Penerapan Algoritma *Brute Force* dan *Pattern Matching* dalam Memecahkan Sandi yang Dienkripsi dengan *XOR Chiper*

Kevin Jonathan Koswara - 13515052
Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
Kevinjonathank97@gmail.com

***Abstrak*—Seiring dengan perkembangan teknologi informasi, kebutuhan manusia terhadap alat komunikasi ikut meningkat. Banyaknya orang yang menggunakan teknologi informasi membutuhkan jaminan keamanan bagi setiap orang yang menggunakannya. Salah satu teknik pengamanan yang dapat digunakan adalah kriptografi *XOR Cipher*. Kriptografi jenis ini mengganti setiap karakter dengan cara mengoperasikannya dengan operasi XOR terhadap sebuah key. Untuk menecahkan sandi tersebut diperlukan key yang tepat. Dengan algoritma Brute Force, setiap kemungkinan key akan dicoba dijadikan key untuk memecahkan pesan terenkripsi. Hasil dekripsi kemudian dinilai dengan mencari kata-kata atau subkata yang paling sering muncul dalam percakapan, pada kasus ini bahasa Inggris. Setiap kata dan subkata yang dijadikan keyword diberi nilai. Nilai tersebut semakin besar jika kata tersebut menunjukkan ke-Inggris-an yang semakin besar, misalnya "what" akan lebih besar nilainya dibandingkan "at". Nilai-nilai tersebut dijumlahkan untuk semua keyword pada setiap kemungkinan key. Key yang memiliki nilai terbesar memiliki kemungkinan terbesar menjadi key dalam enkripsi tersebut, sehingga pesan yang dihasilkan adalah pesan yang didekripsi dengan key tersebut..**

Keywords—kriptografi, key, XOR, cipher, brute force, boyer-moore, pattern matching

I. PENDAHULUAN

Pada era perkembangan teknologi informasi, komunikasi menjadi salah satu teknologi yang paling berpengaruh dalam kehidupan manusia. Komunikasi saat ini sudah berkembang jauh dibandingkan dengan puluhan tahun lalu yang masih menggunakan surat dan telegram sebagai alat komunikasi utamanya. Contoh alat komunikasi yang paling sering digunakan adalah telepon, media sosial, pesan singkat (SMS), dan lain-lain. Keberadaan teknologi tersebut memungkinkan manusia berkomuniksi dengan cara berbincang, bahkan jika lawan bicaranya berada di belahan bumi yang lain.

Selain komunikasi, perkembangan teknologi informasi juga mengubah gaya hidup dan budaya manusia. Saat ini, hampir semua manusia mengenal adanya komputer. Komputer dapat digunakan untuk berbagai macam hal, antara lain bermain, menonton video, berkomunikasi dengan orang lain, bahkan banyak pekerjaan saat ini menggunakan dan dikendalikan oleh komputer. Pekerjaan-pekerjaan seperti dokter, akuntan, manager, perusahaan cetak, dan lain-lain sudah menggunakan komputer untuk memudahkan pekerjaan manusianya. Pekerjaan-pekerjaan pribadi seperti menulis skripsi juga banyak yang menggunakan komputer dan menyimpannya di sana.

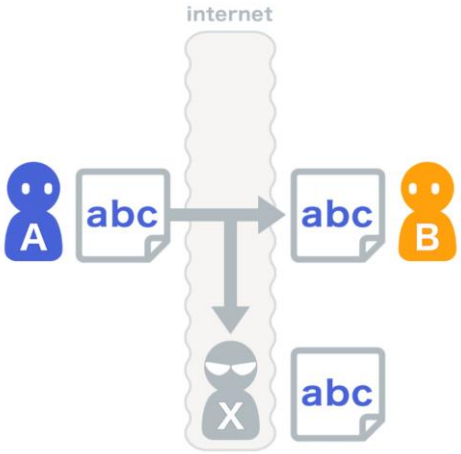

Fig. 1. Masalah dalam komunikasi : X dapat menyadap informasi rahasia dari A yang ingin diberikan kepada B

Perkembangan teknologi informasi membuat manusia semakin bergantung kepada teknologi. Akibatnya, banyak hal-hal yang bersifat rahasia yang disimpan dalam teknologi seperti komputer. Untuk itu, dibutuhkan suatu cara untuk memastikan keamanan dan kerahasiaan data-data yang disimpan ketika kita menggunakan teknologi informasi tersebut. Jika hal ini diabaikan, maka seseorang dengan niat jahat dapat mengambil data-data pribadi kita atau menyadap percakapan yang kita lakukan melalui telepon. Pengamanan yang baik harus memastikan bahwa orang ketiga tidak dapat mengetahui data-data pribadi kita, tetapi orang yang kita perbolehkan dapat mengetahuinya. Beberapa teknik pengamanan yang sudah diciptakan antara lain *password* dan kriptografi.

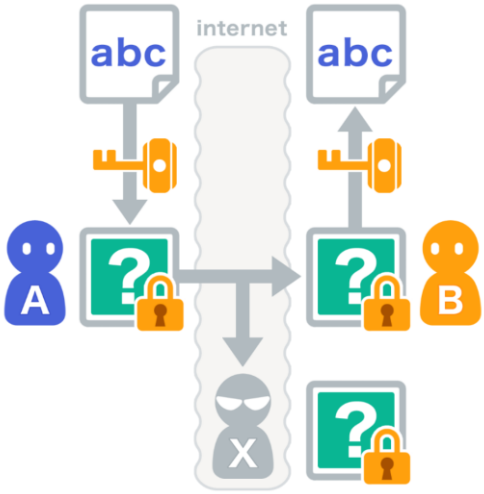

Fig. 2. Kegunaan kriptografi : walaupun X dapat menyadap pesan dari A, dia tidak bisa mengetahui isinya karena pesan tersebut dienkripsi

II. KRIPTOGRAFI

Kriptografi adalah teknik untuk mengamankan komunikasi dari orang ketiga dengan cara mengubah susunan data dengan aturan tertentu sehingga orang ketiga tidak dapat mengetahui pesan yang disembunyikannya. Kriptografi harus memperhatikan keamanan, integritas data dan otentikasi.

Sejak zaman dahulu, kriptografi sudah dipraktekan dalam bentuk *substitution cipher* seperti *Caesar Cipher* dan *transposition cipher* seperti *scytale.* Teknik tersebut sudah ditemukan bahkan sejak masa sebelum masehi. Scytale digunakan oleh bangsa Sparta untuk berkomunikasi dalam perang.

Fig. 3. *Scytale*

Kriptografi modern menggunakan aturan kriptografi yang lebih rumit. Kriptografi modern menggabungkan ilmu matematika dan *computer science*, misalnya dengan menggunakan sepasang key untuk dioperasikan dengan pangkat dan modulo. Key secara eksplisit adalah "kunci" yang digunakan untuk mengenkripsi dan dekripsi suatu pesan. Key dapat berupa string, angka atau kumpulan angka yang nantinya akan dilakukan suatu operasi pada pesan untuk mengubahnya mengjadi pesan terenkripsi. Pada beberapa kriptografi, kunci untuk mengenkripsi dan dekripsi bisa jadi sama seperti pada *XOR Cipher*. Key juga dapat dibagikan kepada publik yang disebut *public key* dan disimpan oleh sendiri yang disebut *private key.*

III. *XOR CIPHER*

Exclusive OR atau XOR adalah sebuah operasi logika, yaitu operasi yang dilakukan pada suatu nilai yang hanya terdiri dari benar atau salah. Sesuai namanya, *Exclusive* OR berarti sesuatu yang "eksklusif", hanya ada 1. XOR hanya menghasilkan nilai benar jika kedua operannya "eksklusif" atau berbeda. Operator XOR biasa dilambangkan dengan tanda $\oplus$. Tabel berikut menunjukkan nilai kebenaran operasi XOR.

TABLE I. TABEL KEBENARAN XOR

Operasi	Hasil
$true \oplus true$	$false$
$true \oplus false$	$true$
$false \oplus true$	$true$
$false \oplus false$	$false$

Dalam ilmu informatika, XOR adalah operasi bit. Operator XOR akan mengoperasikan setiap bit dari kedua operan dengan XOR, nilai benar dilambangkan dengan 1 dan salah dilambangkan dengan 0. Sebagai contoh,

$$\begin{array}{l} 13 : 1101 \\ 5 \ : 0101 \\ - - - - - \\ \quad 1000 = 8 \end{array}$$

$13 \oplus 5 = 8$ dalam informatika. Hal yang menarika dari operator XOR adalah jika kita balik prosesnya,

$$\begin{array}{l} 8 \ : 1000 \\ 5 \ : 0101 \\ - - - - - \\ \quad 1101 = 13 \end{array}$$

$8 \oplus 5 = 13$, nilainya akan kembali seperti semula. Dengan kata lain, kriptografi XOR menggunakan key yang sama untuk enkripsi dan dekripsi.

IV. ALGORITMA *BRUTE FORCE*

Brute Force adalah teknik dalam pembuatan algoritma di mana algoritma ini memecahkan masalah "secara eksplisit". Teknik *Brute Force* mentranslasikan soal menjadi algoritma secara lansung. Biasanya, algoritma *Brute Force* akan mengeksekusi setiap kondisi yang mungkin hingga selesai.

Penggambaran algoritma *Brute Force* akan digambarkan dengan persoalan pengurutan. Definisi pengurutan (menaik) adalah "setiap elemen pada posisi ke-*i* (E_i) pasti lebih kecil atau sama dengan elemen-elemen ke-*i+1* hingga *n* dan lebih besar atau sama dengan elemen-elemen ke-*1* hingga *i-1* untuk data sebanyak *n*" atau $E_1 \leq E_2 \leq \cdots \leq E_{n-1} \leq E_n$. Maka, untuk data-data pada elemen ke-*i* hingga *n*, E_i haruslah menjadi elemen terkecil, sehingga algoritma *Brute Force* untuk pengurutan data dalam bahasa C++ menjadi :

```
void selectionSort(int arr[], int n)
{
    int i, j, min_idx;

    // One by one move boundary of unsorted
    // subarray
    for (i = 0; i < n-1; i++)
    {
        // Find the minimum element in
        // unsorted array
        min_idx = i;
        for (j = i+1; j < n; j++)
          if (arr[j] < arr[min_idx])
            min_idx = j;

        // Swap the found minimum element
        // with the first element
        swap(&arr[min_idx], &arr[i]);
    }
}
```

Pada umumnya, algoritma *Brute Force* membutuhkan waktu yang sangat lama untuk beberapa kasus sehingga algoritma *Brute Force* jarang digunakan jika programnya membutuhkan waktu proses yang cepat. Hal ini dapat dibandingkan dengan algoritma *Quicksort* yang saat ini merupakan algoritma pengurutan tercepat. Algoritma *quicksort* dalam C++ adalah sebagai berikut :

```
int partition (int arr[], int low, int high)
{
    int pivot = arr[high];    // pivot
    int i = (low - 1);  // Index of smaller
                        //  element

    for (int j = low; j <= high- 1; j++)
    {
        // If current element is smaller
        // than or equal to pivot
        if (arr[j] <= pivot)
        {
            i++;    // increment index of
                    // smaller element
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}
```

```
void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
        // pi is partitioning index, arr[p]
        // is now at right place */
        int pi = partition(arr, low, high);

        // Separately sort elements before
        // partition and after partition
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
```

Misalkan $T_B(n)$ adalah jumlah perbandingan algoritma *Brute Force* dan $T_Q(n)$ adalah jumlah perbandingan algoritma Quicksort. Algoritma *Brute Force* melakukan perbandingan untuk setiap $i = 0..(n-1)$ sebanyak $j = (i+1)..n = n-i$ kali, sehingga,

$$T_B(n) = n + n - 1 + \cdots + 1$$
$$= \frac{n(n+1)}{2}$$

Algoritma *Quicksort* melakukan perbandingan sebanyak n kali dan memecah tabel menjadi 2 bagian yang kurang lebih sama besar. Misalkan $n = 2^k$, maka,

$$T_B(n) = n + 2.T_B\left(\frac{n}{2}\right)$$
$$= n + 2\left(\frac{n}{2} + 2.T_B\left(\frac{n}{4}\right)\right)$$
$$= \ ...$$
$$= n + n + \cdots + n \ (k \ kali)$$
$$= k.n$$
$$= n.\log_2 n$$

Berdasarkan jumlah perbandingan, dapat dilihat untuk n yang sangat besar, algoritma *Brute Force* sangat lambat dibandingkan algoritma *Quicksort*. Walaupun begitu, algoritma *Brute Force* masih sering digunakan untuk beberapa persoalan yang tidak bisa atau belum ditemukan algoritma hasil optimasinya. Selain itu, algoritma *Brute Force* juga banyak digunakan dalam tulisan ilmiah untuk membandingkan kecepatan algoritma lain karena kecepatan algoritma *Brute Force* adalah kecepatan terlambat yang paling efektif untuk suatu persoalan.

V. *PATTERN MATCHING*

Pattern matching adalah teknik untuk mencari sebuah pola pada teks yang panjang. *Pattern matching* sangat banyak digunakan dalam bidang komputer, contohnya adalah pencarian kata dalam *web browser*. Masalah pada *pattern matching* adalah pada umumnya, teks yang digunakan sangatlah panjang, misalnya dalam *web browser* bisa mencapai puluhan ribu bahkan ratusan ribu atau jutaan karakter. Jika menggunakan algoritma *Brute Force*, waktu yang dibutuhkan untuk mencari kata tersebut akan sangat lama. Untuk itu, algoritma untuk *pattern matching* perlu dioptimasi. Salah satu algoritma *pattern matching* yang cepat adalah algoritma *Boyer-Moore*.

A. *Boyer-Moore Algorithm*

Algoritma *Boyer-Moore* (BM) memiliki 2 karakteristik yang berbeda dengan algoritma *pattern matching Brute Force*.

1) Looking-Glass Technique

Algoritma BM melakukan pencocokan string dimulai dari ujung kanan pola, dimulai dari ujung kiri teks, berbeda dari algoritma-algoritma lain yang melakukan pencocokan dari kiri. Tujuan dari teknik ini adalah untuk memaksimalkan *Character Jump*.

2) Character Jump Technique

Character Jump Technique adalah teknik dalam algoritma BM untuk melewati pengecekan beberapa karakter dengan mengikuti aturan tertentu karena karakter-karakter tersebut tidak mungkin berupa pola yang dicari. Pada algoritma BM, terdapat 3 aturan *character jump* dan diperiksa secara berurutan.

Misalkan teks adalah T dan pola yang dicari adalah P, maka aturan *character jump* pada algoritma BM adalah :

1. Jika terjadi *mismatch* pada T[i] = x dan P[j], jika P mengandung x maka coba geser P ke kanan agar x terakhir pada P sejajar dengan T[i], lalu mulai memeriksa dari ujung kanan P.

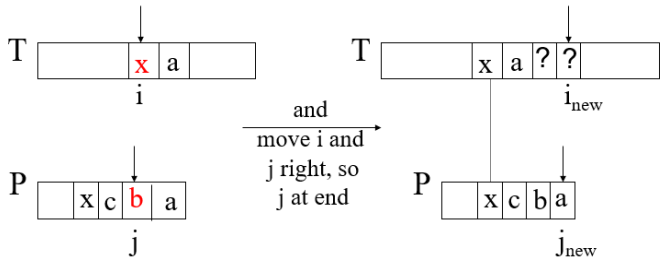

Fig. 4. Aturan *character jump* algoritma Boyer-Moore pertama

2. Jika terjadi *mismatch* pada T[i] = x dan P[j], jika P mengandung x tetapi tidak bisa digeser ke kanan sesuai aturan 1, maka geser P ke kanan sebanyak 1 karakter, lalu mulai memeriksa dari ujung kanan P.

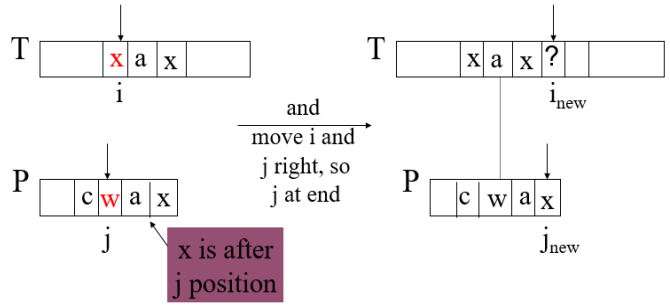

Fig. 5. Aturan *character jump* algoritma Boyer-Moore kedua

3. Jika terjadi *mismatch* pada T[i] = x dan P[j], dan P tidak mengandung x, maka geser P ke kanan sehingga karakter pertama P sejajar dengan T[i+1].

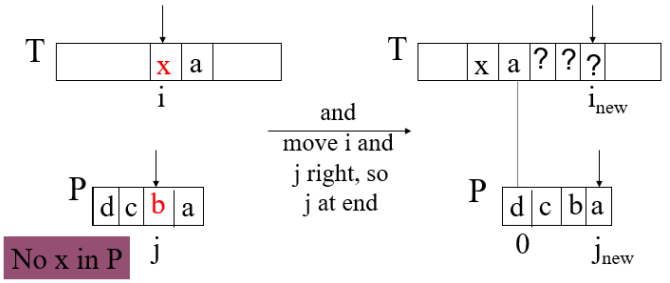

Fig. 6. Aturan *character jump* algoritma Boyer-Moore ketiga

Pada aturan 1, algoritma BM membutuhkan informasi posisi terakhir kemunculan sebuah karakter pada pola. Karena semua karakter posisi kemunculan terakhirnya sama, maka nilai tersebut disimpan pada sebuah tabel dan dijadikan fungsi yang disebut *last occurence function*. *Last occurence function* didefinisikan sebagai L(x) = indeks i terbesar dimana P[i] = x, atau L(x) = -1 jika x tidak ada dalam P. Sebagai contoh, untuk P = "abcaba", nilai L(x) adalah :

TABLE II. TABEL L(X) DENGAN P = "ABCABA"

x	a	b	c	Yang lain
L(x)	5	4	2	-1

jika indeks pertama dimulai dari 0. Nilai dari L(x) dihitung hanya sekali untuk sebuah P.

Dalam Java, algoritma *Boyer-Moore* menjadi :

```
public static int bmMatch(String text,
                          String pattern)
{
  int last[] = buildLast(pattern);
  int n = text.length();
  int m = pattern.length();
  int i = m-1;

  if (i > n-1)
    return -1; // no match if pattern is
longer than text
  int j = m-1;
  do {
    if (pattern.charAt(j) == text.charAt(i))
      if (j == 0)
        return i; // match
      else {  //  looking-glass technique
        i--;
        j--;
      }
    else {  // character jump technique
      int lo = last[text.charAt(i)]; //last
                                       occ
      i = i + m - Math.min(j, 1+lo);
      j = m - 1;
    }
  } while (i <= n-1);

  return -1; // no match
} // end of bmMatch()
```

VI. PEMECAHAN SANDI *XOR CIPHER*

Dalam memecahkan sandi *XOR Cipher*, pemecah sandi harus mengetahui key yang digunakan untuk mengenkripsi sandi karena key tersebut juga digunakan untuk mendekripsi sandi. Namun, pada prakteknya, key untuk mendekripsi pesan tidak pernah dibagikan begitu saja kepada pihak lain. Untuk itu, pemecah sandi perlu mencari key yang tepat untuk memecahkan sandi tersebut. Selama jumlah kombinasi key terbatas, setiap sandi pasti bisa dipecahkan secara teori hanya saja waktu yang dibutuhkan mungkin sangat lama.

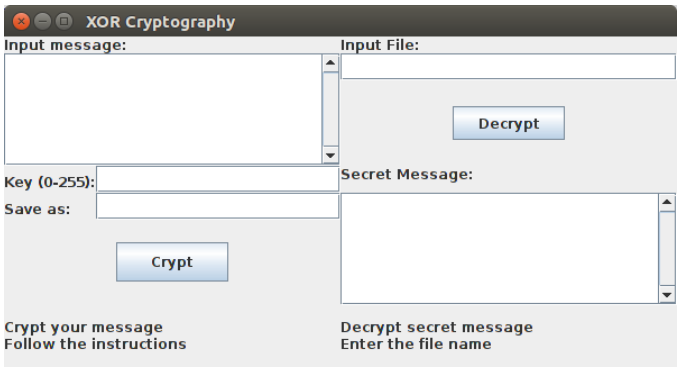

Fig. 7. Tampilan antar muka program pemecah sandi XOR yang dibuat penulis

Penulis membuat sebuah program sederhana menggunakan bahasa Java dengan GUI (Java Swing) untuk mengenkripsi dan mendekripsi file teks berisi pesan dalam bahasa Inggris yang dienkripsi dengan *XOR Cipher*. Program tersebut dapat dilihat pada https://github.com/SpiritCK/XOR-CryptoSolver/blob/master/CryptSolver.java. Pada program yang dibuat oleh penulis, proses pemecahan sandi terbagi atas 3 tahap

1) Pencarian key dengan algoritma Brute Force

Dalam konteks ilmu informatika, jumlah kombinasi key yang mungkin untuk *XOR Cipher* sama dengan jumlah kombinasi bit untuk representasi data tipe *char* yaitu 2^8 atau 256 kemungkinan. Karena jumlah key yang termasuk kecil – dengan pendekatan bahwa komputer dapat memproses 100.000.000 (seratus juta) proses setiap 1 detik – maka algoritma *Brute Force* dapat diterapkan untuk mencoba semua kemungkinan key. Untuk setiap key, setiap karakter dalam pesan terenkripsi akan dioperasikan dengan operator XOR terhadap key tersebut sehingga didapat 256 *string* yang berupa kandidat pesan rahasia. Berikut adalah potongan program yang mencoba seluruh kemungkinan key dengan algoritma *Brute Force*.

```
public String[] solve(char[] buff) {
    //...
    for (char key = 0; key < 256; key++) {
        int messageLength = buff.length - 1;
        while (buff[messageLength] == 0) {
            messageLength--;
        }
        String temp = new String();
        //...
        for (int i = 0; i <= messageLength;
i++) {
            char c = (char) (buff[i] ^ key);
            temp += c;
            //...
        }
        //...
        //temp berisi kandidat pesan rahasia
    }
    //...
}
```

2) Menghitung nilai point setiap kandidat

Dari 256 kandidat pesan rahasia, terdapat masalah baru yaitu bagaimana menentukan pesan yang sebenarnya. Jika diasumsikan pesan tersebut merupakan pesan yang bisa dibaca oleh manusia dan dalam bahasa inggris, maka pesan yang benar pasti yang paling memiliki ciri-ciri bahasa inggris dari semua kandidat pesan.

Dalam bahasa Inggris, terdapat kata-kata dan *digraph* yang sering muncul dalam sebuah kalimat sembarang. Kata-kata dan *digraph* tersebut dijadikan *keyword* untuk menentukan nilai ke-Inggris-an pesan dalam program ini. Setiap keyword diberi nilai berdasarkan panjang dan frekuensi kemunculannya. Kata-kata yang panjang akan memiliki nilai yang jauh lebih besar daripada kata-kata yang pendek, misalnya "what" bernilai lebih tinggi dari "at". Walaupun sama-sama memiliki arti dalam bahasa Inggris, potensi kemunculan "at" jauh lebih besar daripada "what". Maka, jika dalam kandidat pesan ditemukan kata "what", kandidat tersebut lebih mungkin menjadi pesan rahasia yang dimaksud dibandingkan kandidat yang hanya memiliki kata "at". Tabel 3 menunjukkan *keyword* dan nilainya yang digunakan dalam program penulis. Nilai-nilai tersebut bisa saja diganti untuk meningkatkan akurasi program.

TABLE III. TABEL *KEYWORD* UNTUK PENILAIAN KANDIDAT PESAN RAHASIA

Kata	Nilai	Kata	Nilai	Kata	Nilai
th	1	me	3	let	100
er	1	my	3	put	100
re	1	up	3	say	100
ed	1	go	3	she	100
nd	1	no	3	too	100
ha	1	us	3	use	100
en	1	am	3	when	600
es	1	the	100	make	600
nt	1	and	100	like	600
ea	1	for	100	just	600
ti	1	are	100	what	600
st	1	but	100	will	600
io	1	not	100	your	600
le	1	you	100	from	600
ou	1	all	100	they	600
ar	1	any	100	know	600
de	1	had	100	want	600
rt	1	her	100	been	600
ve	1	was	100	good	600
on	3	one	100	much	600
an	3	our	100	some	600
of	3	out	100	time	600
in	3	day	100	that	1000
at	3	get	100	with	1000
or	3	has	100	have	1000
it	3	him	100	this	1000
to	3	his	100	would	4000
is	3	how	100	there	4000
as	3	man	100	their	4000
he	3	see	100	about	4000

be	3	two	100	which	4000
so	3	way	100	could	4000
we	3	who	100	other	4000
by	3	boy	100	think	4000
do	3	did	100		
if	3	its	100		

Keyword-keyword tersebut dicari dalam teks kandidat dan dihitung kemunculannya. Proses pencarian ini menggunakan algoritma BM. Algoritma BM memiliki waktu proses yang terbaik dalam mencari kata pada teks berbahasa inggris. Namun, pada program yang dibuat penulis, algoritma ini dimodifikasi untuk menghitung "jumlah kemunculan pola" dan bukan "posisi kemunculan pertama" layaknya algoritma BM pada dasarnya. Jumlah kemunculan tersebut kemudian akan dikalikan dengan nilai *keyword* dan diakumulasi untuk semua *keyword* yang dimasukkan. Berikut adalah potongan program yang menghitung nilai dari setiap kandidat pesan.

```
/..
        int point = 0;
        for (Object[] o : keyword) {
            point +=
search(temp.toLowerCase(), (String) o[0]) *
(int) o[1];
            //...
        }
        //...
```

```
public int search(String txt, String pat) {
    int[] right;

    right = new int[256];
    for (int c = 0; c < 256; c++)
        right[c] = -1;
    for (int j = 0; j < pat.length(); j++)
        right[pat.charAt(j)] = j;

    int m = pat.length();
    int n = txt.length();
    int skip;
    int found = 0;
    for (int i = 0; i <= n - m; i += skip) {
        skip = 0;
        for (int j = m-1; j >= 0; j--) {
            if (pat.charAt(j) !=
txt.charAt(i+j)) {
                skip = Math.max(1, j -
right[txt.charAt(i+j)]);
                break;
            }
        }
        if (skip == 0) {
            found++;
            skip = 1;
        }
    }
    return found;
}
```

Method search adalah algoritma BM yang sudah dimodifikasi untuk menghitung jumlah kemunculan pola.

3) Menghitung nilai penalti

Pada beberapa kasus, mungkin saja terjadi kondisi di mana sebuah kandidat memiliki banyak substring yang berupa *keyword* sehingga mendapat nilai yang cukup besar. Contoh kejadian kasus ini adalah adanya kandidat pesan "hELLO#wORLD" sementara pesan aslinya adalah "Hello World". Kasus ini terjadi karena kunci yang digunakan kebetulan menghasilkan kandidat seperti itu. Untuk mencegah hal ini, diperlukan nilai "penalti" yang menunjukkan kejanggalan kalimat tersebut dibandingkan dengan kalimat natural. Beberapa poin yang menjadi penalti adalah :

- Karakter huruf besar selain di awal kata ("hELLO")
- Adanya karakter-karakter yang tidak bisa diketik melalui *keyboard* (karakter 0 yang menandakan *null*)
- Adanya karakter yang tidak biasanya muncul dalam pesan bahasa inggris ('*', '^', '~', dll.)

Setiap karakter yang memenuhi ciri-ciri tersebut akan menambahkan nilai penalti. Jika ditemukan dua kandidat dengan nilai yang sama, kandidat yang diambil adalah kandidat dengan nilai penalti yang lebih kecil. Berikut adalah program lengkap untuk proses dekripsi dengan method *search* sesuai dengan bagian VI.2.

```
public String[] solve(char[] buff) {
    int maxPoint = -1;
    int maxPenalty = -1;
    String result = new String();
    int keyUsed = -1;
    for (char key = 0; key < 256; key++) {
        int messageLength = buff.length - 1;
        while (buff[messageLength] == 0) {
            messageLength--;
        }
        String temp = new String();
        int penalty = 0;
        for (int i = 0; i <= messageLength;
i++) {
            char c = (char) (buff[i] ^ key);
            temp += c;
            if (!(((c >= 'A') && (c <= 'Z'))
                    || ((c >= 'a') && (c <=
'z'))
                    || ((c >= '0') && (c <=
'9'))
                    || (c == ',') || (c ==
'.') || (c == '!')
                    || (c == '=') || (c ==
'(') || (c == ')')
                    || (c == '&') || (c ==
'%') || (c == '$')
                    || (c == '?') || (c ==
'-') || (c == '@'))) {
                penalty++;
            }
        }
```

```
        int point = 0;
        for (Object[] o : keyword) {
            point +=
search(temp.toLowerCase(), (String) o[0]) *
(int) o[1];
        }
        for (int i = 0; i <= messageLength;
i++) {
            if (temp.charAt(i) >= 'A' &&
temp.charAt(i) <= 'Z' &&
                    !(i == 0 ||
temp.charAt(i-1) == ' ')) {
                penalty++;
            }
        }
        if ((point > maxPoint) || (point ==
maxPoint && penalty < maxPenalty)) {
            result = temp;
            keyUsed = key;
            maxPoint = point;
            maxPenalty = penalty;
        }
    }
    String[] answer = new String[2];
    answer[0] = result;
    answer[1] = Integer.toString(keyUsed);

    return answer;
}
```

VII. Analisis Percobaan

Untuk membuktikan keakuratan teknik dekripsi yang sudah dijelaskan di atas, penulis mencoba beberapa pesan dalam bahasa Inggris yang dienkripsi untuk didekripsi oleh program.

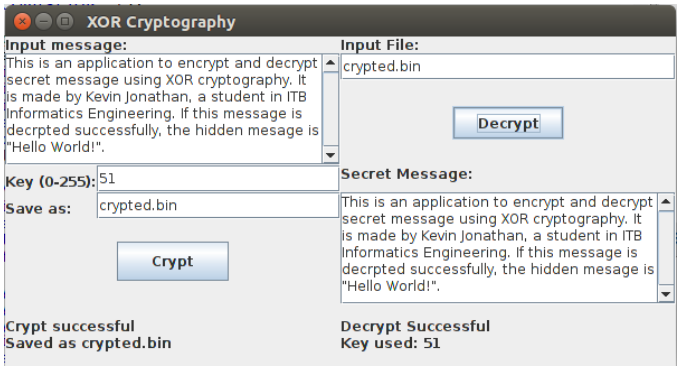

Fig. 8. Contoh dekripsi pesan dalam bahasa inggris

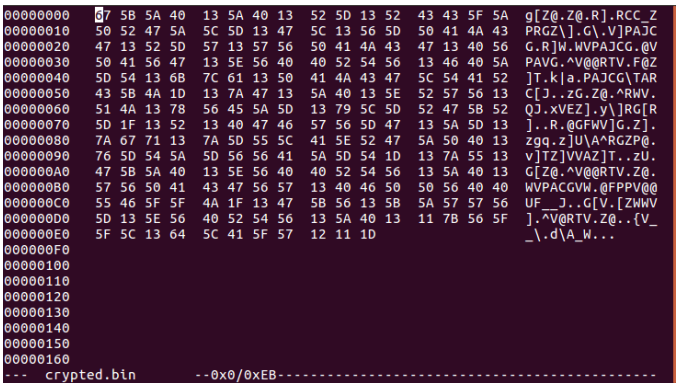

Fig. 9. Isi *crypted.bin*, hasil enkripsi pada Fig.8

Berdasarkan hasil percobaan yang dilakukan, teknik ini sudah dibuktikan dapat mendekripsi pesan bahasa Inggris dengan baik. Semakin panjang isi pesan enkripsi, maka tingkat akurasinya semakin baik karena variansi nilai dari setiap kandidat semakin besar. Untuk beberapa kasus, teknik dekripsi ini juga dapat memecahkan pesan berbahasa Indonesia karena adanya kemunculan beberapa *keyword*.

Fig. 10. Contoh dekripsi pesan dalam bahasa indonesia

```
Ini adalah pesan dalam bahasa indonesia. Walaupun aplikasi ini tidak
dibuat untuk memecahkan pesan berbahasa indonesia, ada kemungkinan
berhasil. Jika berhasil, pesan rahasianya adalah "Hai Dunia!"
```

Fig. 11. Kemunculan keyword pada pesan bahasa indonesia Fig. 10

Tetapi, teknik ini hanya bisa digunakan untuk mendekripsi pesan berupa bahasa inggris natural. Jika pesan rahasia berupa kode atau susunan huruf yang tidak beraturan, maka hasil dekripsinya akan salah. Selain itu, teknik ini juga tidak bisa mendekripsi pesan dalam bahasa-bahasa yang lain karena frekuensi kemunculan kata-kata dan *digraph* berbeda antar bahasa.

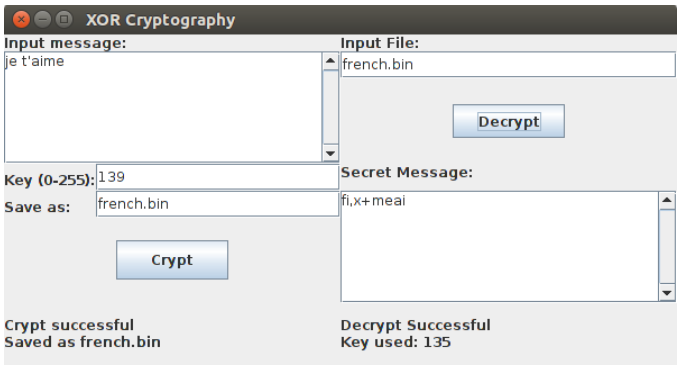

Fig. 12. Contoh kesalahan dekripsi pesan bahasa asing yang singkat

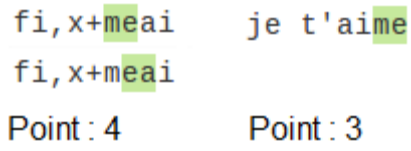

Fig. 13. Penyebab kesalahan pada contoh Fig. 12, nilai pesan asli (sebelah kanan) lebih kecil daripada salah satu kandidat (sebelah kiri)

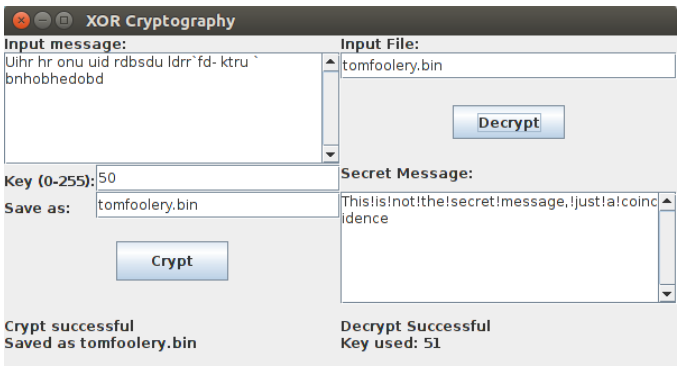

Fig. 14. Contoh kesalahan dekripsi pesan yang bukan berupa bahasa natural

VIII. KESIMPULAN

Dalam memecahkan sandi kriptografi, ada saatnya algoritma *Brute Force* dapat digunakan untuk mencoba seluruh kemungkinan key. Walaupun begitu, tetap diperlukan suatu teknik untuk menentukan key yang tepat. Untuk pesan berupa bahasa Inggris natural (percakapan sehari-hari atau pesan yang bisa dibaca dan dipahami oleh manusia), terdapat kumpulan kata-kata dan *digraph* yang sering muncul dalam teks bahasa Inggris. Kata-kata dan *digraph* tersebut dapat digunakan sebagai *keyword* untuk menentukan key yang tepat dengan cara mencari kemunculan *keyword* pada setiap teks hasil dekripsi dengan semua kemungkinan key. Algoritma *pattern matching Boyer-Moore* dapat dimodifikasi untuk menghitung jumlah kemunculan pola dan digunakan untuk mencari key yang tepat untuk mendekripsi pesan tersebut.

REFERENSI

[1] "Digraph Frequency". https://www.math.cornell.edu/~mec/2003-2004/cryptography/subs/digraphs.html. Web. 16 Mei 2017.

[2] Davison, Andrew, ditambahkan oleh Dr. Rinaldi Munir. 2017. "Pencocokan String". http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2016-2017/Pencocokan-String-(2017).ppt. Web. 16 Mei 2017.

[3] Djekic, Milica. 2013. "A Scytale – Cryptography of the Ancient Sparta". http://www.australianscience.com.au/technology/a-scytale-cryptography-of-the-ancient-sparta/. Web. 16 Mei 2017.

[4] EF Education First Ltd. "100 Most Common Words in English". http://www.ef.com/english-resources/english-vocabulary/top-100-words/. Web. 16 Mei 2017.

[5] Sale, Tony. "The Lorenz Cipher and How Bletchley Park Broke It.". https://www.codesandciphers.org.uk/lorenz/fish.htm. Web. 16 Mei 2017.

[6] Sedgewick, Robert dan Kevin Wayne. "BoyerMoore.java". http://algs4.cs.princeton.edu/53substring/BoyerMoore.java.html. Web. 16 Mei 2017

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, 29 April 2012

Kevin Jonathan Koswara
13515052