$  tidak digeser.



Contoh:

26	3A	65	D6
48	7B	88	0D
47	52	BD	53
16	57	CA	43

$\xrightarrow{\text{ShiftRows()}}$

26	3A	65	D6
7B	88	0D	48
BD	53	47	52
43	16	57	CA

**Gambar 9.** Tabel ShiftRows

### c. MixColumns

Mengacak data di masing-masing kolom array state. Pada kasus ini, pengacakan data dilakukan dengan menggunakan Galois Field sebagai berikut

Contoh:

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} 26 \\ 7B \\ BD \\ 43 \end{bmatrix} = \begin{bmatrix} 3F \\ 4F \\ F9 \\ 2A \end{bmatrix}$$

$$\begin{aligned} (02 \bullet 26) \oplus (03 \bullet 7B) \oplus (01 \bullet BD) \oplus (01 \bullet 43) &= 3F \\ (01 \bullet 26) \oplus (02 \bullet 7B) \oplus (03 \bullet BD) \oplus (01 \bullet 43) &= 4F \\ (01 \bullet 26) \oplus (01 \bullet 7B) \oplus (02 \bullet BD) \oplus (03 \bullet 43) &= F9 \\ (03 \bullet 26) \oplus (01 \bullet 7B) \oplus (01 \bullet BD) \oplus (02 \bullet 43) &= 2A \end{aligned}$$

**Gambar 10.** Tabel MixColumns

$$(02 \bullet 26) = (00000010) * (0010\ 0110)$$

$$\begin{aligned}
&= x * (x^5 + x^2 + x) \bmod (x^8 + x^4 + x^3 + x + 1) \\
&= (x^6 + x^3 + x^2) \bmod (x^8 + x^4 + x^3 + x + 1) \\
&= x^6 + x^3 + x^2 \\
&= (010001100) \\
&= 4C
\end{aligned}$$

$$\begin{aligned}
(03 \cdot 7B) &= (00000011) * (01111011) \\
&= (x+1) * (x^6 + x^5 + x^4 + x^3 + x+1) \bmod (x^8 + x^4 + x^3 + x + 1) \\
&= ((x^7 + x^6 + x^5 + x^4 + x^2 + x) + (x^6 + x^5 + x^4 + x^3 + x + 1)) \bmod \\
&\quad (x^8 + x^4 + x^3 + x + 1) \\
&= (x^7 + (1+1)x^6 + (1+1)x^5 + (1+1)x^4 + x^3 + x^2 + (1+1)x + 1) \bmod \\
&\quad (x^8 + x^4 + x^3 + x+1) \\
&= (x^7 + x^3 + x^2 + 1) \bmod (x^8 + x^4 + x^3 + x + 1) \\
&= (x^7 + x^3 + x^2 + 1) \\
&= (1000 1101) = 8D
\end{aligned}$$

$$(01 \cdot BD) = BD = 10111101$$

$$(01 \cdot 43) = 43 = 01000011$$

Selanjutnya, XOR-kan semua hasil antara tersebut:

$$(02 \cdot 26) = 4C = 0100 1100$$

$$(03 \cdot 7B) = 8D = 1000 1101$$

$$(01 \cdot BD) = BD = 1011 1101$$

$$(01 \cdot 43) = 43 = \underline{0100 0011}^\oplus$$

$$0011 1111 = 3F$$

$$\text{Jadi, } (02 \cdot 26)^\oplus (03 \cdot 7B)^\oplus (01 \cdot BD)^\oplus (01 \cdot 43) = 3F$$

Persamaan lainnya diselesaikan dengan cara yang sama.

#### d. AddRoundKey

Melakukan XOR antara state sekarang round key, sama seperti initial round.

### 3. Final round: proses untuk putaran terakhir:

#### a. SubBytes

- b. ShiftRows
- c. AddRoundKey

Di sisi lain, proses dekripsi menggunakan invers semua transformasi dasar pada algoritma AES kecuali addroundkey dengan urutan transformasi InvShiftRows, InvSubBytes, AddRoundKey, dan InvMixColumns.

- a. InvSubBytes

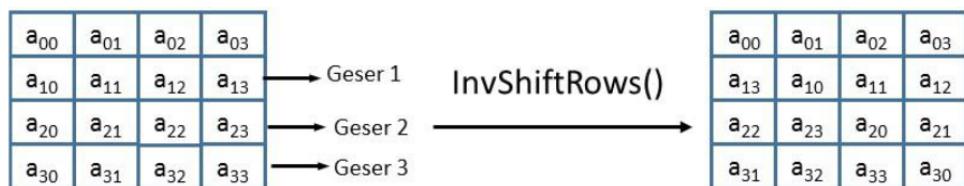
InvSubBytes() sama seperti di dalam SubBytes(), hanya saja S-box yang digunakan adalah inversi dari S-box

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	B	3C	83	53	99	61
F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

Gambar 11. Tabel InvSubBytes

- b. InvShiftRows

InvShiftRows sama seperti ShiftRows, tetapi melakukan pergeseran dalam arah berlawanan (ke kanan) untuk tiap-tiap baris pada tiga baris terakhir di dalam state



Gambar 12. Tabel InvShiftRows

c. InvMixColumns

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

**Gambar 13.** Tabel InvMixColumns

### 3.6 Ilustrasi Kasus

Diketahui bahwa string ASCII sidik jari yang ingin dicari adalah öläçýüsç, dan salah satu string ASCII sidik jari dari database adalah Đçýöläçýüs.

Pencocokan string ini secara KMP diilustrasikan sebagai berikut. String yang sesuai ditemukan setelah melakukan perbandingan karakter sebanyak 10 kali dengan pencarian dimulai dari karakter Đ.

ø	Ì	á	ç	ÿ	ü	s
0	0	0	1	0	0	0

**Gambar 14.** Tabel Border Function KMP

sidik jari database	Đ	ç	ÿ	ø	Ì	á	ç	ÿ	ü	s	ç
sidik jari yang diupload	ø	Ì	á	ç	ÿ	ü	s	ç			
	ø	Ì	á	ç	ÿ	ü	s	ç			
		ø	Ì	á	ç	ÿ	ü	s	ç		
			ø	Ì	á	ç	ÿ	ü	s	ç	

**Gambar 15.** Ilustrasi Kasus KMP

Pencocokan string ini secara BM diilustrasikan sebagai berikut. String yang sesuai ditemukan setelah melakukan perbandingan karakter sebanyak 9 kali dimulai dari karakter ç string sidik jari database pada indeks 10.

ø	Ì	á	ç	ÿ	ü	s
0	1	2	7	4	5	6

**Gambar 16.** Tabel Last Occurrence BM

sidik jari database	Đ	ç	ÿ	ø	Ì	á	ç	ÿ	ü	s	ç
sidik jari yang diupload	ø	Ì	á	ç	ÿ	ü	s	ç			
				ø	Ì	á	ç	ÿ	ü	s	ç

**Gambar 17.** Ilustrasi Kasus BM

## **BAB IV**

### **IMPLEMENTASI DAN PENGUJIAN**

#### **4.1 Spesifikasi Teknis Program**

Program diimplementasikan dengan object oriented berdasarkan C#, kami membagi modularitasnya menjadi beberapa class sebagai berikut

##### **4.1.1 Kelas Ascii Converter**

Kelas AsciiConverter bertindak sebagai mediator dari program yang bertanggung jawab atas proses konversi gambar ke teks ASCII. Dalam setiap langkahnya, kelas ini menggunakan beberapa method untuk preprocessing:

- ConvertBinaryToString: Method ini akan bertindak sebagai proses konversi. Saat diberikan data biner, metode ini memisahkan blok-blok 8 bit dan mengonversinya menjadi karakter ASCII sesuai dengan nilai binernya. Proses ini memastikan representasi teks yang tepat dari data biner yang diterima.
- CropImage: Sebelum konversi dimulai, metode ini menentukan area gambar yang paling relevan. Ini dilakukan dengan menganalisis tingkat keabuan piksel dalam gambar untuk mengidentifikasi batas-batas yang signifikan yang ditentukan sesuai penjelasan pada bab 3. Proses ini membantu menghilangkan bagian yang tidak relevan dari gambar, memastikan bahwa hanya informasi penting yang diambil untuk konversi.
- ImageToAscii: Metode ini mengambil jalur file gambar sebagai input. Pertama, gambar dibaca dan kemudian diproses untuk mengubahnya menjadi data biner berdasarkan teknik grayscale. Setiap piksel diubah menjadi bit biner, dengan nilai tinggi dan rendah direpresentasikan sebagai '1' dan '0' secara berurutan. Data biner yang dihasilkan kemudian dikonversi menjadi teks ASCII dengan memanfaatkan method lain yang ada.
- MidOneBitmap: Salah satu fitur kunci dari AsciiConverter adalah kemampuannya untuk menghasilkan representasi ASCII dari pattern yang ringkas dan informatif. Metode MidOneBitmap memfokuskan pada bagian tengah horizontal dari gambar dengan teknik seperti pada bab 3.

#### **4.1.2 Kelas TextProcessing**

Kelas ini berperan sebagai mediator foreign key antara tabel sidik\_jari dan biodata dengan melakukan serangkaian proses melalui regular expression (regex). Salah satu fungsi utama kelas ini adalah NumToChar, yang bertugas mentransformasi digit atau karakter alay dalam teks menjadi huruf yang sesuai merepresentasikan mereka. Selanjutnya, terdapat fungsi AlayNormalization yang melakukan normalisasi teks ke dalam huruf kecil, diikuti dengan konversi digit atau karakter alay menggunakan metode NumToChar. Selain itu, kelas ini juga memiliki fungsi RemoveVokal untuk mengeliminasi vokal ('a', 'i', 'u', 'e', 'o') dari teks, serta fungsi RemoveNonAlphabeticAndNumber yang bertujuan menghapus karakter-karakter non-huruf dan non-angka dari teks.

Kelas ini juga memiliki fungsi yang paling utama adalah CompareWord(string sentence, string sourceSentence) yang berfungsi membandingkan dua kalimat untuk menentukan tingkat kemiripan berdasarkan kriteria yang telah ditetapkan sebelumnya.

#### **4.1.3 Kelas CustomAes**

Kelas CustomAes merupakan implementasi algoritma Advanced Encryption Standard (AES) dengan panjang kunci 128 bit. Tujuan utama kelas ini adalah untuk menyediakan fungsionalitas enkripsi dan dekripsi data dengan menggunakan AES. Dalam kelas ini, terdapat properti Key dan IV yang digunakan untuk menyimpan kunci dan Initialization Vector yang diperlukan dalam proses enkripsi dan dekripsi.

Proses enkripsi dimulai dengan membagi data yang akan dienkripsi menjadi blok-blok 128 bit (16 byte), sesuai dengan ukuran blok standar AES. Setiap blok akan mengalami serangkaian operasi, termasuk substitusi byte (SubBytes), pergeseran baris (ShiftRows), pencampuran kolom (MixColumns), dan penambahan kunci putaran (AddRoundKey). Proses ini diulang sebanyak 10 putaran untuk AES-128, yang menghasilkan blok-blok data terenkripsi.

Sedangkan untuk dekripsi, prosesnya dilakukan secara invers dari enkripsi. Setiap blok data terenkripsi akan mengalami substitusi byte terbalik (InvSubBytes), pergeseran baris terbalik (InvShiftRows), pencampuran kolom terbalik

(InvMixColumns), dan penambahan kunci putaran terbalik (AddRoundKey). Proses ini juga diulang sebanyak 10 putaran untuk melakukan dekripsi yang benar.

Selain itu, kelas ini juga memiliki metode KeyExpansion yang bertanggung jawab untuk menghasilkan kunci tambahan yang diperlukan pada setiap putaran enkripsi dan dekripsi. Ekspansi kunci ini menggunakan algoritma khusus yang melibatkan substitusi kata (SubWord), rotasi kata (RotWord), dan operasi XOR dengan nilai konstan (Rcon).

#### 4.1.4 Kelas Database

Kelas Database ini digunakan untuk mengelola koneksi dan operasi database menggunakan MySql.Data.MySqlClient. Kelas ini memiliki atribut seperti server, user, databaseName, dan password yang digunakan untuk menginisialisasi koneksi ke database MySQL dengan aplikasi, setiap method yang ada pada class ini sebagai mediator query untuk komunikasi dengan database.

#### 4.1.5 Kelas BM

Kelas Boyer-Moore (BM) adalah implementasi dari algoritma pencarian pola dalam teks yang mengadopsi strategi pencarian last occurrence (kemunculan terakhir). Kelas BM memanfaatkan tabel last occurrence untuk setiap karakter dalam pola yang akan dicocokkan dengan teks. Dengan demikian, ketika terjadi ketidakcocokan antara karakter pada posisi tertentu dalam pola dan teks, BM akan menggunakan informasi terakhir munculnya karakter tersebut dalam pola untuk menggeser pola ke posisi yang lebih dekat dengan teks. Ini mengurangi jumlah perbandingan yang harus dilakukan, membuat pencarian menjadi lebih efisien.

Metode utama dalam kelas BM adalah Match, yang menerima pola dan teks sebagai argumen. Metode ini mengembalikan indeks pertama di mana pola ditemukan dalam teks, atau -1 jika tidak ada kecocokan yang ditemukan. Dengan menggunakan strategi last occurrence, BM mampu melakukan pencocokan dengan cepat terutama pada kasus di mana pola berukuran besar dan memiliki banyak karakter unik.

#### 4.1.6 Kelas KMP

Kelas KMP adalah implementasi dari algoritma KMP menggunakan tabel border untuk mengidentifikasi pola yang sama berulang dalam pola itu sendiri, sehingga

menghindari perbandingan yang tidak perlu saat mencocokkan pola dengan teks. Hal ini membuat KMP efisien terutama pada kasus di mana pola memiliki struktur yang berulang atau terdapat karakter yang sering muncul.

Kelas KMP memiliki metode Match yang menerima pola dan teks sebagai input. Metode ini mengembalikan indeks pertama di mana pola ditemukan dalam teks, atau -1 jika tidak ada kecocokan yang ditemukan. Dengan memanfaatkan tabel border, KMP dapat mempercepat pencarian pola dalam teks dengan mengurangi jumlah langkah-langkah yang harus dilakukan.

#### 4.1.7 Kelas Levenshtein

Kelas Levenshtein merupakan implementasi algoritma Levenshtein Distance, yang mengukur jarak atau perbedaan antara dua string. Dalam konteks pencocokan string, Levenshtein Distance sering digunakan untuk mengukur seberapa mirip atau berbedanya dua string. Semakin kecil nilai Levenshtein Distance antara dua string, semakin mirip kedua string tersebut.

Kelas Levenshtein memiliki beberapa metode penting, seperti LevenshteinDistance yang menghitung jumlah operasi yang diperlukan untuk mengubah satu string menjadi string lainnya. Terdapat juga metode CalculateSimilarityPercentage yang menghitung persentase kemiripan antara dua string berdasarkan jarak Levenshtein. Metode MatchWithLevenshtein mengkombinasikan algoritma BM, KMP, dan Levenshtein untuk menemukan kemiripan teks dengan persentase kemiripan yang ditentukan.

#### 4.1.8 Kelas MainWindow

Kelas MainWindow memiliki peran vital dalam aplikasi, berfungsi sebagai penghubung antara logika bisnis yang kompleks dengan antarmuka pengguna yang interaktif. Kelas ini menginisialisasi koneksi database dengan mengizinkan pengguna untuk menyesuaikan detail koneksi atau menggunakan detail default. Setelah koneksi terbentuk, kelas ini memuat data biodata dan sidik jari dari database untuk digunakan dalam proses selanjutnya.

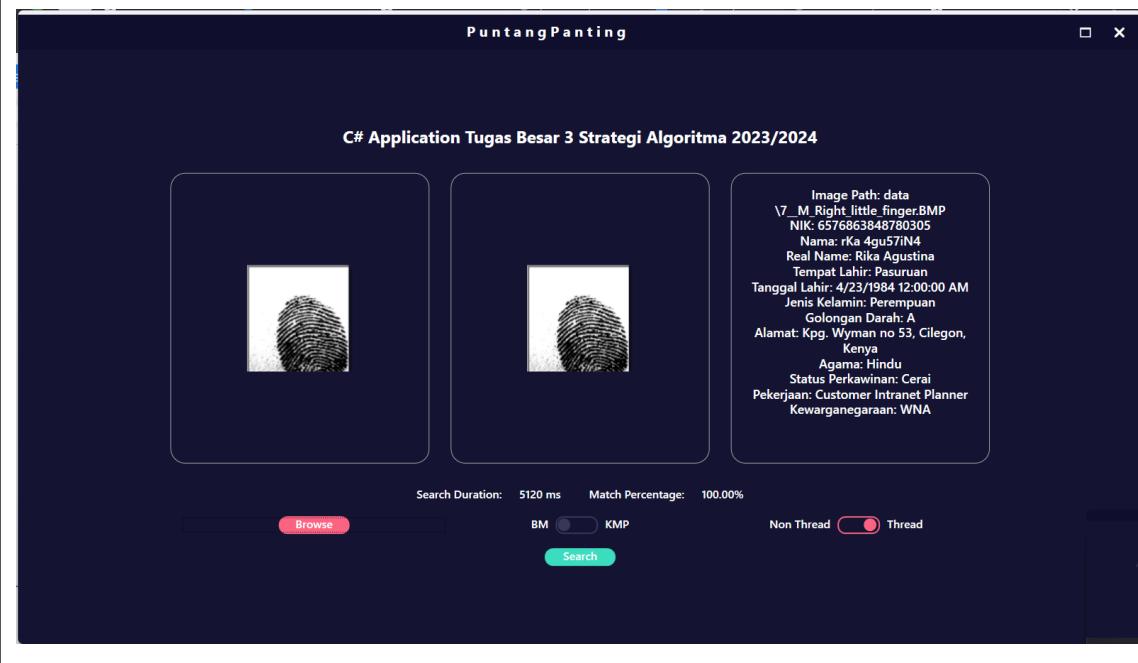
Interaksi dengan antarmuka pengguna terjadi melalui berbagai fungsi tombol, seperti mengunggah gambar, menutup aplikasi, atau mengubah tampilan layar. Saat gambar diunggah, kelas ini mengonversinya menjadi pola untuk pencocokan dengan data sidik jari. Kelas ini juga menggunakan Task dan Async Await memastikan pemrosesan data dilakukan secara asinkron untuk menjaga responsivitas antarmuka pengguna.

Hasil dari proses pencocokan, termasuk gambar sidik jari yang cocok dan biodata terkait, ditampilkan secara visual pada antarmuka pengguna. Ini memungkinkan pengguna untuk melihat hasil pencocokan dengan detail yang diperlukan.

## 4.2 Hasil Pengujian

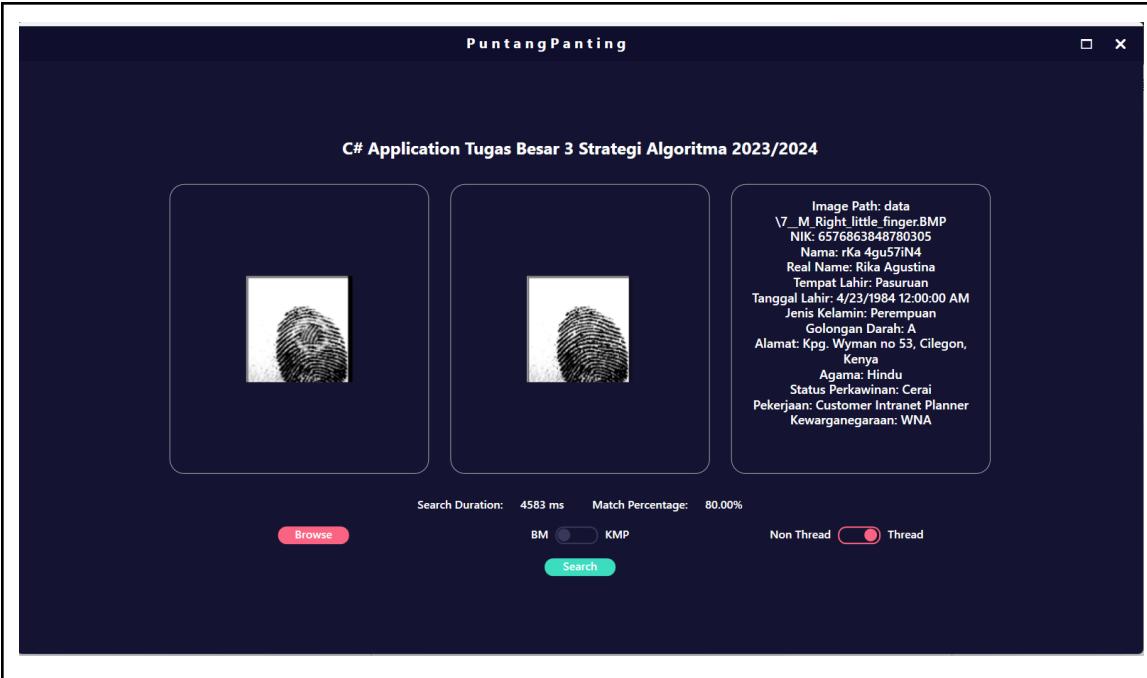
Real

7\_M\_Right\_little\_finger\_Obl



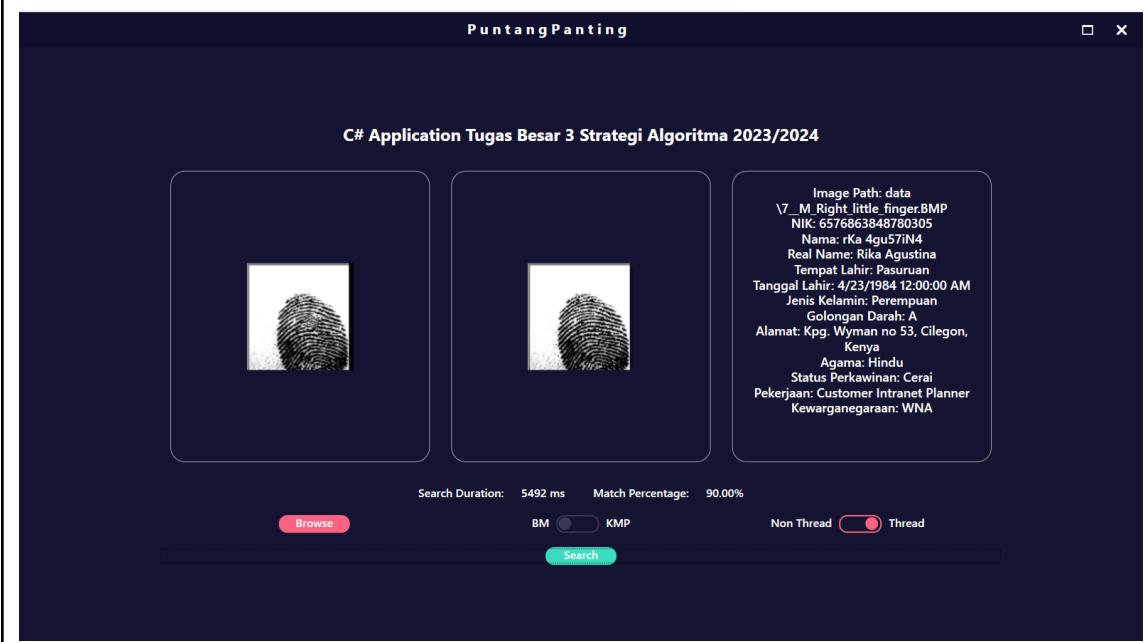
Altered Easy

7\_M\_Right\_little\_finger\_Obl



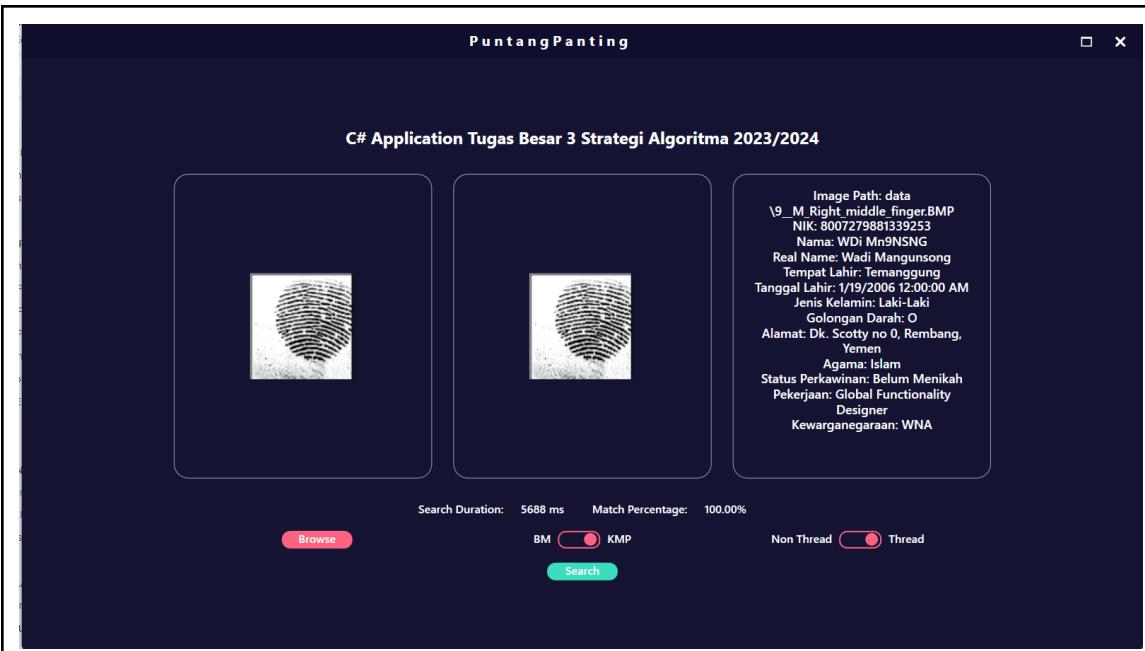
Altered Easy

7\_M\_Right\_little\_finger\_Zcut



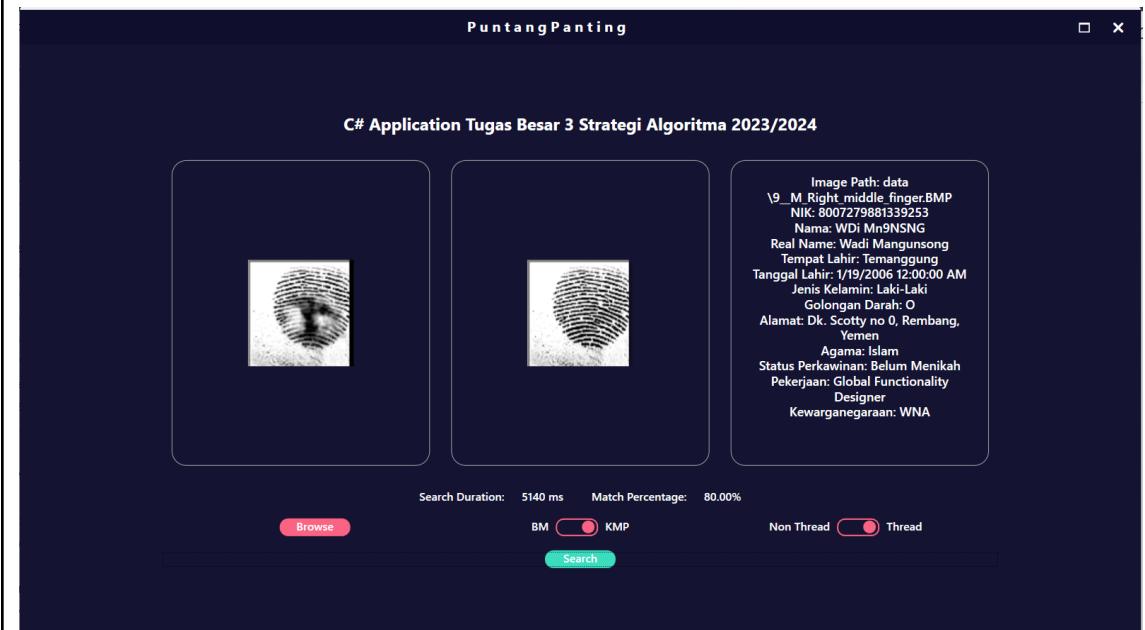
Real

9\_M\_Right\_middle\_finger



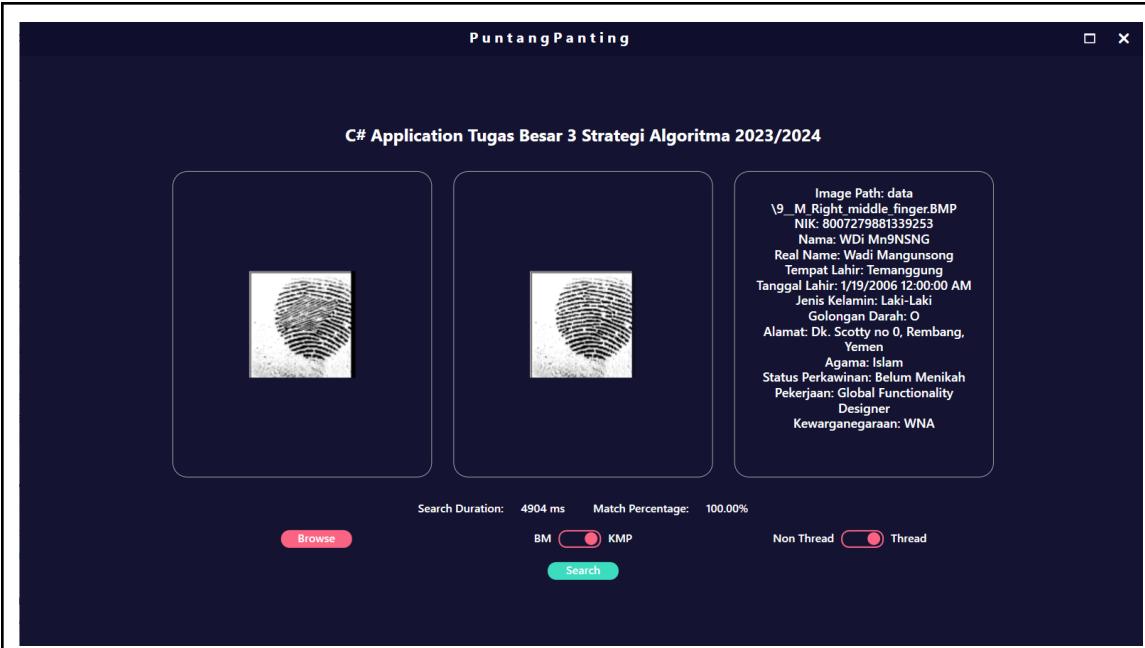
Altered Medium

9\_M\_Right\_middle\_finger\_Obl



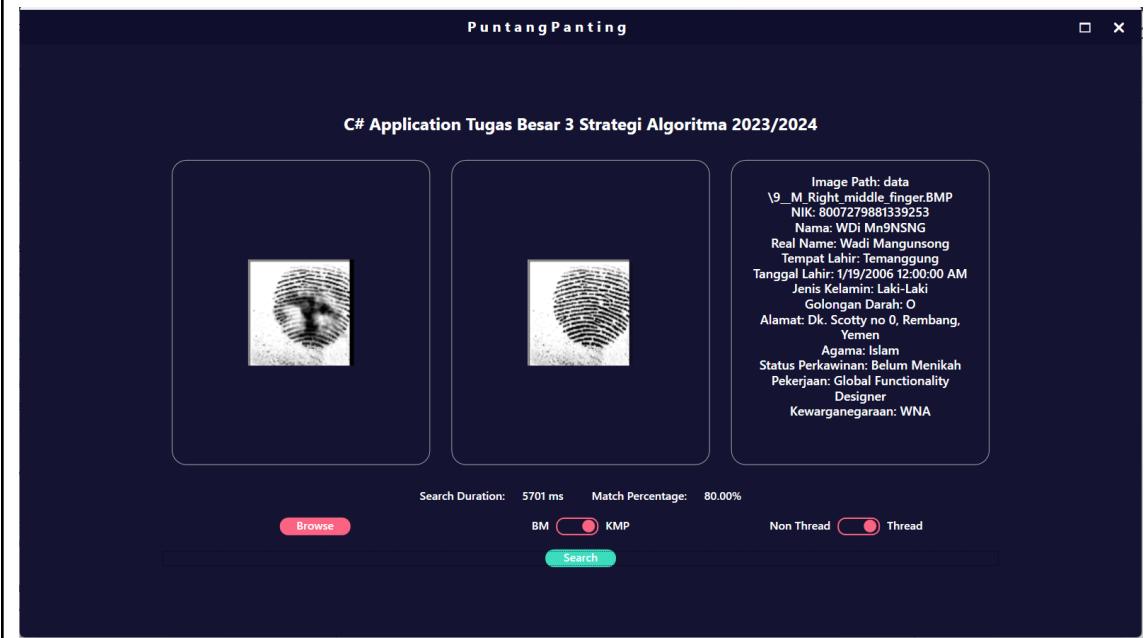
Altered Medium

9\_M\_Right\_middle\_finger\_Cut



Altered Hard

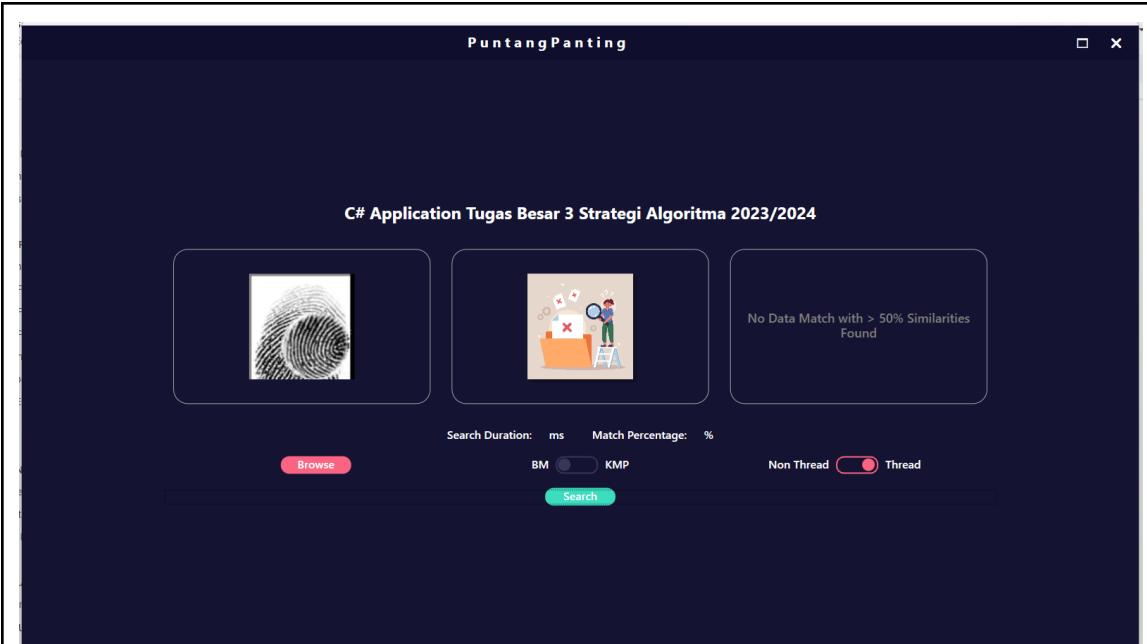
9\_M\_Right\_middle\_finger\_Obl



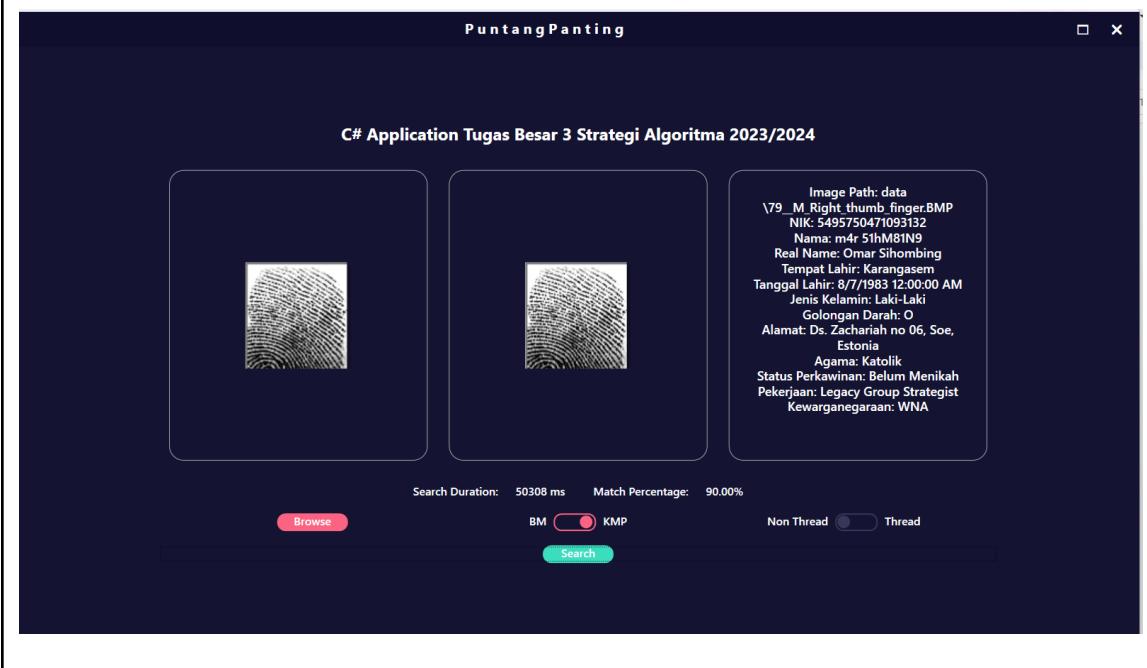
If the data percentage under 50%

Altered Hard

5\_M\_Left\_little\_finger\_CR



Perbandingan hasil kecepatan KMP dan BM



### 4.3 Analisis Hasil Pengujian

Dari hasil pengujian, terbukti bahwa algoritma pencocokan string Boyer-Moore (BM) secara signifikan lebih cepat daripada algoritma Knuth-Morris-Pratt (KMP). Perbedaan

ini sangat mencolok dan dapat dijelaskan karena metode yang digunakan dalam algoritma BM memungkinkan untuk menghindari sejumlah besar perbandingan yang tidak perlu.

Algoritma BM mengoptimalkan pencocokan string dengan memperhitungkan informasi dari pola yang dicari, seperti menggunakan tabel penyesuaian (bad character shift) dan tabel geser baik karakter buruk (bad character shift) maupun geser kata (good suffix shift). Informasi ini memungkinkan algoritma BM untuk lebih efisien dalam menghindari langkah-langkah yang tidak perlu dalam pencocokan pola.

Sebagai contoh, ketika sebuah karakter tidak cocok dengan pola yang dicari dalam algoritma BM, langkah-langkahnya dapat langsung memindahkan pola ke posisi yang lebih jauh dengan memanfaatkan tabel penyesuaian karakter buruk. Sementara itu, algoritma KMP memerlukan langkah-langkah tambahan untuk mengatasi kecocokan karakter yang tidak sesuai, sehingga memerlukan lebih banyak perbandingan dan langkah-langkah tambahan.

Pada test case lain juga dapat dilihat bahwa algoritma sudah dapat menangani jika terjadi ketidaksesuaian hasil data secara 100%, yang diatasi dengan algoritma Levenshtein. Terdapat beberapa kasus yang tidak dapat dihandle oleh threshold yang telah ditetapkan, yaitu kasus dimana similarity antara data asli dan input data berada di bawah 50%. Pada kasus ini, data altered hard tersebut telah mengalami rotasi fingerprint dan pemisahan yang signifikan, sehingga perbedaan antara data asli dan data yang diubah menjadi lebih sulit dideteksi.

Meskipun demikian, selain kasus data altered hard yang berjenis CR (Central Rotation) tersebut, algoritma masih dapat menyelesaikan dan menemukan similarity pada ketidaksesuaian di atas threshold yang telah ditetapkan pada beberapa test case lain. Hal ini menunjukkan bahwa algoritma telah berhasil mengatasi sebagian besar kasus ketidaksesuaian, namun masih membutuhkan peningkatan khususnya dalam menangani kasus yang memiliki perubahan signifikan seperti rotasi fingerprint dan padding yang memisahkan data secara signifikan.

## **BAB V**

### **KESIMPULAN DAN SARAN**

#### **5.1 Kesimpulan**

Melalui Tugas Besar III Strategi Algoritma ini, kami diminta untuk membuat pencocokan string antara sidik jari yang diunggah dengan sidik jari yang terdaftar di database. Kami membuat sebuah aplikasi berbasis bahasa pemrograman C# dimana user dapat mengunggah sidik jari yang ingin diketahui informasinya lalu aplikasi ini akan mengeluarkan sidik jari yang paling cocok berdasarkan pencocokan string menggunakan algoritma Boyer Moore atau Knuth Morris Pratt. Melalui pengujian yang kami lakukan, algoritma Boyer-Moore (BM) terbukti lebih efisien dan cepat dibandingkan dengan Knuth-Morris-Pratt (KMP) karena kemampuannya mengurangi perbandingan yang tidak perlu dengan teknik heuristik alias berdasarkan last occurrence masing-masing karakter pada sidik jari yang dicari. Meskipun BM lebih efektif dalam banyak skenario, kedua algoritma memerlukan pengembangan lebih lanjut untuk mengatasi kasus data yang telah mengalami perubahan signifikan seperti pemisahan yang besar.

#### **5.2 Saran**

Saran bagi kelompok PuntangPanting untuk kedepannya adalah sebagai berikut,

1. Mengoptimalkan struktur pembagian tugas serta meningkatkan kerjasama tim untuk mengoptimalkan hasil program.
2. Perluasan pengembangan algoritma untuk meningkatkan efisiensi dalam mencapai goal program.
3. Peningkatan dalam penulisan komentar dan dokumentasi untuk mempermudah kerja sama tim dan *maintenance* kode program.

## DAFTAR PUSTAKA

- [1] Rinaldi, M. (n.d.). *Pencocokan string 2021*. Diakses 25 Mei 2024 melalui  
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf>
- [2] Rinaldi, M. (n.d.). *String Matching dengan Regex 2019*. Diakses 25 Mei 2024 melalui  
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/String-Matching-dengan-Regex-2019.pdf>
- [3] GeeksforGeeks. (2024, January 31). *Introduction to Levenshtein distance*. GeeksforGeeks.  
<https://www.geeksforgeeks.org/introduction-to-levenshtein-distance/>
- [4] Rinaldi, M. (n.d.). *Review Beberapa Block Cipher (Bagian 2: Advanced Encryption Standard - AES)*. Diakses 25 Mei 2024 melalui  
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2023-2024/15-Beberapa-block-cipher-bagian2-2024.pdf>

## **LAMPIRAN**

Tautan *repository* Github : [https://github.com/wigaandini/Tubes3\\_PuntangPanting](https://github.com/wigaandini/Tubes3_PuntangPanting)

Tautan video : <https://youtu.be/d1EP67B5YDU?si=kwLHBmAmFGhucAPI>