TUGAS AKHIR - KI141502

OPTIMASI KASISKI EXAMINATION PADA STUDI KASUS SPOJ THE BYTELANDIAN CRYPTOGRAPHER (ACT IV)

FREDDY HERMAWAN YUWONO
NRP 5113100040

Dosen Pembimbing I
Rully Soelaiman, S.Kom, M.Kom

Dosen Pembimbing II
Wijayanti Nurul Khotimah, S.Kom., M.Sc

Departemen INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

(Halaman ini sengaja dikosongkan)

TUGAS AKHIR - KI141502

OPTIMASI KASISKI EXAMINATION PADA STUDI KASUS SPOJ THE BYTELANDIAN CRYPTOGRAPHER (ACT IV)

FREDDY HERMAWAN YUWONO
NRP 5113100040

Dosen Pembimbing I
Rully Soelaiman, S.Kom, M.Kom

Dosen Pembimbing II
Wijayanti Nurul Khotimah, S.Kom., M.Sc

Departemen INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESIS - KI141502

OPTIMIZATION KASISKI EXAMINATION ON STUDY CASE SPOJ THE BYTELANDIAN CRYPTOGRAPHER (ACT IV)

FREDDY HERMAWAN YUWONO
NRP 5113100040

Supervisor I
Rully Soelaiman, S.Kom, M.Kom

Supervisor II
Wijayanti Nurul Khotimah, S.Kom., M.Sc

Department of INFORMATICS
Faculty of Information Technology
Institut Teknologi Sepuluh
Nopember Surabaya, 2018

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

OPTIMASI KASISKI EXAMINATION PADA STUDI KASUS SPOJ THE BYTELANDIAN CRYPTOGRAPHER (ACT IV)

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Algoritma Pemrograman
Program Studi S1 Departement Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh:
FREDDY HERMAWAN YUWONO
NRP: 5113100040

Disetujui oleh Dosen Pembimbing Tugas Akhir

Rully Soelaiman, S.Kom, M.Kom. .....
NIP: 197002131994021001 (Pembimbing 1)

Wijayanti Nurul Khotimah, S.Kom., M.Sc. .....
NIP: 198603122012122004 (Pembimbing 2)

SURABAYA
Januari 2018

(Halaman ini sengaja dikosongkan)

OPTIMASI KASISKI EXAMINATION PADA STUDI KASUS SPOJ THE BYTELANDIAN CRYPTOGRAPHER (ACT IV)

Nama	:	**FREDDY HERMAWAN YUWONO**
NRP	:	**5113100040**
Departemen	:	**Informatika FTIK**
Pembimbing I	:	**Rully Soelaiman, S.Kom, M.Kom**
Pembimbing II	:	**Wijayanti Nurul Khotimah, S.Kom., M.Sc**

Abstrak

Pada era digitalisasi ini, tingkat kebutuhan masyarakat akan informasi semakin meningkat. Hal ini menyebabkan pertukaran informasi menjadi sangat mudah. Hal ini membuat informasi yang bersifat sensitif dapat terjadi kebocoran informasi kepada pihak-pihak yang tidak berkepentingan. Kebocoran informasi terbagi menjadi dua apabila dilihat dari keutuhan informasi yang didapat, yaitu sebagian dan seutuhnya. Kebocoran informasi yang bersifat sebagian, membuat pihak-pihak yang tidak berkepentingan tetapi yang menginginkan informasi tersebut, berusaha untuk mendapatkan informasi yang utuh dari potongan-potongan informasi yang telah didapatkan.

Permasalahan dalam buku tugas akhir ini adalah permasalahan untuk mendapatkan plaintext sebanyak-banyaknya dari ciphertext dengan diberikan batas atas panjang kunci. Metode enkripsi yang digunakan adalah teknik Vigenere Cipher. Dalam permasalahan ini diberikan plaintext dan ciphertext, akan tetapi terdapat informasi yang hilang pada keduanya. Batas atas panjang kunci tersebut belum tentu panjang kunci yang sesungguhnya. Untuk dapat merekonstruksi plaintext dari kepingan informasi yang didapatkan. Diperlukan

untuk mencari panjang kunci yang didapatkan dengan cara memodifikasi Kasiski Examination dan Intersection. Beberapa hal yang perlu diperhatikan seperti mempercepat Kasiski Examination dan juga Intersection terhadap hasil yang diperoleh dari pencarian panjang kunci.

Hasil dari tugas akhir ini telah berhasil untuk menyelesaikan permasalahan yang telah diangkat dengan benar. Waktu yang diperlukan untuk dapat menyelesaikan masukan sebesar 2MB adalah 4,42 detik dengan alokasi memori sebesar 26,5MB.

***Kata-Kunci**: **ciphertext**, **Intersection**, **Kasiski Examination**, **optimisasi**, **plaintext**.*

OPTIMIZATION KASISKI EXAMINATION ON STUDY CASE SPOJ THE BYTELANDIAN CRYPTOGRAPHER (ACT IV)

Name	:	**FREDDY HERMAWAN YUWONO**
NRP	:	**5113100040**
Major	:	**Informatics FTIK**
Supervisor I	:	**Rully Soelaiman, S.Kom, M.Kom**
Supervisor II	:	**Wijayanti Nurul Khotimah, S.Kom., M.Sc**

Abstract

In this digitalization era, the level of community needs for information is increasing. This make exchanging the information very easy. This makes the sensitive information leakage to the unauthorized party. The leakage of information divided into two when viewed from the integrity of information they get. First they get all information or second they only get a partial of information. Partial information leakage, make unauthorized party to reconstruct the information they get from all piece information they have already obtain.

The problem in this undergraduate thesis book is the problem to get plaintext as much as possible from the ciphertext and upper bound of key length. The encryption method is using the Vigenere Cipher technique. Given the plaintext and ciphertext, but there is missing information in both of them. Given the upper bound of key length, that is not real key length. To reconstruct plaintext from piece information, must find the length of key from modify Kasiski Examination and Intersection. Some things to note that using only modify Kasiski Examination and Intersection can not solving the problem. Required to optimize both of them to get the answer and right time.

The results show that this problem is successfully solved. Averaging time of 4.42 second and averaging memory use about 26.5MB to solve 2MB data.
***Keywords**: **ciphertext**, **Intersection**, **Kasiski Examination**, **optimization**, **plaintext**.*

KATA PENGANTAR

Puji Syukur kepada Tuhan yang Maha Esa, atas berkatNya penulis dapat menyelesaikan buku berjudul **Optimasi Kasiski Examination pada Studi Kasus SPOJ The Bytelandian Cryptographer (Act IV)**.

Selain itu, pada kesempatan ini penulis menghaturkan terima kasih sebesar-besarnya kepada pihak-pihak yang tanpa mereka, penulis tidak akan dapat menyelesaikan buku ini:

1. ***Tuhan Yesus Kristus*** atas segala berkat, limpahan karunia, kesempatan, dan rancangan-Nya penulis masih diberi nafas kehidupan, waktu, tenaga dan pikiran untuk menyelesaikan buku ini.
2. **Alm. Papa** yang selalu menguatkan, menasehati, dan luar biasa sabar dalam mengingatkan penulis agar tidak lupa menjaga kesehatan dan selalu bersyukur selama masa studi.
3. **Mama dan saudara** yang selalu memberikan saran, dukungan, doa dan tidak lupa untuk selalu bersyukur selama masa studi.
4. **Yth. Bapak Rully Soelaiman** sebagai dosen pembimbing I yang telah banyak memberikan ilmu, bimbingan, nasihat, motivasi, serta waktu diskusi sehingga penulis dapat menyelesaikan tugas akhir ini; dan
   Yth. Ibu Wijayanti Nurul Khotimah sebagai dosen pembimbing II yang memberi bimbingan, saran teknis dan administratif, diskusi dan pemecahan masalah dalam pembuatan dan penulisan buku tugas akhir.
5. **Teman-teman Sarjana Komedi** yang telah mengingatkan, memberikan semangat dan inspirasi untuk terus melanjutkan tugas akhir di saat penulis kehilangan semangat.
6. **Teman-teman S1 Teknik Informatika 2013** yang membantu, menyemangati dan bertukar pikiran dengan penulis selama pengerjaan tugas akhir ini.
7. **Teman-teman S1 Teknik Informatika bukan 2013**, yang

telah banyak membantu, menyemangati dan bertukar pikiran dengan penulis selama pengerjaan tugas akhir ini, terutama pada Steven, Theo, Daniel, dan Glenn.

8. Serta semua pihak yang tidak tertulis, baik yang membantu dalam proses pengujian, membantu memikir saat ada masalah, dan lainnya yang telah turut membantu penulis dalam menyelesaikan Tugas Akhir ini.

Penulis menyadari bahwa Tugas Akhir ini masih memiliki banyak kekurangan. Oleh karena itu, penulis berharap kritik dan saran dari pembaca sekalian untuk memperbaiki buku ini ke depannya. Semoga tugas akhir ini dapat memberikan manfaat yang sebaik-baiknya.

Surabaya, Januari 2018

Freddy Hermawan Yuwono

DAFTAR ISI

DAFTAR TABEL

(Halaman ini sengaja dikosongkan)

DAFTAR GAMBAR

(Halaman ini sengaja dikosongkan)

DAFTAR KODE SUMBER

(Halaman ini sengaja dikosongkan)

BAB I

PENDAHULUAN

Pada bab ini akan dipaparkan mengenai garis besar Tugas Akhir yang meliputi latar belakang, tujuan, rumusan dan batasan permasalahan, metodologi pembuatan Tugas Akhir, dan sistematika penulisan.

1.1 Latar Belakang

Ketergantungan seseorang terhadap informasi tidak terlepas dari kebutuhan manusia akan informasi yang berada di sekitarnya. Informasi yang diterima seseorang pada masa sekarang dapat melalui media fisik dan media digital. Media fisik seperti koran dan majalah, sedangkan media digital seperti Facebook dan Twitter. Media-media tersebut sanggup untuk menyebarkan informasi dengan sangat cepat, sehingga orang-orang dengan cepat mengetahui informasi yang berada di sekitarnya.

Pada zaman modern ini suatu informasi, terutama yang bersifat rahasia menjadi semakin rentan akan penyalahgunaan informasi tersebut. Oleh karena itu, informasi ini akan disimpan pada tempat-tempat yang aman dan penulisan dari informasi ini pada umumnya menggunakan sandi yang hanya dimengerti oleh orang-orang yang berkepentingan terhadap informasi tersebut.

Informasi digital yang beredar di dunia maya pun tidak lepas dari penyalahgunaan informasi. Dibutuhkan suatu teknik penyandian terhadap data yang dimiliki agar data yang bersifat rahasia itu tidak diketahui dengan orang–orang yang tidak berkepentingan. Teknik penyandian terhadap data digital dapat dibagi menjadi dua jika melihat dari teknik penyandiannya yaitu *symmetric cipher* dan *asymmetric cipher*. Teknik *symmetric cipher* dapat dibagi menjadi menjadi empat bagian jika dilihat dari penyubtitusiannya yaitu *Caesar cipher*, *monoalphabetic cipher*, *polyalphabetic cipher*, dan *one time pad*. Pada dasarnya

pendeskripsian dari data yang terenkripsi dengan penyandian *symmetric cipher* dengan cara mengetahui kuncinya dan tipe dari penyubtitusiannya.

Dalam Tugas Akhir ini penulis akan mencoba mendeskripsikan informasi terbut dengan menggunakan metode *symmetric cipher* dan teknik subtitusinya menggunakan *polyalphabetic cipher*. Salah satunya dengan menggunakan modifikasi *Kasiski Examination*, akan tetapi dalam permasalahan ini apabila hanya menggunakan *Kasiski Examination* waktu yang dibutuhkan sangatlah besar, oleh karena itu penulis mengoptimasi metode yang telah ada.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam tugas akhir ini adalah sebagai berikut:

1. Bagaimana penerapan Optimasi *Kasiski Examination* untuk menyelesaikan studi kasus SPOJ *The Bytelandian Cryptographer(Act IV)*?
2. Bagaimana hasil dari kinerja Optimasi *Kasiski Examination* yang digunakan untuk menyelesaikan studi kasus SPOJ *The Bytelandian Cryptographer(Act IV)*?

1.3 Batasan Masalah

Dari permasalahan yang telah diuraikan di atas, terdapat beberapa batasan masalah pada tugas akhir ini, yaitu:

1. Bahasa pemrograman yang akan digunakan adalah bahasa pemrograman C/C++.
2. Batasan maksimum panjang dari *input file* sebesar 2 MB.
3. Batasan maksimum panjang dari batas atas *key* sebesar 100, 000 karakter.

4. *Dataset* yang digunakan adalah *dataset* pada problem SPOJ *The Bytelandian Cryptographer (Act IV)*.

1.4 Tujuan

Tujuan dari pengerjaan Tugas Akhir ini adalah:

1. Menerapkan Optimasi *Kasiski Examination* untuk menyelesaikan studi kasus SPOJ *The Bytelandian Cryptographer (Act IV)*.
2. Mengevalusi hasil dan kinerja Optimasi *Kasiski Examination* dalam menyelesaikan studi kasus SPOJ *The Bytelandian Cryptographer(Act IV)*

1.5 Metodologi

Langkah-langkah yang ditempuh dalam pengerjaan Tugas Akhir ini yaitu:

1. **Penyusunan proposal Tugas Akhir**
   Pada tahap ini dilakukan penyusunan proposal Tugas Akhir yang berisi permasalahan dan gagasan solusi yang akan diteliti pada SPOJ *The Bytelandian Cryptographer (Act IV)*.

2. **Studi literatur**
   Pada tahap ini dilakukan pencarian informasi dan studi literatur mengenai pengetahuan atau metode yang dapat digunakan dalam penyelesaian masalah. Informasi didapatkan dari materi-materi yang berhubungan dengan algoritma yang digunakan untuk penyelesaian permasalahan ini, materi-materi tersebut didapatkan dari buku, jurnal, maupun internet.
3. **Desain**
   Pada tahap ini dilakukan desain rancangan algoritma yang digunakan dalam solusi untuk pemecahan SPOJ *The*

Bytelandian Cryptographer (Act IV)

4. **Implementasi perangkat lunak**
   Pada tahap ini dilakukan implementasi atau realiasi dari rancangan desain algoritma yang telah dibangun pada tahap desain ke dalam bentuk program.
5. **Uji coba dan evaluasi**
   Pada tahap ini dilakukan uji coba kebenaran implementasi. Pengujian kebenaran dilakukan pada sistem penilaian daring SPOJ sesuai dengan masalah yang dikerjakan untuk diuji apakah luaran dari program telah sesuai.
6. **Penyusunan buku Tugas Akhir** Pada tahap ini dilakukan penyusunan buku Tugas Akhir yang berisi dokumentasi hasil pengerjaan Tugas Akhir.

1.6 Sistematika Penulisan

Buku Tugas Akhir ini bertujuan untuk mendapatkan gambaran dari pengerjaan Tugas Akhir ini. Secara garis besar, buku Tugas Akhir terdiri atas beberapa bagian seperti berikut ini:

Bab I Pendahuluan

Bab ini berisi latar belakang masalah, tujuan dan manfaat pembuatan Tugas Akhir, permasalahan, batasan masalah, metodologi yang digunakan, dan sistematika penyusunan Tugas Akhir.

Bab II Dasar Teori

Bab ini berisi dasar teori mengenai permasalahan dan garis besar penyelesaian yang digunakan dalam Tugas Akhir dan deskripsi permasalahan yang digunakan dalam Tugas Akhir.

Bab III Desain

Bab ini berisi desain algoritma yang digunakan dalam penyelesaian permasalahan.

Bab IV Implementasi

Bab ini berisi implementasi berdasarkan desain algortima yang telah dilakukan pada tahap desain.

Bab V Pengujian dan Evaluasi

Bab ini berisi uji coba dan evaluasi dari hasil implementasi yang telah dilakukan pada tahap implementasi.

Bab VI Kesimpulan dan Saran

Bab ini berisi kesimpulan dari hasil pengujian yang dilakukan, dan membahas saran untuk pengembangan algoritma lebih lanjut.

Daftar Pustaka

Merupakan daftar referensi yang digunakan untuk mengembangkan Tugas Akhir.

Lampiran

Merupakan bab tambahan yang berisi hal-hal terkait yang penting dalam aplikasi ini.

(Halaman ini sengaja dikosongkan)

BAB II

LANDASAN TEORI

Bab ini akan membahas mengenai dasar teori dan literatur yang menjadi dasar pengerjaan tugas akhir ini. Pada subbab 2.1 membahas mengenai definisi umum yang digunakan dalam memecahkan permasalahan ini. Pada subbab 2.2 membahas mengenai deskripsi permasalahan. Pada subbab 2.3 membahas mengenai contoh permasalahan. Pada subbab 2.4 membahas mengenai penyelesaian masalah secara lengkap.

2.1 Definisi Umum

Pada subbab ini membahas definisi-definisi yang digunakan sebagai dasar untuk memahami permasalahan ini dan pemecahannya.

2.1.1 Polyalphabetic Cipher

Polyalphabetic Cipher merupakan salah satu teknik untuk menenkripsi dengan menggunakan subtitusi huruf untuk menyubtitusikannya. Secara garis besar yang dimaksud dengan *polyalphabetic cipher* memiliki 2 aturan dasar yang harus dipenuhi yaitu :

1. Memiliki satu set aturan subtitusi *monoalphabetic cipher* yang digunakan.
2. Sebuah kunci mengatur suatu aturan tertentu yang dipilih untuk mengatur transformasi yang dilakukan.

Untuk memperjelas aturan di atas, dapat dilihat pada Gambar 2.1.

$$C = c_0, c_1, c_2, ..., c_{n-1}$$

$$E(K, P) = E[(k_0, k_1, ..., k_{m-1})(p_0, p_1, ..., p_{n-1})]$$
$$= (p_0 + k_0) mod 26, (p_1 + k_1) mod 26, ..., (p_{m-1} + k_{m-1}) mod 26,$$
$$(p_m + k_0) mod 26, (p_{m+1} + k_1) mod 26, ..., (p_{2m-1} + k_{m-1}) mod 26, ...$$

Gambar 2.1 Aturan *Polyalphabetical Cipher*

C adalah *ciphertext*, *E* adalah fungsi enkripsi, *K* adalah kunci, dan *P* adalah *plaintext*. Salah satu turunan dari *polyalphabetic cipher* adalah teknik *Vigenere Cipher* yang menjadi dasar permasalahan yang diangkat dalam tugas akhir ini [1].

2.1.2 Ciphertext

Ciphertext adalah suatu pesan / teks acak yang dihasilkan dari suatu algoritma kriptografi. Contoh dari *ciphertext* dalam kasus *polyalphabetical cipher* adalah ”RTPPRKGFI” yang merupakan hasil enkripsi dari ”PLAINTEXT” dan menggunakan kunci ”CIPHER” [2].

2.1.3 Plaintext

Plaintext adalah data asli sebagai masukan dari suatu metode enkripsi yang akan dilakukan [2]. Biasanya merupakan suatu rangkaian kata yang masih dapat dipahami artinya atau hasil keluaran dari suatu algoritma kriptografi yang akan dienkripsi lagi.

2.1.4 Secret Key

Secret Key atau yang lebih dikenal dengan *key* adalah suatu masukan dari algoritma enkripsi yang akan menentukan suatu

transformasi dan subtitusi yang akan dilakukan oleh algoritma enkripsi[2]. Dalam kasus *polyalphabetical cipher* pada permasalahan yang diangkat dalam tugas akhir ini, memiliki panjang kunci setidaknya 1.

2.1.5 Friedman test

Friedman test atau *Kappa Test* merupakan salah satu metode yang digunakan untuk mendeskripsikan *Polyalphabetical Cipher* dengan menggunakan *index of coincidence*. *Index of coincidence* digunakan untuk mengukur tingkat ketidakrataan frekuensi *ciphertext*. Untuk menghitung *index of coincidence* digunakan persamaan 2.1[3].

$$IC = c * \left(\left(\frac{n_a}{N} * \frac{n_a - 1}{N - 1}\right) + \left(\frac{n_b}{N} * \frac{n_b - 1}{N - 1}\right) + \left(\frac{n_c}{N} * \frac{n_c - 1}{N - 1}\right) + \cdots + \left(\frac{n_z}{N} * \frac{n_z - 1}{N - 1}\right)\right) \quad (2.1)$$

Persamaan 2.1 dapat disederhanakan menjadi persamaan 2.2.

$$IC = \frac{\sum_{i=1}^{c} n_i(n_i - 1)}{\frac{N*(N-1)}{c}} \quad (2.2)$$

IC adalah *index of coincidence*, n_i adalah jumlah huruf ke-i yang terdapat pada suatu alfabet yang terdapat pada *ciphertext*, n_a sampai dengan n_z merupakan jumlah huruf ”a” sampai dengan huruf ”z” yang terdapat pada *ciphertext* , *N* adalah jumlah huruf yang terdapat pada *ciphertext*, dan *c* adalah jumlah huruf dalam alfabet. Alfabet yang terbentuk di dalam suatu *ciphertext* belum tentu jumlahnya pasti 26. Karena tidak dapat mengetahui jumlah huruf yang mungkin terbentuk dalam alfabet yang ada dan harus mengetahui jumlah setiap huruf yang muncul dalam suatu *ciphertext*. Karena keterbatasan informasi mengenai *ciphertext* yang didapatkan, maka metode ini tidak dapat digunakan untuk

menyelesaikan permasalahan dalam tugas akhir ini.

2.1.6 Kasiski Examination

Kasiski Examination merupakan suatu teknik yang digunakan untuk mendeskripsikan secara paksa suatu *ciphertext* yang menggunakan teknik substitusi, baik itu *polyalphabetical cipher* maupun *monoalphabetical cipher*. Teknik menggunakan kelemahan yang ditimbulkan oleh teknik substitusi itu sendiri. Yaitu apabila suatu *subtring* dari *plaintext* dan *subtring* dari suatu set kunci yang berulang terdapat yang berulang, maka dapat dipastikan untuk menebak panjang huruf / karakter kunci yang digunakan, tanpa diketahui *plaintext* dan kuncinya. Sebagai contoh dapat dilihat pada Tabel 2.1.

Tabel 2.1 Contoh *Kasiski Examintaion*

ciphertext		C	S	A	S	X	S
J	O	S	P	C	S	A	S
				X			

Dari Tabel 2.1 yang ada dapat disimpulkan bahwa kata ”CSASX” berulang. Sehingga setidaknya dapat disimpulkan bahwa panjang kuncinya mungkin 5 karakter atau faktor dari 10[4]. Cara pencarian panjangnya:

1. Mencari semua subtring yang berulang. Seperti Tabel 2.1.
2. Mencari semua faktor kapan subtring itu berulang lagi sebagai contoh Tabel 2.1 itu selisih antara *substring* ”CSASX” adalah 10. Maka, faktor dari 10 adalah 10,5,2,1.
3. Kemungkinan besar faktor yang sering berulang adalah jawabannya.

Hal ini yang menjadi dasar pengerjaan permasalahan yang diangkat dalam tugas akhir ini.

2.1.7 Intersection

Intersection adalah himpunan A dan himpunan B di mana ada bagian dari A juga merupakan bagian dari B. Sehingga dapat ditulis dengan persamaan 2.3.

$$A \cap B = \{x : x \in A \text{ dan } x \in B\} \quad (2.3)$$

Sebagai contoh *intersection* antara *{*1, 2, 3*}* dan *{*1, 4, 5*}* adalah *{*1*}*[5] .

2.2 Deskripsi Permasalahan

Permasalahan yang diangkat dalam tugas akhir ini diangkat dari suatu permasalahan yang terdapat pada suatu situs penilaian daring atau *online judge* SPOJ yaitu *The Bytelandian Cryptographer (Act IV)* dengan kode soal CRYPTO4[1] [6].

Permasalahan pada The Bytelandian Cryptographer (Act IV) diberikan pesan dengan panjang *N* huruf, huruf yang digunakan adalah huruf kapital latin dari A sampai dengan Z, yang dapat ditafsirkan menjadi bilangan bulat dari 0 sampai dengan 25. Diberikan kunci untuk mentransmisikan pesan yang diketahui oleh kedua belah pihak yang terdiri dari *M* bilangan bulat. Dengan menggunakan kunci yang ada bahwa pada index ke *i* dari pesan pada index x_i akan dienkripsikan ke dalam bentuk index ke *i* dari pesan hasil enkripsi *y*, yang mengikuti aturan 2.4.

$$y_i = x_i + k_{1+(i-1) \bmod M} \bmod 26 \quad (2.4)$$

Diketahui *plaintext* dan *ciphertext* yang diberikan hanya berupa potongan-potongan dari kedua pesan tersebut. Dicari bagaimana menkonstruksi ulang pesan yang telah didapat sehingga bisa membentuk *plaintext* yang asli dari pesan yang telah didapatkan

1 http://www.spoj.com/problems/CRYPTO4

sebanyak-banyaknya.

Deskripsi mengenai format masukan sebagai berikut:

1. Baris pertama berisi sejumlah *T* kasus uji coba
2. Setiap kasus uji coba, baris pertama berisi 1 bilangan bulat *M* di mana merupakan sejumlah batas atas panjang kunci yang digunakan
3. Baris kedua dari setiap kasus uji coba berisi *plaintext*.
4. Baris ketiga dari setiap kasus uji coba berisi *ciphertext*.

Plain text dan *ciphertext* dalam format masukan menggunakan karakter A sampai dengan Z yang dapat ditafsirkan kedalam bilangan bulat 0 sampai dengan 25 dan karakter ”*” yang dapat ditafsirkan sebagai karakter yang hilang.

Format keluaran yang dihasilkan adalah 1 baris *plaintext* yang dapat direkontruksi dan ”*” apabila nilai dari karakter pada posisi tersebut tidak dapat ditentukan. Batasan permasalahan *The Bytelandian Cryptographer (Act IV)* adalah sebagai berikut:

1. $T \leq 200$
2. $1 \leq M \leq 100,000$
3. Panjang *input file* tidak melebihi dari 2MB.
4. Lingkungan penilaian Intel Pentium G860 3GHz.
5. Batas Waktu: ≤ 17 detik
6. Batas Kode sumber : 50000B
7. Batas Memory : 1536 MB.

2.3 Contoh Kasus Permasalahan

Dalam Permasalahan yang diangkat ini huruf A sampai dengan Z ditafsirkan sebagai 0 sampai dengan 25. Contoh 1. Diketahui *M* bernilai 1 yang menunjukkan batas atas dari panjang kunci. Diketahui *plaintext* adalah A*X*C dan *ciphertext* adalah **CM*.

Tabel 2.2 Contoh 1

index	0	1	2	3	4
plain text	A	*	X	*	C
ciphertext	*	*	C	M	*
Selisih yang diketahui			5		
Hasil	A	*	X	H	C

Dari Tabel 2.2 dapat diketahui bagaimana hasil yang akan dicapai. Cara mencapai hasil yang diinginkan adalah sebagai berikut. Pertama-tama pencarian panjang kunci. Pencarian panjang kunci harus mengetahui dahulu batas atas dari panjang kunci tersebut. Contoh kasus ini *M* bernilai 1, maka kemungkinan jawabannya hanya ada 1, yaitu *M* itu sendiri, karena suatu panjang kunci yang digunakan dalam enkripsi tidak mungkin 0. Diasumsikan bahwa 1 warna mewakili satu blok kunci yang akan digunakan. Maka akan membentuk seperti pada Tabel 2.3.

Tabel 2.3 Langkah 1 Contoh 1

index	0	1	2	3	4
plain text	A	*	X	*	C
ciphertext	*	*	C	M	*
Selisih yang diketahui			5		

Karena selisih yang diketahui hanya ada 1, maka itulah yang menjadi jawabannya. Selanjutnya mencari yang *plaintext* yang bernilai ”*” dan *ciphertext* tidak kosong, seperti indeks ketiga. Maka indeks ketiga jawabannya adalah ”H”. Karena ”M” − 5 adalah ”H”.

Contoh 2. Diketahui bahwa *M* bernilai 4 yang menunjukkan batas atas dari panjang kunci. Diketahui *plaintext* adalah *B***A dan *ciphertext* adalah AAAAAA. Dalam Contoh ini

panjang kuncinya bisa dari 1 sampai dengan 4.

Tabel 2.4 Contoh 2

index	0	1	2	3	4	5
plain text	*	B	*	*	*	A
ciphertext	A	A	A	A	A	A
Selisih yang diketahui		25				0
Panjang Kunci 1	tidak bisa karena ada yang *collision*					
Panjang Kunci 2	tidak bisa karena ada yang *collision*					
Panjang Kunci 3	*	B	A	*	B	A
Panjang Kunci 4	tidak bisa karena ada yang *collision*					

Cara yang harus dilalui adalah sebagai berikut. Pertama mencari panjang kunci yang bernilai antara 1 - 4.

Tabel 2.5 Panjang Kunci 1 Contoh 2

index	0	1	2	3	4	5
plain text	*	B	*	*	*	A
ciphertext	A	A	A	A	A	A
Selisih yang diketahui		25				0
Panjang Kunci 1	tidak bisa karena ada yang *collision*					

Tabel 2.5 memperlihatkan proses pencarian panjang kunci ketika panjang kuncinya 1. Apabila 1 warna direpresentasikan menjadi 1 blok kunci, maka terdapat 5 blok kunci yang terbentuk. Tabel 2.5 bahwa hanya ada 2 indeks di mana *plaintext* dan *ciphertext* yang diketahui. Keduanya terletak pada blok kunci yang berbeda. Akan tetapi karena panjang bloknya 1 maka tidak mungkin panjang kunci 1 dilakukan karena keduanya saling bertabrakan dan membawa nilai yang berbeda. Yang menjadi permasalahan pada panjang kunci 1 adalah ketika blok

yang diketahui bertabrakan dan masing-masing membawa nilai yang berbeda juga. Sehingga dapat disimpulkan bahwa panjang kunci 1 pada contoh kasus 2 tidak mungkin terjadi.

Tabel 2.6 Panjang Kunci 2 Contoh 2

index	0	1	2	3	4	5
plain text	*	B	*	*	*	A
ciphertext	A	A	A	A	A	A
Selisih yang diketahui		25				0
Panjang Kunci 2	tidak bisa karena ada yang *collision*					

Tabel 2.6 memperlihatkan proses pencarian panjang kunci, ketika panjang kuncinya 2. Apabila 1 warna direprentasikan menjadi 1 blok kunci, maka terdapat 3 blok kunci yang terbentuk. Tabel 2.6 terlihat bahwa terdapat 2 indeks di mana *plaintext* dan *ciphertext* yang diketahui. Indeks yang pertama terletak pada blok pertama bagian akhir, begitu juga indeks yang kedua terletak pada blok 3 bagian akhir. Karena keduanya terletak pada bagian akhir dari masing-masing blok maka dapat disimpulkan panjang kunci 2 juga tidak dapat digunakan.

Tabel 2.7 Panjang Kunci 3 Contoh 2

index	0	1	2	3	4	5
plain text	*	B	*	*	*	A
ciphertext	A	A	A	A	A	A
Selisih yang diketahui		25				0
Panjang Kunci 3	*	B	A	*	B	A

Tabel 2.7 memperlihatkan proses pencarian panjang kunci, ketika panjang kuncinya 3. Apabila 1 warna direpresentasikan

menjadi 1 blok kunci, maka terdapat 2 blok kunci yang terbentuk. Tabel 2.7 terlihat bahwa terdapat 2 indeks di mana *plaintext* dan *ciphertext* yang diketahui. Indeks pertama terletak dibagian tengah pada blok 1 dan indeks kedua terletak pada bagian akhir pada blok 2. Maka, *plaintext* yang terbentuk adalah pada bagian akhir dari blok kunci 1 dan bagian tengah dari blok kunci 2. Karena keduanya saling melengkapi antara satu bagian dengan bagian lainnya. Sehingga hasil yang terbentuk pada bagian akhir dari blok kunci 1 adalah ”A” − 0 adalah ”A” dan bagian tengah dari blok kunci 2 adalah ”A” −25 + 26 adalah ”B”.

Tabel 2.8 Panjang Kunci 4 Contoh 2

index	0	1	2	3	4	5
plain text	*	B	*	*	*	A
ciphertext	A	A	A	A	A	A
Selisih yang diketahui		25				0
Panjang Kunci 4	tidak bisa karena ada yang *collision*					

Tabel 2.8 memperlihatkan proses pencarian panjang kunci, ketika panjang kuncinya 4. Apabila 1 warna direpresentasikan menjadi 1 blok kunci, maka terdapat 2 blok kunci yang terbentuk. Tabel 2.8 terlihat bahwa terdapat 2 indeks di mana *plaintext* dan *ciphertext* yang diketahui. Indeks yang pertama terletak dibagian kedua dari depan pada blok 1 dan indeks yang kedua terletak pada bagian kedua dari depan pada blok 2. Maka dalam ini tidak bisa digunakan panjang kunci 4, karena terdapat bagian dari kedua blok pada panjang kunci 4 yang bertabrakan dan membawa nilai yang berbeda. Sehingga hasil yang terbentuk dapat dilihat pada Tabel 2.9.

Tabel 2.9 Hasil Contoh 2

index	0	1	2	3	4	5
plain text	*	B	*	*	*	A
ciphertext	A	A	A	A	A	A
Selisih yang diketahui		25				0
Hasil	*	B	A	*	B	A

Contoh 3. Diketahui bahwa *M* bernilai 4 yang menunjukkan batas atas dari panjang kunci. Diketahui *plaintext* adalah *AA******* dan *ciphertext* adalah AAAAAAAAAA. Indeks akan dihitung mulai dari 0.

Tabel 2.10 Contoh 3

index	0	1	2	3	4	5	6	7	8	9
plain text	*	A	A	*	*	*	*	*	*	*
ciphertext	A	A	A	A	A	A	A	A	A	A
Selisih yang diketahui		0	0							

Cara yang harus dilalui adalah sebagai berikut. Pertama mencari panjang kunci yang bernilai antara 1 - 4.

Tabel 2.11 Panjang Kunci 1 Contoh 3

index	0	1	2	3	4	5	6	7	8	9
plain text	*	A	A	*	*	*	*	*	*	*
ciphertext	A	A	A	A	A	A	A	A	A	A
Selisih yang diketahui		0	0							
Panjang kunci 1	A	A	A	A	A	A	A	A	A	A

Tabel 2.11 memperlihatkan proses pencarian panjang kunci, ketika panjang kuncinya 1. Apabila 1 warna merepresentasikan

sebagai 1 blok kunci maka, dapat terbentuk sebanyak 10 blok kunci. Dapat dilihat pada Tabel 2.11. Indeks yang diketahui baik *plaintext* dan *ciphertext* adalah indeks 1 dan indeks 2, yang mana keduanya terletak pada blok yang berbeda. Akan tetapi, karena mereka membawa 1 nilai yang sama maka semua *plaintext* yang kosong pasti bernilai ”A”.

Tabel 2.12 Panjang Kunci 2 Contoh 3

index	0	1	2	3	4	5	6	7	8	9
plain text	*	A	A	*	*	*	*	*	*	*
ciphertext	A	A	A	A	A	A	A	A	A	A
Selisih yang diketahui		0	0							
Panjang kunci 2	A	A	A	A	A	A	A	A	A	A

Tabel 2.12 memperlihatkan proses pencarian panjang kunci, ketika panjang kuncinya 2. Apabila 1 warna merepresentasikan sebagai 1 blok kunci maka, dapat terbentuk sebanyak 5 blok kunci. Dapat dilihat pada Tabel 2.12. Indeks yang diketahui baik *plaintext* dan *ciphertext* adalah indeks 1 dan indeks 2, yang mana keduanya terletak pada blok yang berbeda dan posisi yang berbeda pula. Hal ini dapat dilihat pada Tabel 2.12. Indeks yang diketahui pertama terletak pada bagian akhir dari blok 1 dan indeks yang diketahui kedua terletak pada bagian awal dari blok 2. Maka dapat disimpulkan bahwa seluruh *plaintext* yang nilai ”*” dan *ciphertext* yang nilainya tidak ”*”, nilai dari *plaintext* yang terbentuk seluruhnya ”A”, seperti yang terlihat pada Tabel 2.12.

Tabel 2.13 Panjang Kunci 3 Contoh 3

index	0	1	2	3	4	5	6	7	8	9
plain text	*	A	A	*	*	*	*	*	*	*
ciphertext	A	A	A	A	A	A	A	A	A	A
Selisih yang diketahui		0	0							
Panjang kunci 3	*	A	A	*	A	A	*	A	A	*

Tabel 2.13 memperlihatkan proses pencarian panjang kunci, ketika panjang kuncinya 3. Apabila 1 warna merepresentasikan sebagai 1 blok kunci maka, dapat terbentuk sebanyak 4 blok kunci. Dapat dilihat pada Tabel 2.13. Indeks yang diketahui baik *plaintext* dan *ciphertext* adalah indeks 1 dan indeks 2, di mana semuanya terletak pada satu blok yang sama, yaitu blok kunci 1. Jadi, sisanya menngikuti perulangan dari blok kunci 1.
Pada Tabel 2.14 akan diperlihatkan proses pencarian ketika panjang kunci adalah 4.

Tabel 2.14 Panjang Kunci 4 Contoh 3

index	0	1	2	3	4	5	6	7	8	9
plain text	*	A	A	*	*	*	*	*	*	*
ciphertext	A	A	A	A	A	A	A	A	A	A
Selisih yang diketahui		0	0							
Panjang kunci 4	*	A	A	*	*	A	A	*	*	A

Tabel 2.14 memperlihatkan proses pencarian panjang kunci, ketika panjang kuncinya 4. Apabila 1 warna merepresentasikan sebagai 1 blok kunci maka, dapat terbentuk sebanyak 4 blok kunci. Dapat dilihat pada Tabel 2.14. Indeks yang diketahui baik *plaintext* dan *ciphertext* adalah indeks 1 dan indeks 2, di mana semuanya terletak pada satu blok yang sama, yaitu blok kunci 1. Jadi, sisanya mengikuti perulangan dari blok kunci 1.

Tabel 2.15 Hasil dari Contoh 3

index	0	1	2	3	4	5	6	7	8	9
plain text awal	*	A	A	*	*	*	*	*	*	*
Panjang kunci 1	A	A	A	A	A	A	A	A	A	A
Panjang kunci 2	A	A	A	A	A	A	A	A	A	A
Panjang kunci 3	*	A	A	*	A	A	*	A	A	*
Panjang kunci 4	*	A	A	*	*	A	A	*	*	A
Hasil Akhir	*	A	A	*	*	A	*	*	*	*

Pada contoh ini kalau dilihat pada indeks selisih yang telah diketahui, terjadi bahwa panjang kunci 1 sampai dengan 4 dapat tercipta sedangkan pada contoh sebelumnya tidak bisa. Hal ini juga dipengaruhi oleh karena isinya itu sama. Walaupun terjadi *collision* pada indeks yang selisih yang diketahui, tetapi kalau nilainya sama maka nilai tersebut merupakan jawaban untuk setiap panjang kunci bisa terbentuk. Keempat panjang kunci tersebut benar terhadap pesan tersebut. Hasil keluaran yang diinginkan hanya 1 saja tetapi keempatnya benar, oleh karena itu perlu dilakukan *intersection* atau perpotongan dari himpunan keempat kunci tersebut. Perpotongan itu menghasilkan ,A,A,,A,,,,,. Terdapat bagian bagian yang kosong yang dapat diisi dengan *. Sehingga hasil akhirnya dapat dilihat pada Tabel 2.15 pada bagian hasil akhir.

2.4 Penyelesaian Masalah The Bytelandian Cryptographer (Act IV)

Permasalahan *The Bytelandian Cryptographer (Act IV)* dapat diselesaikan dengan menggunakan *Kasiski Examination* dan *Intersection*. Untuk menyelesaikan masalah ini perlu ditafsirkan bahwa karakter A sampai dengan Z menjadi 0 sampai dengan 25, karena untuk memudahkan perhitungan mencari selisih dan

merekonstruksi *plaintext* dari *ciphertext* dan karakter kunci. Berikut ini tahapan-tahapan untuk menyelesaikan masalah ini:

1. Menyimpan posisi indeks karakter, di mana pada indeks tersebut baik *ciphertext* maupun *plaintext* tidak bernilai '*', beserta menyimpan hasil perhitungan selisih antara *ciphertext* dan *plaintext* [7].
2. Menyimpan posisi indeks, apabila *ciphertext* diketahui dan *plaintext* tidak diketahui[7].
3. Tahapan ini adalah modifikasi dari *Kasiski Examination*. Apabila menggunakan *Kasiski Examination* pada umumnya yang hanya mencari posisi berulang dari suatu sub kalimat dalam suatu kalimat tidak bisa digunakan untuk menyelesaikan permasalahan ini. Oleh karena itu diubah menjadi mengiterasi panjang kunci dari 1 sampai dengan *M*, yang akan digunakan untuk membagi selisih yang diketahui antara *plaintext* dan *ciphertext* sebesar posisi iterasi yang telah berjalan. Tahapan selanjutnya melihat dalam blok-blok yang telah terbentuk ini terdapat *collision* dan indeks yang saling bertabrakan memiliki nilai yang sama atau tidak. Apabila sama maka tidak terjadi tabrakan. Sebaliknya jika terjadi tabrakan maka harus mencari panjang kunci yang baru. Panjang kunci dapat dicari dari $\frac{M}{2} + 1 <= N <= M$. Panjang kunci dapat dicari dari $\frac{M}{2} + 1$ dan tidak dari 1 karena suatu panjang kunci bernilai benar maka kelipatan dari panjang kunci itu pun juga pasti benar dan untuk mempersingkat waktu *running time* yang seharusnya terjadi. Dasar pernyataan ini dari Tabel 2.4 pada bagian 2.2. Pada bagian ini dilakukan untuk mencari panjang kunci yang benar dengan cara mengiterasi hasil yang diperoleh pada tahap satu *modulo* dengan posisi iterasi yang dilakukan. Apabila tidak terjadi
   konflik maka panjang kunci tersebut benar. Jika sebaliknya yang terjadi maka, panjang kunci tersebut

salah. Contohnya dapat dilihat pada contoh 2 pada bagian 2.2. Bagian ini memungkinkan bahwa bisa jadi lebih dari 1 panjang kunci yang bernilai benar. Contohnya seperti yang terjadi pada Tabel 2.10 pada bagian 2.2.

4. Melakukan *intersection* terhadap himpunan dari kunci yang telah dihasilkan [7]. *Intersection* yang dilakukan berada di dalam perulangan panjang kunci pada waktu *generate* setiap karakter yang terdapat dalam penyimpanan tahap 2 pada panjang kunci tersebut dengan ketentuan sebagai berikut:
   (a) Panjang kunci yang terbentuk harus benar.
   (b) Apabila terdapat indeks yang tidak dapat dipastikan isinya maka posisinya harus dibuang dari penyimpanan dan hasil yang diperoleh pasti '*'.
   (c) Apabila *plaintext* pada indeks tersebut nilainya masih '*' dan pada penyimpanan masih menyimpan indeks tersebut maka, hasilnya adalah *ciphertext* pada indeks tersebut dikurangi dengan hasil yang diperoleh dari tahap 1 bagian perhitungan selisih pada indeks yang sama kemudian dimodulo 26.

Sehingga kompleksitasnya $O(T * \frac{M}{2} * (N + S))$. Dengan T adalah kasus uji coba, $\frac{M}{2}$ adalah batas atas kunci dibagi dengan 2, N adalah jumlah posisi karakter yang terdapat pada tahap 2, dan S adalah jumlah posisi karakter yang terdapat pada tahap 1. Pada kondisi *worst case* $T * (N + S) = 1.000.000$, sedangkan $\frac{M}{2}$ adalah 50.000. Hasilnya 50 miliar perulangan, dengan asumsi 1 detik adalah 1 miliar perulangan maka waktu yang dibutuhkan adalah 50 detik. Oleh karena itu hal ini tidak mungkin bisa dilakukan begitu saja, sehingga diperlukan *pruning* pada sumber kode yang ada untuk memangkas waktu eksekusi program.

BAB III

DESAIN

Pada bab ini akan dijelaskan mengenai desain algoritma yang digunakan untuk menyelesaikan permasalahan pada Tugas Akhir ini.

3.1 Desain Umum Sistem

Sistem pertama kali akan menjalankan fungsi MAIN terlebih dahulu. Desain dari fungsi MAIN dapat dilihat pada Gambar 3.1.

Tabel 3.1 Masukan, Proses, dan Keluaran fungsi MAIN

Masukan	Proses	Keluaran
Diperoleh dari format masukan pada studi kasus	Menerima masukan dari studi kasus dan menjalankan fungsi SOLVE	Tidak ada

```
1: function MAIN
2:     T ← INPUT
3:     while T ≠ 0 do
4:         T = T − 1
5:         m ← input                          ▷ masukkan batas atas dari kunci
6:         message[ ] ← input                 ▷ masukkan plaintext
7:         cipher[ ] ← input                  ▷ masukkan ciphertext
8:         SOLVE(message,chiper,m)
9:     end while
10: end function
```

Gambar 3.1 Gambar Fungsi Main

Tabel 3.2 Penjelasan variabel yang digunakan dalam fungsi MAIN

Nama Variabel	Penjelasan
T	Digunakan untuk menerima dan menyimpan masukan kasus uji coba.
m	Digunakan untuk menerima dan menyimpan masukan batas atas panjang kunci dari setiap kasus uji coba.
message	Digunakan untuk menerima dan menyimpan masukan *plaintext* dari setiap kasus uji coba.
cipher	Digunakan untuk menerima dan menyimpan masukan *ciphertext* dari setiap kasus uji coba.

3.2 Desain Algoritma

Pada bagian ini akan dibahas secara rinci mengenai fungsi-fungsi yang digunakan dalam sistem.

3.2.1 Desain fungsi SOLVE

Fungsi ini digunakan untuk menyelesaikan permasalahan yang diangkat pada tugas akhir ini. Tahapan-tahapan prosesnya terdapat pada subbab 2.2 dan subbab 2.4. Pada bagian ini tidak dapat mengecek kebenaran dari suatu panjang kunci. Gambar mengenai fungsi SOLVE dapat dilihat pada Gambar 3.2 dan 3.3.

Tabel 3.3 Masukan, Proses, dan Keluaran fungsi SOLVE

Masukan	Proses	Keluaran
Plaintext, *ciphertext*, dan batas atas panjang kunci yang diperoleh dari fungsi MAIN	Mencari semua posisi, jumlah di mana *plaintext* dan *ciphertext* tidak bernilai "*". Mencari semua posisi di mana *ciphertext* tidak bernilai "*" dan *plaintext* bernilai "*". Mengiterasi panjang kunci dari $\frac{m}{2} + 1$ sampai dengan m. Melakukan *intersection*.	*Plaintext* yang telah direkonstruksi ulang

Mengenai modulo 26 yang terdapat pada Gambar 3.2 dan Gambar 3.3 digunakan untuk memastikan bahwa selisih dari *plaintext* dan *ciphertext* adalah pasti < 26. Ditambah 26 pada Gambar 3.2 dan Gambar 3.3 dimaksudkan agar silisih antara *plaintext* dan *ciphertext* selalu bernilai positif.

```
1: function SOLVE(message,chiper,m)
2:     counter_diketahui ← 0
3:     counter_yang_ingin_diketahui ← 0
4:     diketahui[ ]
5:     Selisih_diketahui[ ]
6:     ingin_diketahui[ ]
7:     Key[ ]
8:     for i = 0 to message[i] ≠ 0 ; i+ = 1 do
9:         if message[i] ≠' *' dan cipher[i] ≠' *' then
10:            diketahui[counter_diketahui] = i
11:            Selisih_diketahui[i] = (message[i] − cipher[i] + 26)%26
12:            counter_diketahui = counter_diketahui + 1
13:        else if message[i] =' *' dan cipher[i] ≠' *' then
14:            ingin_diketahui[counter_yang_ingin_diketahui] = i
15:            counter_yang_ingin_diketahui + 1
16:        end if
17:    end for
```

Gambar 3.2 Gambar Fungsi SOLVE (1)

```
18:  m = min(m, panjang message)
19:  for n = m/2 + 1 to n ≤ m;n+ = 1 do
20:      if VALIDITY(Key, counter_diketahui, diketahui, Selisih_diketahui, n) = True
     then
21:          counter ← 0
22:          while counter ≠ sizeof(ingin_diketahui) do
23:              if Key[ingin_diketahui[counter]%n] = null then
24:                  message[ingin_diketahui[counter]] =' *'
25:                  remove element index i in ingin_diketahui
26:              else if message[ingin_diketahui[counter]] =' *' then
27:                  message[ingin_diketahui[counter]] = (ciphertext[ingin_diketahui[counter]]−
     Key[ingin_diketahui[counter] + 26)%26
28:                  counter = counter + 1
29:              else if message[ingin_diketahui[counter]] ≠
     (ciphertext[ingin_diketahui[counter]] − Key[ingin_diketahui[counter] + 26) %26) then
30:                  message[ingin_diketahui[counter]] =' *'
31:                  remove element index i in ingin_diketahui
32:              else
33:                  counter = counter + 1
34:              end if
35:          end while
36:      end if
37:  end for
38:  puts(message)
39:  end function
```

Gambar 3.3 Gambar Fungsi SOLVE (2)

Tabel 3.4 Penjelasan variabel yang digunakan dalam fungsi SOLVE

Nama Variabel	Penjelasan
m	Digunakan untuk menerima dan menyimpan masukan batas atas panjang kunci dari fungsi MAIN.
message	Digunakan untuk menerima dan menyimpan masukan *plaintext* dari fungsi MAIN.
cipher	Digunakan untuk menerima dan menyimpan masukan *ciphertext* dari fungsi MAIN.
counter_diketahui	Digunakan untuk menghitung indeks jumlah *plaintext* dan *ciphertext* yang tidak ”*”.
counter yang ingin_diketahui	Digunakan untuk menghitung jumlah indeks *ciphertext* tidak bernilai ”*” dan *plaintext* bernilai ”*”.
diketahui	Merupakan suatu *array* yang digunakan untuk menyimpan indeks di mana *plaintext* dan *ciphertext* yang tidak ”*”.
Selisih_diketahui	Merupakan suatu *array* yang digunakan untuk menyimpan selisih antara *plaintext* dan *ciphertext* di mana *plaintext* dan *ciphertext* pada indeks tersebut tidak bernilai ”*”.
ingin_diketahui	Merupakan suatu *array* yang digunakan untuk menyimpan indeks di mana *ciphertext* tidak bernilai ”*” dan *plaintext* bernilai ”*” pada indeks tersebut.

Key	Merupakan suatu *array* yang digunakan untuk menyimpan hasil yang diperoleh dari fungsi VALIDITY

3.2.2 Desain Fungsi VALIDITY

Fungsi ini digunakan untuk memvalidasi suatu panjang kunci yang sekarang dicek kebenarannya. Gambar mengenai fungsi VALIDITY dapat dilihat pada gambar 3.4. Penjelasan mengenai fungsi ini terdapat pada subbab 2.4.

Tabel 3.5 Masukan, Proses, dan Keluaran fungsi MAIN

Masukan	Proses	Keluaran
Key, counter_diketahui, diketahui, Selisih_diketahui, n dari fungsi SOLVE	Mencari apakah suatu panjang kunci dapat digunakan atau tidak. Merupakan bagian dari modifikasi dari algoritma *Kasiski Examination*.	Berupa kunci yang telah dibentuk dan nilai benar atau salah

```
1: function VALIDITY(Key,counter_diketahui,diketahui,Selisih_diketahui,n)
2:     Initialize(Key, −1)
3:     for i = 0 to i < counter_diketahui; i+=1 do
4:         temp = diketahui[i]
5:         if Key[temp%n] = −1 then
6:             Key[temp%n] = Selisih_diketahui[temp]
7:         else if Key[temp%n] ≠ Selisih_diketahui[temp] then return False
8:         end if
9:     end for
10:    return True
11: end function
```

Gambar 3.4 Gambar Fungsi VALIDITY

Tabel 3.6 Penjelasan variabel yang digunakan dalam fungsi VALIDITY

Nama Variabel	Penjelasan
Key	Merupakan suatu *array* yang digunakan untuk menyimpan blok kunci yang telah dibentuk.
counter_diketahui	Digunakan untuk menerima dan menyimpan masukan counter_diketahui dari fungsi SOLVE.
diketahui	Merupakan suatu *array* yang digunakan untuk menerima dan menyimpan masukan variabel diketahui pada fungsi SOLVE.
Selisih_diketahui	Merupakan suatu *array* yang digunakan untuk menerima dan menyimpan Selisih_diketahui dari fungsi SOLVE.
n	Digunakan untuk menerima dan menyimpan posisi iterasi saat ini dari fungsi SOLVE.

BAB IV

IMPLEMENTASI

Pada bab ini menjelaskan implementasi yang sesuai dengan desain algoritma yang telah ditentukan sebelumnya.

4.1 Lingkungan Implementasi

Lingkungan uji coba yang digunakan adalah sebagai berikut:

1. Perangkat Keras
    - *Processor* Intel(R) Core(TM)i7-5700 @ 2.7GHz.
    - Memori 8 GB
2. Perangkat Lunak
    - Sistem Operasi Windows 10 Home 64 bit
    - *Text editor* Bloodshed Dev-C++ 5.11.
    - *Compiler* g++ (TDM-GCC 4.9.2 32-bit).

4.2 Rancangan Data

Pada subbab ini dijelaskan mengenai desain data masukan yang diperlukan untuk melakukan proses algoritma, dan data keluaran yang dihasilkan oleh program.

4.2.1 Data Masukan

Data masukan adalah data yang akan diproses oleh program sebagai masukan menggunakan algoritma yang telah dirancang dalam tugas akhir ini.

Data masukan berupa berkas teks yang berisi data dengan format yang telah ditentukan pada deskripsi *The Bytelandian Cryptographer (Act IV)*. Pada masing-masing berkas data masukan, baris pertama berupa sebuah bilangan bulat yang merepresentasikan jumlah kasus uji yang ada pada berkas tersebut. Untuk setiap kasus uji, baris pertama berupa sebuah bilangan bulat yang merepresentasikan batas atas dari kunci.

baris kedua berupa *string* yang merepresentasikan *plaintext* dan baris ketiga berupa *string* yang merepresentasikan *ciphertext* .

4.2.2 Data Keluaran

Data keluaran yang dihasilkan oleh program hanya berupa satu kalimat yang berisikan *plaintext* yang bisa didapatkan dari *ciphertext* dan batas atas panjang kunci yang telah diberikan.

4.3 Implementasi Algoritma dan Struktur Data

Pada subbab ini akan dijelaskan tentang implementasi proses algoritma secara keseluruhan berdasarkan desain yang telah dijelaskan pada bab III. Pada bagian ini menggunakan kode untuk mengoptimasi kompiler yang bertujuan untuk mempersingkat waktu eksekusi, seperti ”inline” dan ”noexcept”. Inline berguna untuk membuat baris kode dalam kompiler menjadi berurutan, karena bisa saja baris kode yang terjadi pada kompiler tidak berurutan. Noexcept adalah membuang *exception* apabila terjadinya *exception*.

4.3.1 Struktur Data yang Digunakan

Pada Implementasi algoritma dibutuhkan struktur data *unordered_map*. *Unordered _map* adalah sebuah wadah assosiatif yang berisi pasangan kunci dan nilai dengan kunci yang unik untuk setiap pasangannya. Operasi yang dapat dilakukan seperti pencarian, penyisipan, dan pemindahan atau pembuangan elemen. Struktur data ini merupakan struktur data bawaan yang terdapat dalam C++. Tujuan struktur data ini digunakan dalam tugas akhir ini adalah pencarian elemen yang memiliki kompleksitas waktu konstan dan penggunaan *dynamic memory allocation* yang dapat menekan penggunaan memori.

4.3.2 *Header* yang Diperlukan

Implementasi algoritma dengan teknik *Kasiski Examination* untuk menyelesaikan *The Bytelandian Cryptographer (Act IV)* untuk membutuhkan 4 *header* yaitu cstdio, cstring, algorithm, dan unordered_map . Seperti yang terdapat pada Kode Sumber 4.1.

```
1 #include <cstdio>
2 #include <cstring>
3 #include <unordered_map>
4 #include <algorithm>
```

Kode Sumber 4.1 *Header* yang diperlukan

Header cstdio berisi modul untuk menerima masukan dan memberikan keluaran. *Header* algorithm berisi modul yang memiliki fungsi-fungsi yang sangat berguna dalam membantu mengimplementasi algortima yang telah dibangun. Contohnya adalah fungsi max dan min. *Header* cstring berisi modul yang memiliki fungsi-fungsi untuk melakukan pemrosesan string. Contoh fungsi yang membantu mengimplementasikan algoritma yang dibangun adalah fungsi memset. *Header* unordered_map berisi modul-modul untuk membuat suatu tempat penyimpanan data yang dapat diisi, dihapus, dipindah, dan dicari untuk setiap elemennya, tetapi hanya dapat menyimpan data dalam bentuk seperti array 1 dimensi, akan tetapi media penyimpanannya seperti memetakan suatu elemen himpunan ke dalam elemen lainnya.

4.3.3 *Preprocessor Directives*

Preprocessor directives digunakan untuk memudahkan dalam menyingkat kode-kode yang akan dibuat dan biasanya berupa fungsi ataupun suatu konstanta yang akan digunakan dalam

proses perhitungan, yang nantinya akan diterjemahkan terlebih dahulu sebelum mengeksekusi kode. Kode sumber implementasi konstanta variabel dapat dilihat pada Kode Sumber 4.2.

```
1 #define mp make_pair
2 #define ins insert
3 #define MAX (int)(1e6)+1
4 #define MAXK (int)(1e5)+1
```

Kode Sumber 4.2 Preprocessor Directives

4.3.4 Variabel Global

Variabel global digunakan untuk memudahkan dalam mengakses data yang digunakan lintas fungsi. Kode sumber implementasi variabel global dapat dilihat pada Kode Sumber 4.3.

```
1 char plaintext[MAX], ciphertext[MAX];
2 int key[MAXK], both[MAX], known[MAX], knownall;
3 unordered_map<int, int> tf;
```

Kode Sumber 4.3 Variabel Global

Tabel 4.1 Penjelasan Variabel yang digunakan dalam variabel global

No	Nama Variabel	Tipe Data	Penjelasan
1	plaintext	*array of character*	Digunakan untuk menerima dan menyimpan masukan *plaintext*.
2	ciphertext	*array of character*	Digunakan untuk menyimpan dan menerima masukan *ciphertext*.
3	key	*array of integer*	Digunakan untuk menyimpan kunci yang telah *generate*.
4	both	*array of integer*	Digunakan untuk menyimpan selisih antara *plaintext* dan *ciphertext* di mana *plaintext* dan *ciphertext* pada indeks tersebut tidak bernilai ”*”.
5	known	*array of integer*	Digunakan untuk menyimpan indeks di mana pada indeks tersebut *plaintext* dan *ciphertext* yang tidak bernilai ”*”.
6	knownall	int	Digunakan untuk menghitung jumlah indeks *plaintext* dan *ciphertext* yang tidak ”*”.

7	tf	*unordered_map of integer to integer*	Digunakan untuk menyimpan indeks di mana pada indeks tersebut *ciphertext* tidak bernilai "*" dan *plaintext* bernilai "*".

4.3.5 Implementasi Fungsi Main

Fungsi Main adalah implementasi algoritma yang dirancang pada Gambar 3.1. Implementasi fungsi Main dapat dilihat pada Kode Sumber 4.4.

```
1  int main()noexcept{
2      int tc;
3      scanf("%d",&tc);
4      while(tc--){
5          int m;
6          scanf("%d",&m);
7          scanf("%s %s", plaintext,ciphertext);
8          tf.clear();
9          knownall=0;
10         SOLVE(m);
11     }
```

Kode Sumber 4.4 Fungsi main

Tabel 4.2 Penjelasan Variabel yang digunakan dalam fungsi MAIN

No	Nama Variabel	Tipe Data	Penjelasan
1	tc	integer	Digunakan untuk menerima dan menyimpan masukan kasus uji coba.

2	m	integer	Digunakan untuk menerima dan menyimpan masukan batas atas panjang kunci dari setiap kasus uji coba.

4.3.6 Implementasi Fungsi SOLVE

Fungsi SOLVE adalah implementasi algoritma yang dirancang pada Gambar 3.2 dan Gambar 3.3. Implementasi fungsi SOLVE dapat dilihat pada Kode Sumber 4.5 dan 4.6.

```
1  inline void SOLVE ( int m ) noexcept
2  {
3      int ntofind =    0;
4      for(int i=0; plaintext [ i ]!=0;  i ++)
5          if(plaintext[i]!='*'&&ciphertext[i]!='*'){
6                  known [ knownall ++]= i ;
7                  both [ i ]=(( ciphertext [ i]-plaintext [ i ]
8                  +26)%26);
9          }
10         else if(plaintext[i]=='*'&&ciphertext[i]!='*'){
11                 tf . ins ( mp ( ntofind , i ));
12                 ntofind ++;
13         }
14         unordered_map < int , int >:: iterator  it ;
15         m =  min ( m , ( int ) strlen ( plaintext ));
16         for(int n=m/2+1;n<=m;n++)
17         {
18             if(VALIDITY(n))
19             {
20                 it=tf . begin ();
```

Kode Sumber 4.5 Fungsi SOLVE

```
1                 while(it!=tf.end())
2                     if(key[(it->second)%n]==-1){
3                         plaintext[it->second]='*';
4                         it=tf.erase(it);
5                     }
6                     else if(plaintext[it->second]=='*'){
7                         plaintext[it->second]=
8                         (ciphertext[it->second]-'A'
9                     -key[(it->second)%n]+26)%26 +'A';
10                        it++;
11                    }
12                    else if(plaintext[it->second]      !=
13                    (ciphertext[it->second]-'A'
14                    -key[(it->second)%n]+26)%26 +'A'){
15                        plaintext[it->second]='*';
16                        it=tf.erase(it);
17                    }
18                    else it++;
19            }
20        }
21        printf("%s\n", plaintext);
22    }
```

Kode Sumber 4.6 Fungsi SOLVE

Tabel 4.3 Penjelasan Variabel yang digunakan dalam fungsi SOLVE

No	Nama Variabel	Tipe Data	Penjelasan
1	ntofind	integer	Digunakan untuk menghitung jumlah indeks *ciphertext* tidak bernilai "*" dan *plaintext* bernilai "*".
2	it	*unordered map of integer to integer*	Digunakan sebagai *iterator* dari variabel tf.

4.3.7 Implementasi Fungsi VALIDITY

Fungsi VALIDITY adalah implementasi algoritma yang dirancang pada Gambar 3.4. Implementasi fungsi VALIDITY dapat dilihat pada Kode Sumber 4.7.

```
1  inline bool VALIDITY ( int n ) noexcept
2  {
3      memset ( key ,   -1, sizeof(int)*n);
4      for(int x = 0; x< knownall ; x ++){
5          int temp=known[x];
6          if( key[ temp%n]== -1)   {
7              key [ temp%n ]= both [ temp ];
8              }
9          else if( key[ temp%n ]!= both[ temp ])
10             return false;
11         }
12     return true;
13 }
```

Kode Sumber 4.7 Fungsi VALIDITY

Tabel 4.4 Penjelasan Variabel yang digunakan dalam fungsi VALIDITY

No	Nama Variabel	Tipe Data	Penjelasan
1	temp	integer	Digunakan untuk menyimpan sementara nilai dari variabel known pada indeks saat dijalankannya perulangan tersebut.

(Halaman ini sengaja dikosongkan)

BAB V

UJI COBA DAN EVALUASI

Pada bab ini dijelaskan tentang uji coba dan evaluasi dari implementasi yang telah dilakukan pada tugas akhir ini.

5.1 Lingkungan Uji Coba

Linkungan uji coba yang digunakan adalah salah satu sistem yang digunakan situs penilaian daring SPOJ, yaitu kluster *Cube* dengan spesifikasi sebagai berikut:

1. Perangkat Keras:
    - *Processor* Intel(R) Pentium G860 CPU @ 3GHz.
    - *Memory* 1536 MB.
2. Perangkat Lunak:
    - *Compiler* CPP14.

5.2 Uji Coba Kebenaran

Uji coba kebenaran dilakukan dengan mengirimkan kode sumber program ke dalam situs penilaian daring SPOJ dan melakukan hasil uji coba kasus sederhana dengan langkah-langkah sesuai dengan algoritma yang telah dirancang dengan keluaran sistem. Permasalahan yang diselesaikan adalah *The Bytelandian Cryptographer (Act IV)*. Hasil uji coba dengan waktu terbaik pada situs SPOJ ditunjukkan pada Gambar A.1.

Selain itu, dilakukan pengujian sebanyak 30 kali pada situs penilaian daring SPOJ untuk melihat variasi waktu dan memori yang dibutuhkan program. Hasil uji coba sebanyak 30 kali dapat dilihat pada Gambar A.3, A.4.

Dari hasil uji coba pada Gambar A.3 dan A.4, dapat disederhanakan menjadi Gambar A.2. Dari informasi yang terdapat pada Gambar A.2 dapat ditarik beberapa informasi seperti yang tertera pada Tabel 5.1.

Tabel 5.1 Kecepatan Maksimal, Minimal, dan Rata-Rata dari Hasil Uji Coba Pengumpulan 30 Kali pada Situs Pengujian Daring Spoj

Waktu Maksimal	4, 49 detik
Waktu Minimal	4, 38 detik
Waktu Rata-Rata	4.418 detik
Memori Maksimal	27 MB
Memori Minimal	26 MB
Memori Rata-Rata	26.5 MB

Berdasarkan Tabel 5.1 didapatkan waktu eksekusi rata-rata 4.418 detik dan waktu maksimal 4, 47 detik. Waktu eksekusi tersebut 3, 8 kali lebih cepat dari batas waktu eksekusi yang tertera pada deskripsi permasalahan, yaitu 17 detik.

Uji Coba dengan menggunakan contoh kasus uji coba yang tersedia di dalam SPOJ *The Bytelandian Cryptographer (Act IV)*. Sebagai contoh akan digunakan kasus ujicoba yang menggunakan baik mencari panjang kunci maupun *intersection* yang terjadi dalam permasalahan ini.

Sesuai dengan algoritma yang telah dirancang pada *pseudocode* yang terdapat pada Gambar 3.2 dan 3.3 maupun pada Gambar 3.4. Algoritma ini akan melakukan perulangan yang terdapat pada *plaintext* dan *ciphertext* yang diperoleh dari inputan dan akan menyimpan posisi karakter dengan ketentuan apabila pada indeks tersebut diketahui baik *plaintext* dan *ciphertext* beserta dengan selisih antara *ciphertext* dan *plaintext* . Selanjutnya juga menyimpan posisi karakter yang diperoleh dari *ciphertext* dengan ketentuan apabila *ciphertext* pada indeks tersebut diketahui karakternya dan *plaintext* diindeks tersebut tidak diketahui karakternya. Misalnya diambilkan contoh dari Tabel 2.10. Maka yang disimpan untuk bagian yang diketahui keduanya adalah indeks ke 1 dengan selisih 0 dan indeks ke 2 dengan selisih 0, sedangkan untuk yang disimpan pada diketahui *ciphertext* saja maka jawabannya semua indeks kecual indeks ke

1 dan 2. Selanjutnya, akan membandingkan antara panjang *plaintext* atau *ciphertext* dengan *m* untuk dicari yang lebih kecil yang mana. Selanjutnya, mulai mengiterasi panjang kunci yang akan muncul dari $\frac{m}{2}$ + 1 sampai dengan *m*. Di dalam iterasi tersebut akan dilakukan pengecekan apakah nilai *m* dengan fungsi VALIDITY(*m*). Jika gagal, program akan melanjutkan untuk mencari panjang kunci selanjutnya, sebaliknya jika hasil dari fungsi tersebut benar maka akan melanjutnya proses *generate* hasil yang telah diperoleh dari panjang kunci secara satu per satu dan dibandingan dengan hasil yang sudah ada sebelumnya. Perbandingan tersebut akan mengikuti aturan apabila suatu indeks ternyata ada yang konflik, baik yang nilainya berubah ataupun tidak memiliki suatu aturan kunci dari panjang kunci yang saat itu tersedia. Maka, hasil dari indeks tersebut adalah ”*” dan menghapus elemen dari tempat penyimpanan yang menampung indeks *plaintext* yang ”*” dan *ciphertext* yang tidak ”*”. Apabila tidak ada konflik maka *plaintext* pada indeks tersebut tidak menjadi ”*”. Seperti contohnya terdapat pada Tabel 2.4 dan Tabel 2.10

Sehingga hasil keluaran yang diperoleh dari algoritma ini adalah seluruh *plaintext* yang dapat dibentuk.

5.3 Analisa Kompleksitas Waktu

Pada *pseudocode* yang terdapat pada Gambar 3.1. Untuk setiap kasus ujicoba terdapat 2 fungsi utama. Dengan menggunakan *Kasiski Examination* dan *Intersection* yang terdapat pada fungsi SOLVE dan VALIDITY memiliki kompleksitas $O((n + (\frac{M}{2} * (N + S))))$. *n* adalah panjang karakter *plaintext* atau *ciphertext*, *M*/2 adalah batas atas kunci dibagi dengan 2, *N* adalah jumlah posisi karakter yang terdapat bisa jadi memiliki suatu nilai yang bisa didapat dari *ciphertext*, dan *S* adalah jumlah posisi karakter yang diketahui. Untuk

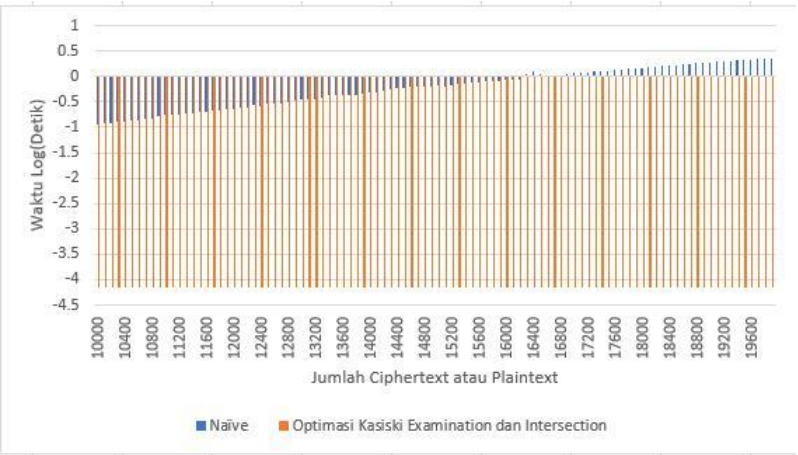
kompleksitas fungsi VALIDITY $O(S)$. Sehingga kompleksitas dapat disederhanakan menjadi $O(T * \frac{M}{2} * (N + S))$.

Sehingga secara keseluruhan kompleksitas dari algoritma yang dirancang pada tugas akhir ini adalah $O(T * \frac{M}{2} * (N + S))$.

Pada umumnya, eksekusi program pada situs penilaian daring SPOJ adalah 1 detik untuk setiap 1.000.000.000 proses. Ekseskusi program dengan kompleksitas $O(T * \frac{M}{2} * (N + S))$. Pada *worst case* $T * (N + S) = 1.000.000$, sedangkan $\frac{M}{2}$ adalah 50.000. Hasilnya melebihi dari 50 miliar perulangan, maka waktu yang dibutuhkan adalah 50 detik. Jika hal ini terjadi maka waktu eksekusi akan berjalan dengan sangat lama. Estimasi waktu kurang lebih 100 detik, tetapi kenyataan yang terjadi tidak demikian, alasannya karena jumlah S bisa berkurang seiring dengan perulangan yang ada.

Algoritma *Naive* yang digunakan adalah sama menggunakan algoritma optimasi *Kasiski Examination* dengan *Intersection*, tetapi perbedaannya terletak pada *Intersection* yang dilakukan. kompleksitas algoritma *Naive* menjadi $O(T * \frac{M}{2} * (N + S)^2)$.

Sebagai perbandingan ujicoba kinerja antara menggunakan algoritma *Naive* dan algoritma optimasi *Kasiski Examination* dengan *Intersection*, dapat dilihat pada Gambar 5.1. Data tersebut didapatkan dari Lampiran B. Dengan jumlah data 10.000 sampai dengan 19.900 log waktu yang diperlukan oleh algoritma optimasi *Kasiski Examination* dengan *Intersection* lebih rendah daripada algoritma *Naive*. Maka, dapat ditarik kesimpulan bahwa algoritma optimasi *Kasiski Examination* dengan *Intersection* jauh lebih cepat dibandingan algoritma *Naive*.



Gambar 5.1 Gambar Perbandingan Kinerja Algoritma Optimasi *Kasiski Examination* dengan *Intersection* dan *Naive*

(Halaman ini sengaja dikosongkan)

BAB VI

KESIMPULAN

Bab ini membahas kesimpulan yang dapat diambil dari tujuan pembuatan sistem dan hubungannya dengan hasil uji coba yang telah dilakukan. Selain itu, terdapat beberapa saran yang bisa dijadikan acuan untuk melakukan pengembangan dan penelitian lebih lanjut.

6.1 Kesimpulan

Dari hasil uji coba yang telah dilakukan terhadap perancangan dan implementasi algoritma untuk menyelesaikan SPOJ *The Bytelandian Cryptographer (Act IV)* dapat diambil kesimpulan sebagai berikut:

1. Implementasi algoritma dengan menggunakan teknik *Kasiski Examnination* dengan adanya optimasi saja tidak dapat menyelesaikan permasalahan SPOJ *The Bytelandian Cryptographer (Act IV)* dengan benar. Akan tetapi apabila ditambahkan metode *Intersection* di dalam teknik *Kasiski Examnination* ditambah dengan optimasi dapat menyelesaikan permasalahan SPOJ *The Bytelandian Cryptographer (Act IV)* dengan benar.
2. Kompleksitas waktu $O(T * \frac{M}{2} * (N + S))$ masih dapat menyelesaikan permasalahan SPOJ *The Bytelandian Cryptographer (Act IV)* dalam rentang waktu yang telah ditetapkan.
3. Waktu yang dibutuhkan oleh program untuk menyelesaikan SPOJ *The Bytelandian Cryptographer (Act IV)* minimum 4, 38 detik, maksimum 4, 49 detik dan rata-rata 4.418 detik. Memori yang dibutuhkan berkisar antara 26-27 MB.

6.2 Saran

Pada Tugas Akhir kali ini tentunya terdapat kekurangan serta nilai-nilai yang dapat penulis ambil. Berikut adalah saran-saran yang dapat diambil melalui Tugas Akhir ini:

1. Teknik *Kasiski Examination* dalam pencarian panjang kunci masih cenderung lambat. Hal ini terjadi karena masih menggunakan teknik *brute force*. Sehingga *running time* yang diperoleh kurang optimal. Perlu adanya optimisasi lanjutan atau penggantian metode yang dapat mencari suatu panjang kunci lebih cepat. Melihat pada Gambar A.5 dan A.6 terdapat yang program yang memiliki *running time* yang lebih cepat.

DAFTAR PUSTAKA

[1] W. Stallings and L. Brown, *Computer Security Principles and Practice*, 3rd ed. Pearson, 2015.

[2] S. William, *Crytography and Network Security*, 5th ed. Pearson, 2011.

[3] C. Henk, *Encyclopedia of Cryptography and Security*. Springer, 2005.

[4] "Kasiski Method." [Online]. Available: http://pages.mtu.edu/~shene/NSF-4/Tutorial/VIG/Vig-Kasiski.html

[5] K. Devlin, *The Joy of Sets: Fundamentals of Contemporary Set Theory*, 2nd ed. New York: Springer, 1993.

[6] K. Piwakowski, "CRYPTO4 - The Bytelandian Cryptographer (Act IV)," 2004. [Online]. Available: http://www.spoj.com/problems/CRYPTO4/

[7] john_jones, "SPOJ Discussion Board," 2009. [Online]. Available: http://discuss.spoj.com/t/problemset-3/242/13

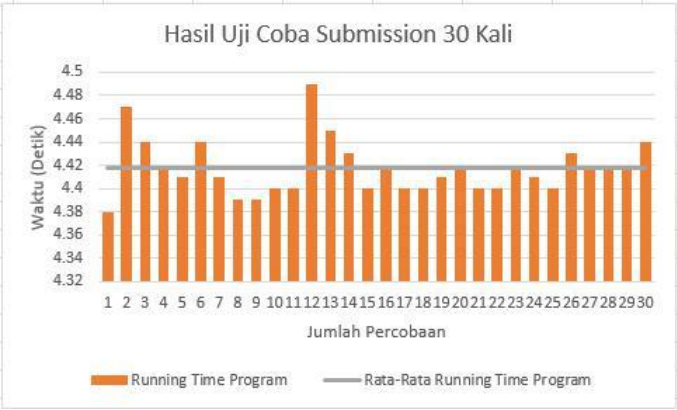
(Halaman ini sengaja dikosongkan)

BAB A

LAMPIRAN

20609690		2017-11-16 06:24:28	The Bytelandian Cryptographer (Act IV)	**accepted** edit ideone it	4.38	26M	CPP14

Gambar A.1 Hasil Uji Coba pada Situs Penilaian SPOJ



Gambar A.2 Grafik Hasil Uji Coba pada Situs SPOJ Sebanyak 30 Kali

20609690	☐	2017-11-19 06:24:26	The Bytelandian Cryptographer (Act IV)	accepted edit · ideone it	4.38	26M	CPP14
20609689	☐	2017-11-19 06:24:20	The Bytelandian Cryptographer (Act IV)	accepted edit · ideone it	4.47	27M	CPP14
20609688	☐	2017-11-19 06:24:10	The Bytelandian Cryptographer (Act IV)	accepted edit · ideone it	4.44	26M	CPP14
20609686	☐	2017-11-19 06:24:02	The Bytelandian Cryptographer (Act IV)	accepted edit · ideone it	4.42	26M	CPP14
20609684	☐	2017-11-19 06:23:52	The Bytelandian Cryptographer (Act IV)	accepted edit · ideone it	4.41	26M	CPP14
20609682	☐	2017-11-19 06:23:43	The Bytelandian Cryptographer (Act IV)	accepted edit · ideone it	4.44	26M	CPP14
20609681	☐	2017-11-19 06:23:36	The Bytelandian Cryptographer (Act IV)	accepted edit · ideone it	4.41	26M	CPP14
20609680	☐	2017-11-19 06:23:29	The Bytelandian Cryptographer (Act IV)	accepted edit · ideone it	4.39	26M	CPP14
20609676	☐	2017-11-19 06:23:20	The Bytelandian Cryptographer (Act IV)	accepted edit · ideone it	4.39	26M	CPP14
20609675	☐	2017-11-19 06:23:10	The Bytelandian Cryptographer (Act IV)	accepted edit · ideone it	4.40	27M	CPP14
20609673	☐	2017-11-19 06:23:01	The Bytelandian Cryptographer (Act IV)	accepted edit · ideone it	4.40	26M	CPP14
20609672	☐	2017-11-19 06:22:52	The Bytelandian Cryptographer (Act IV)	accepted edit · ideone it	4.49	27M	CPP14
20609671	☐	2017-11-19 06:22:43	The Bytelandian Cryptographer (Act IV)	accepted edit · ideone it	4.45	27M	CPP14
20609669	☐	2017-11-19 06:22:26	The Bytelandian Cryptographer (Act IV)	accepted edit · ideone it	4.43	27M	CPP14
20609667	☐	2017-11-19 06:22:18	The Bytelandian Cryptographer (Act IV)	accepted edit · ideone it	4.40	27M	CPP14
20609666	☐	2017-11-19 06:22:07	The Bytelandian Cryptographer (Act IV)	accepted edit · ideone it	4.42	27M	CPP14
20609665	☐	2017-11-19 06:21:56	The Bytelandian Cryptographer (Act IV)	accepted edit · ideone it	4.40	26M	CPP14
20609663	☐	2017-11-19 06:21:44	The Bytelandian Cryptographer (Act IV)	accepted edit · ideone it	4.40	26M	CPP14
20609662	☐	2017-11-19 06:21:35	The Bytelandian Cryptographer (Act IV)	accepted edit · ideone it	4.41	26M	CPP14
20609661	☐	2017-11-19 06:21:26	The Bytelandian Cryptographer (Act IV)	accepted edit · ideone it	4.42	26M	CPP14

Gambar A.3 Hasil Pengujian Sebanyak 30 Kali pada Situs Penilaian Daring SPOJ (1)

20609659	☐	2017-11-16 06:21:21	The Bytelandian Cryptographer (Act IV)	**accepted** edit ideone it	4.40	26M	CPP14
20609657	☐	2017-11-16 06:21:12	The Bytelandian Cryptographer (Act IV)	**accepted** edit ideone it	4.40	27M	CPP14
20609656	☐	2017-11-16 06:21:02	The Bytelandian Cryptographer (Act IV)	**accepted** edit ideone it	4.42	26M	CPP14
20609655	☐	2017-11-16 06:20:50	The Bytelandian Cryptographer (Act IV)	**accepted** edit ideone it	4.41	26M	CPP14
20609654	☐	2017-11-16 06:20:32	The Bytelandian Cryptographer (Act IV)	**accepted** edit ideone it	4.40	27M	CPP14
20609652	☐	2017-11-16 06:20:11	The Bytelandian Cryptographer (Act IV)	**accepted** edit ideone it	4.43	26M	CPP14
20609651	☐	2017-11-16 06:20:00	The Bytelandian Cryptographer (Act IV)	**accepted** edit ideone it	4.42	27M	CPP14
20609650	☐	2017-11-16 06:19:35	The Bytelandian Cryptographer (Act IV)	**accepted** edit ideone it	4.42	26M	CPP14
20603932	☐	2017-11-15 12:19:55	The Bytelandian Cryptographer (Act IV)	**accepted** edit ideone it	4.42	27M	CPP14
20603856	☐	2017-11-15 12:00:01	The Bytelandian Cryptographer (Act IV)	**accepted** edit ideone it	4.44	27M	CPP14

Gambar A.4 Hasil Pengujian Sebanyak 30 Kali pada Situs Penilaian Daring SPOJ (2)

RANK	DATE	USER	RESULT	TIME	MEM	LANG
1	2014-07-09 20:26:13	sidharth jain	**accepted**	0.21	18M	C++ 4.3.2
2	2011-02-04 10:11:14	Josef K.	**accepted**	0.32	9.0M	CPP
3	2010-02-25 22:16:52	Carlos Eduardo Rodrigues Alves [USJT]	**accepted**	0.33	31M	CPP
4	2011-06-04 01:25:06	NiHaobin	**accepted**	0.34	151M	CPP
5	2009-12-29 16:48:15	[Rampage] Blue.Mary	**accepted**	0.39	28M	C++ 4.3.2
6	2010-05-03 01:07:25	Carlos Eduardo Rodrigues Alves [USJT]	**accepted**	0.39	30M	C++ 4.3.2
7	2011-02-04 10:12:30	Josef K.	**accepted**	0.39	8.4M	C++ 4.3.2
8	2004-12-08 18:13:03	Jakub Łopuszański	**accepted**	0.42	14M	CPP
9	2009-03-07 01:11:57	Robert Gerbicz	**accepted**	0.42	33M	CPP
10	2010-04-23 11:23:55	Oleg	**accepted**	0.44	7.5M	C++ 4.3.2
11	2004-11-27 07:22:27	Tomek Czajka	**accepted**	0.53	4.3M	CPP
12	2007-05-31 03:54:03	Huacheng Yu	**accepted**	0.60	17M	C
13	2010-07-02 12:47:04	刘启鹏	**accepted**	0.61	17M	C
14	2010-12-18 14:59:02	sevenkplus	**accepted**	0.70	21M	C++ 4.3.2
15	2010-12-18 14:55:16	sevenkplus	**accepted**	0.77	21M	CPP
16	2011-03-05 23:42:24	Alexander Pivovarov	**accepted**	0.77	118M	CPP
17	2013-08-02 09:56:37	Tomasz Stanislawski	**accepted**	0.77	14M	C99
18	2011-06-06 13:20:16	blashyrkh	**accepted**	0.80	23M	C
19	2007-05-31 01:58:27	FG	**accepted**	0.95	2.7M	PAS-FPC
20	2008-03-19 03:57:13	zhengxi	**accepted**	0.98	52M	CPP

Gambar A.5 Daftar Peringkat Berdasarkan Kecepatan Eksekusi Program yang Diperoleh dari Dari SPOJ(1)

RANK	DATE	USER	RESULT	TIME	MEM	LANG
21	2009-06-22 16:57:33	QiZiChao	**accepted**	0.98	13M	CPP
22	2007-09-08 22:37:16	Sandeep Kumar	**accepted**	1.24	17M	CPP
23	2006-08-31 22:54:06	Tijs van Bakel	**accepted**	1.32	59M	CPP
24	2007-02-04 18:33:29	James Cook	**accepted**	1.52	10M	C
25	2010-09-14 13:48:55	Laxminarayana	**accepted**	1.57	12M	C++ 4.3.2
26	2013-02-05 03:27:04	roginn	**accepted**	1.60	3.9M	C++ 4.3.2
27	2009-11-01 21:00:00	anonymous	**accepted**	1.64	3.1M	C++ 4.3.2
28	2011-08-06 21:38:41	Peutri	**accepted**	1.91	2.8M	C++ 4.3.2
29	2004-11-18 17:26:14	Pascal Zimmer	**accepted**	1.94	7.2M	C
30	2005-02-03 10:06:50	Tim Green @ Ark	**accepted**	1.94	13M	CPP
31	2015-09-03 20:00:03	vijaygbvv	**accepted**	2.36	5.6M	C++ 4.3.2
32	2011-08-23 16:06:32	Darren Izzard	**accepted**	2.48	3.1M	C++ 4.3.2
33	2017-11-15 11:46:27	freddy	**accepted**	4.38	27M	CPP14

Gambar A.6 Daftar Hasil Peringkat Berdasarkan Kecepatan Eksekusi Program yang Diperoleh dari Dari SPOJ(2)

(Halaman ini sengaja dikosongkan)

BAB B

HASIL PERCOBAAN DENGAN MENGGUNAKAN ALGORITMA NAIVE DAN KASISKI EXAMINATION

Tabel B.1 Hasil Percobaan Penyelesaian Studi Kasus SPOJ The Bytelandian Cryptographer(Act IV) dengan menggunakan algoritma optimasi *Kasiski Examination* dan *Intersection* (1)

N	Waktu(Detik)	Waktu(log Detik)
10000	0.0156259	-1.80615
10100	0.0156259	-1.80615
10200	0.0156244	-1.8062
10300	0.0156244	-1.8062
10400	0.0156244	-1.8062
10500	0.0156255	-1.80617
10600	0.0156255	-1.80617
10700	0.0156255	-1.80617
10800	0.0156255	-1.80617
10900	0.0156255	-1.80617
11000	0.0156248	-1.80619
11100	0.0156351	-1.8059
11200	0.0156351	-1.8059
11300	0.0156351	-1.8059
11400	0.0156251	-1.80618
11500	0.0156251	-1.80618
11600	0.0156259	-1.80615
11700	0.0156259	-1.80615
11800	0.0156259	-1.80615
11900	0.0156259	-1.80615

Tabel B.2 Hasil Percobaan Penyelesaian Studi Kasus SPOJ The Bytelandian Cryptographer(Act IV) dengan menggunakan algoritma optimasi *Kasiski Examination* dan *Intersection* (2)

N	Waktu(Detik)	Waktu(log Detik)
12000	0.0156252	-1.80617
12100	0.0156252	-1.80617
12200	0.001048	-2.97964
12300	0.0156267	-1.80613
12400	0.0156267	-1.80613
12500	0.0156267	-1.80613
12600	0.0156237	-1.80622
12700	0.0156237	-1.80622
12800	0.0156244	-1.8062
12900	0.0156244	-1.8062
13000	0.0156244	-1.8062
13100	0.0156244	-1.8062
13200	0.0156244	-1.8062
13300	0.0156244	-1.8062
13400	0.0156229	-1.80624
13500	0.0156251	-1.80618
13600	0.0156441	-1.80565
13700	0.0156351	-1.8059
13800	0.0156351	-1.8059
13900	0.0156351	-1.8059

Tabel B.3 Hasil Percobaan Penyelesaian Studi Kasus SPOJ The Bytelandian Cryptographer(Act IV) dengan menggunakan algoritma optimasi *Kasiski Examination* dan *Intersection* (3)

N	Waktu(Detik)	Waktu(log Detik)
14000	0.0156149	-1.80646
14100	0.0156252	-1.80617
14200	0.0156252	-1.80617
14300	0.0156252	-1.80617
14400	0.0156343	-1.80592
14500	0.0156343	-1.80592
14600	0.0156343	-1.80592
14700	0.0156343	-1.80592
14800	0.0156229	-1.80624
14900	0.0156229	-1.80624
15000	0.0156229	-1.80624
15100	0.0156229	-1.80624
15200	0.0156229	-1.80624
15300	0.0156229	-1.80624
15400	0.0156354	-1.80589
15500	0.0156263	-1.80614
15600	0.0156263	-1.80614
15700	0.0156263	-1.80614
15800	0.0156263	-1.80614
15900	0.0156278	-1.8061

Tabel B.4 Hasil Percobaan Penyelesaian Studi Kasus SPOJ The Bytelandian Cryptographer(Act IV) dengan menggunakan algoritma optimasi *Kasiski Examination* dan *Intersection* (4)

N	Waktu(Detik)	Waktu(log Detik)
16000	0.0156256	-1.80616
16100	0.0156256	-1.80616
16200	0.0156256	-1.80616
16300	0.0156351	-1.8059
16400	0.0156244	-1.8062
16500	0.0156278	-1.8061
16600	0.0156278	-1.8061
16700	0.0156278	-1.8061
16800	0.0227098	-1.64379
16900	0.0156351	-1.8059
17000	0.0156153	-1.80645
17100	0.0156153	-1.80645
17200	0.0156153	-1.80645
17300	0.015635	-1.8059
17400	0.0156259	-1.80615
17500	0.0156347	-1.80591
17600	0.0156153	-1.80645
17700	0.0156153	-1.80645
17800	0.0156191	-1.80634
17900	0.0156247	-1.80619

Tabel B.5 Hasil Percobaan Penyelesaian Studi Kasus SPOJ The Bytelandian Cryptographer(Act IV) dengan menggunakan algoritma optimasi *Kasiski Examination* dan *Intersection* (5)

N	Waktu(Detik)	Waktu(log Detik)
18000	0.0156247	-1.80619
18100	0.0156247	-1.80619
18200	0.0156247	-1.80619
18300	0.0156229	-1.80624
18400	0.0156255	-1.80617
18500	0.0156247	-1.80619
18600	0.0156225	-1.80625
18700	0.0156225	-1.80625
18800	0.0156225	-1.80625
18900	0.0156225	-1.80625
19000	0.0156248	-1.80619
19100	0.0156275	-1.80611
19200	0.0156252	-1.80617
19300	0.0156206	-1.8063
19400	0.015626	-1.80615
19500	0.015626	-1.80615
19600	0.015626	-1.80615
19700	0.015626	-1.80615
19800	0.015626	-1.80615
19900	0.0156248	-1.80619

Tabel B.6 Hasil Percobaan Penyelesaian Studi Kasus SPOJ The Bytelandian Cryptographer(Act IV) dengan menggunakan algoritma *Naive* (1)

N	Waktu(Detik)	Waktu(Log Detik)
10000	0.390619	-0.408247
10100	0.394834	-0.403585
10200	0.400778	-0.397096
10300	0.406246	-0.391211
10400	0.406737	-0.390686
10500	0.42187	-0.374821
10600	0.421361	-0.375346
10700	0.437496	-0.359026
10800	0.437661	-0.358862
10900	0.45313	-0.343777
11000	0.468254	-0.329519
11100	0.468746	-0.329062
11200	0.469241	-0.328604
11300	0.48438	-0.314814
11400	0.48388	-0.315262
11500	0.498918	-0.301971
11600	0.501698	-0.299558
11700	0.515621	-0.287669
11800	0.516122	-0.287248
11900	0.531256	-0.274696

Tabel B.7 Hasil Percobaan Penyelesaian Studi Kasus SPOJ The Bytelandian Cryptographer(Act IV) dengan menggunakan algoritma *Naive* (2)

N	Waktu(Detik)	Waktu(Log Detik)
12000	0.530714	-0.275139
12100	0.546407	-0.262484
12200	0.54688	-0.262108
12300	0.578888	-0.237405
12400	0.562516	-0.249865
12500	0.593242	-0.226768
12600	0.594233	-0.226043
12700	0.593746	-0.226399
12800	0.608855	-0.215486
12900	0.621452	-0.206592
13000	0.640633	-0.193391
13100	0.640096	-0.193755
13200	0.641099	-0.193075
13300	0.656247	-0.182933
13400	0.687057	-0.163007
13500	0.687981	-0.162424
13600	0.687508	-0.162722
13700	0.702599	-0.153292
13800	0.703614	-0.152666
13900	0.718225	-0.143739

Tabel B.8 Hasil Percobaan Penyelesaian Studi Kasus SPOJ The Bytelandian Cryptographer(Act IV) dengan menggunakan algoritma *Naive* (3)

N	Waktu(Detik)	Waktu(Log Detik)
14000	0.734383	-0.134077
14100	0.734832	-0.133812
14200	0.749211	-0.125396
14300	0.766334	-0.115582
14400	0.789337	-0.102738
14500	0.796879	-0.0986076
14600	0.81308	-0.0898667
14700	0.812038	-0.0904236
14800	0.813018	-0.0898998
14900	0.827625	-0.0821664
15000	0.844241	-0.0735336
15100	0.827581	-0.0821895
15200	0.843761	-0.0737806
15300	0.859857	-0.0655738
15400	0.874509	-0.0582357
15500	0.891106	-0.0500706
15600	0.890121	-0.050551
15700	0.906285	-0.0427352
15800	0.921589	-0.0354627
15900	0.922729	-0.0349258

Tabel B.9 Hasil Percobaan Penyelesaian Studi Kasus SPOJ The Bytelandian Cryptographer(Act IV) dengan menggunakan algoritma *Naive* (4)

N	Waktu(Detik)	Waktu(Log Detik)
16000	0.937003	-0.028259
16100	0.952834	-0.0209828
16200	0.953624	-0.0206228
16300	1.04645	0.0197185
16400	1.10416	0.043032
16500	1.0375	0.0159881
16600	1.03005	0.0128583
16700	1.03179	0.0135913
16800	1.03075	0.0131533
16900	1.04765	0.0202162
17000	1.07823	0.0327114
17100	1.09324	0.0387155
17200	1.09423	0.0391086
17300	1.10882	0.0448611
17400	1.10902	0.0449394
17500	1.12558	0.0513764
17600	1.14016	0.0569658
17700	1.15626	0.0630555
17800	1.17237	0.0690647
17900	1.18701	0.0744544

Tabel B.10 Hasil Percobaan Penyelesaian Studi Kasus SPOJ The Bytelandian Cryptographer(Act IV) dengan menggunakan algoritma *Naive* (5)

N	Waktu(Detik)	Waktu(Log Detik)
18000	1.18803	0.0748274
18100	1.20266	0.0801429
18200	1.21944	0.0861604
18300	1.23395	0.0912976
18400	1.24748	0.0960336
18500	1.25243	0.0977535
18600	1.26614	0.102482
18700	1.28063	0.107424
18800	1.2969	0.112906
18900	1.31404	0.118609
19000	1.32662	0.122747
19100	1.35917	0.133274
19200	1.34388	0.12836
19300	1.36163	0.134059
19400	1.37299	0.137667
19500	1.39379	0.144197
19600	1.40303	0.147067
19700	1.43746	0.157596
19800	1.43739	0.157575
19900	1.43803	0.157768

BIODATA PENULIS

Freddy Hermawan Yuwono, kelahiran dan besar di Bondowoso-Jawa Timur, sangat suka membaca. Penulis menempuh jenjang pendidikan S1 Teknik Informatika ITS dari tahun 2013 sampai dengan dibuatnya buku ini.

Motto penulis yaitu ”Segala sesuatu pasti akan terjadi dan pasti akan dilewati”, membawa penulis mencoba belajar yang baru topik tugas akhir ini, di mana penulis dapat menerapkan sesuatu yang belum pernah penulis untuk melaluinya. Algoritma, optimasi dan pelajaran yang penulis petik dari yang pernah dilakukan oleh penulis sebelumnya, dengan bimbingan dosen-dosen pembimbing. Dalam pendalaman topik tugas akhir ini juga, penulis banyak belajar dan menjadi sangat tertarik mendalami *cryptography*, dan *data scientist*.

Dengan segala kerendahan hati, ilmu penulis masihlah setitik dibandingkan susu sebelanga. Penulis sangat mengharapkan diskusi, ajaran dan bantuan dalam memperbaiki diri. Apabila pembaca berkenan, penulis dapat dihubungi melalui *email* ke `freddy.yuwono@gmail.com`.