Vigenere Minimum-Prime Key-Adding Cipher

Zakka Fauzan Muhammad [1)]
1) Teknik Informatika ITB, Bandung, email: if14020@students.if.itb.ac.id

Abstraksi – *Akhir-akhir ini, keamanan data dan berkas yang dikirimkan dari satu pihak ke pihak lain merupakan suatu permasalahan yang cukup penting. Hal ini terutama dalam pengiriman berkas rahasia. Oleh karena itu, digunakanlah kriptografi, sebagai teknik penyandian dari suatu berkas, agar berkas sulit dibaca oleh pihak yang tidak berwenang. Kriptografi dengan menggunakan teknik vigenere cipher dirasakan kurang aman, karena pemecahan terhadap algoritma tersebut sangatlah mudah dilakukan. Sedangkan one-time pad, meskipun sulit dipecahkan, akan tetapi juga sulit untuk pemberian data kuncinya. Oleh karena itu, pada makalah ini akan dibahas mengenai teknik baru, yang memanfaatkan kesederhanaan dari vigenere cipher dan kerumitan pemecahan masalah pada one-time pad, yaitu key yang berbeda untuk setiap karakternya. Algoritma baru ini diberi nama Vigenere Minimum-Prime Key-Adding Cipher. Sesuai dengan namanya, pada algoritma ini, dimanfaatkan bilangan-bilangan prima untuk menghasilkan kunci yang selalu berubah-ubah di setiap karakternya.*

Kata Kunci*: kriptografi, bilangan prima, vigenere cipher*

1. PENDAHULUAN

Dari berbagai macam algoritma kriptografi, dikenal dua metode yang secara garis besar agak bertentangan. *Vigenere Cipher* merupakan metode yang cukup mengurangi frekuensi kemunculan karakter di dalam pesan yang dienkripsi, akan tetapi didapatkan kekurangan yang cukup nyata. Kelemahan *vigenere cipher* adalah cukup mudah dipecahkan akibat adanya perulangan kunci yang digunakan di setiap periodenya, periode di sini adalah panjang kunci. Di sisi lain, ada pula metode *one-time pad*, dengan kunci yang dihasilkan akan benar-benar acak, akan tetapi metode ini pun memiliki kelemahan, karena pemberitahuan kunci akan menjadi sebuah masalah lainnya.

Untuk mengatasi kedua masalah tersebut, diciptakan sebuah metode yang memanfaatkan kesederhanaan dari *vigenere cipher* dan memanfaatkan kesulitan pemecahan data terenkripsi dari *one-time pad*, untuk menambah kerumitan, diciptakanlah algoritma *vigenere minimum-prime key-adding cipher*.

2. IMPLEMENTASI ALGORITMA

Pada dasarnya metode ini tidak jauh berbeda dengan metode *vigenere cipher*, ada pengulangan kunci yang dipakai jika kunci lebih pendek dari teks yang nantinya akan disampaikan. Langkah-langkah enkripsi pesan adalah sebagai berikut (asumsikan panjang kunci adalah n karakter):

a. Karakter ke-1 sampai ke-n dienkripsi dengan kunci yang normal (tanpa perubahan kunci)
b. Karakter ke-(n+1) dienkripsi dengan kunci yang ditambahkan dengan bilangan prima terkecil yang tidak kurang dari penjumlahan setiap karakter kuncinya, dengan a = 0, b = 1, …, z = 25
c. Karakter ke-(n+2) sampai ke-(2n) dienkripsi dengan kunci yang ditambahkan dengan bilangan prima terkecil setelah bilangan prima yang merupakan penambah pada karakter sebelumnya
d. Untuk k > 1, karakter ke-(kn+1) sampai ke-((k+1)n-1) dienkripsi dengan kunci yang ditambahkan dengan bilangan prima yang sama dengan penambah pada karakter ke-((k-1)n+2) sampai ke-(kn)
e. Untuk k > 2, karakter ke-(kn) dienkripsi dengan kunci yang ditambahkan dengan bilangan prima terkecil setelah bilangan prima yang merupakan penambah pada karakter sebelumnya (karakter ke-[kn-1])

Sedangkan untuk proses dekripsi pesan, tidak ada perbedaan mencolok. Perbedaan yang ada hanyalah jika untuk menghasilkan pesan terenkripsi, teks akan ditambahkan dengan penambah, untuk dekripsi pesan, teks (cipherteks) akan dikurangi dengan bilangan prima yang sama dengan penambah. Untuk lebih jelasnya mengenai teknik kriptografi yang digunakan, akan dibahas pada bab 3, yaitu mengenai hasil implementasi dengan contoh nyata yang ada.

Secara garis besar, algoritmanya adalah sebagai berikut (dalam bentuk *pseudo-code*):

```
x ← jumlah karakter kunci

y ← bilangan prima terkecil yang tidak
lebih kecil dari x

n ← panjang karakter kunci

m ← panjang pesan

enkripsi pesan pada karakter 1 sampai
karakter n dengan vigenere cipher biasa

for i←n+1 to 2n
       enkripsi karakter i dengan kunci
ditambah dengan y
       y ← bilangan prima terkecil setelah
y
endfor

for i←2n+1 to m
       if (i habis dibagi m)
              enkripsi    karakter    ke-i
dengan kunci ditambah dengan y
       else
              enkripsi    karakter    ke-i
dengan kunci ditambah dengan penambah yang
sama pada karakter ke i-n+1
       endif

endfor
```

Bilangan prima sendiri dipilih karena sampai saat ini, membangkitkan bilangan prima ke-n dengan rumus umum belum ditemukan, sehingga jika nantinya kriptanalisis ingin mencoba dengan segala kemungkinan bilangan prima penambah, ia akan mengalami sedikit kesulitan membangkitkan bilangan prima. Sampai saat ini, penulis merasa bilangan prima merupakan bilangan yang sangat sulit dibangkitkan dengan cepat.

Selain dari hal-hal tersebut, terdapat batasan implementasi dari algoritma ini, yaitu enkripsi tidak akan dilakukan untuk simbol selain karakter ‘a’-‘z’ dan ‘A’-‘Z’, dan juga, untuk mempersulit kriptanalisi melakukan kriptanalisis, setiap karakter akan diubah menjadi huruf kecil serta spasi akan dihilangkan dari teks yang dienkripsi. Selain itu, sebagai batasan lainnya, setiap karakter dalam kunci harus berupa huruf kecil.

Algoritma ini, sama seperti vigenere cipher, juga bersifat berputar, artinya karakter setelah karakter ‘z’ dianggap karakter ‘a’.

Andaikan panjang dari teks yang akan dienkripsi (dengan mengabaikan simbol-simbol selain ‘a’-‘z’ dan ‘A’-‘Z’) adalah N karakter dan panjang dari kunci yang dimiliki adalah M karakter, maka jumlah bilangan prima yang harus dibangkitkan adalah sebanyak berikut:

Andai N <= M, maka tidak ada bilangan prima yang perlu dihasilkan, kompleksitas dari algoritma ini hanyalah perhitungan penjumlahan yang mudah, atau dapat dikatakan bahwa kompleksitas algoritmanya adalah O(1), akan tetapi dengan N <= M, algoritma ini tidak ada bedanya dengan algoritma vigenere cipher.

Andai M < N <= 2M, maka jumlah bilangan prima yang harus dibangkitkan adalah sebanyak (N-M) buah, sehingga kompleksitas dari algoritma ini, yang terasa di mata pengguna (karena pembangkitan bilangan prima membutuhkan waktu yang lama) adalah (N-M)X dengan X adalah lama dari pembangkitan bilangan prima.

Andai 2M < N, maka jumlah bilangan prima yang harus dibangkitkan adalah, untuk M karakter pertama tidak perlu dibangkitkan satu bilangan prima pun, untuk M karakter kedua, perlu dibangkitkan M bilangan prima, dan untuk karakter-karakter selanjutnya (anggap saja ada P karakter), perlu dibangkitkan sebanyak

$$\left\lfloor \frac{P}{M} \right\rfloor$$

karakter, sehingga untuk total N karakter yang ada perlu dibangkitkan sebanyak

$$M + \left\lfloor \frac{M \cdot 2N}{M} \right\rfloor$$

karakter, atau angka yang dihasilkan diatas dapat juga dinyatakan sebagai kompleksitas algoritma ini secara keseluruhan (perlu dikalikan lagi dengan kompleksitas menghasilkan bilangan prima). Dapat disimpulkan bahwa jumlah ini adalah jumlah yang tidak terlalu besar, tidak lebih besar dari *O(n)*, akan tetapi masih lebih besar daripada *O(1)*. Meskipun demikian, secara keseluruhan, implementasi dari algoritma ini terasa cukup cepat, tanpa memperhatikan bagaimana kompleksitas dari pembangkitan bilangan prima, karena bagaimanapun juga, kecepatan pembangkitan bilangan prima tidak dapat diperhitungkan dengan pasti.

3. HASIL IMPLEMENTASI

Contoh penggunaan algoritma vigenere minimum-prime key-adding cipher ini adalah sebagai berikut:

Diberikan suatu string “My name is Zakka Fauzan” dengan kunci “lampu”, maka langkah-langkah enkripsinya adalah sebagai berikut:

a. Hitung jumlah dari string “lampu”
   Diperoleh jumlahnya adalah l + a + m + p + u = 11 + 0 + 12 + 15 + 20 = 58.
b. Cari bilangan prima terkecil yang tidak kurang dari jumlah pada langkah a.
   Bilangan prima terkecil yang memenuhi adalah 59
c. Lakukan enkripsi pada karakter 1-5

dengan cara biasa (sama seperti pada vigenere cipher)
Karakter pertama sampai kelima dienkripsi dengan kunci “lampu”

d. Lakukan enkripsi pada karakter 6 dengan penambahan bilangan prima yang diperoleh pada langkah b. terhadap kunci
Karakter keenam akan dienkripsi dengan kunci ‘l’ + 59 = ‘s’

e. Lakukan enkripsi pada karakter selanjutnya sesuai dengan aturan pada bab 2.
Karakter ketujuh akan dienkripsi dengan kunci ‘a’ + 61 = ‘j’, karakter kedelapan akan dienkripsi dengan kunci ‘m’ + 67 = ‘b’, demikian seterusnya.

Sehingga hasil enkripsi dari “My Name Is Zakka Fauzan” dengan kunci “lampu” adalah “xyzpgwrthpezfpvushd”.

Untuk langkah dekripsi, lakukan hal yang berkebalikan dengan enkripsinya, hanya saja pada langkah d. dan e. dekripsi menyebabkan pengurangan nilai, bukan penambahan nilai seperti enkripsi.

Dengan cara yang sama, dengan kunci yang benar (yaitu “lampu”), string “xyzpgwrthpezfpvushd” akan dienkripsi menjadi “mynameiszakkafauzan”

4. KEKUATAN DAN KELEMAHAN ALGORITMA

Dari langkah-langkah implementasi tersebut, dapat diketahui bahwa algoritma ini memiliki kekuatan dan juga tetap memiliki beberapa kelemahan, kekuatan yang dimiliki oleh algoritma ini adalah sebagai berikut:

a. Perulangan yang terjadi di vigenere cipher, tidak akan terjadi pada algoritma ini, hal ini menyebabkan (dilihat dari contoh pada bab 3), meskipun terdapat karakter yang sama pada posisi perulangan yang sama (satu perulangan = panjang dari kunci = 5 karakter), ternyata hasilnya mungkin saja berbeda. Sebagai contoh, pada kasus “My Name Is Zakka Fauzan”, dengan mengabaikan karakter spasi, diperoleh bahwa karakter ke-13 sama dengan karakter ke-18, yaitu ‘a’, dan keduanya berada pada posisi perulangan yang sama untuk kunci “lampu” (5 karakter), akan tetapi hasil dari enkripsinya ternyata berbeda, ‘a’ pada karakter ke-13 dienkripsi dengan ‘m’ + 71 = ’f’ dan menghasilkan karakter ‘f’ sebagai karakter hasil enkripsinya, sedangkan ‘a’ pada karakter ke-18 dienkripsi dengan ‘m’ + 73 = ’h’ dan menghasilkan karakter ‘h’ sebagai karakter hasil enkripsinya.

b. Andaikan kriptanalis mengetahui sebagian dari kuncinya, ia tetap tidak akan mengetahui apa-apa, karena setiap karakter kunci berpengaruh besar terhadap setiap karakter hasil enkripsi, terutama pada karakter-karakter mulai dari karakter ke-6 sampai terakhir. Hal ini disebabkan adanya penggunaan bilangan prima yang dibangkitkan dari penjumlahan. Sebagai contoh nyata, andaikan kriptanalis mengetahui bahwa panjang karakter adalah 5 dan mengetahui bahwa karakter ke-1 sampai ke-4 berturut-turut adalah ‘l’, ‘a’, ‘m’, dan ‘p’, maka dengan hal tersebut (asumsikan karakter ke-5 pada kunci adalah karakter kosong), maka dengan menggunakan kunci “lamp “ (karakter ke-5 adalah karakter kosong), dengan teks yang didekripsi adalah “xyzpgwrthpezfpvushd”, hasil yang diperoleh adalah “mynagwamvocewzoovuh”, dari sini diketahui bahwa, meskipun pada karakter 1 sampai karakter 4 bersesuaian dengan karakter yang seharusnya “myna”, tetapi kriptanalis tidak mengetahui isi pesan yang sesungguhnya (isi pesan yang lengkap).

c. Bilangan prima adalah bilangan yang sulit dihasilkan, membutuhkan waktu yang lama untuk mengetes kevalidan suatu bilangan sebagai bilangan prima, sehingga jika nantinya kriptanalis melakukan percobaan dengan mencoba-coba semua kemungkinan bilangan prima, maka hasil yang diperoleh pun pasti akan memakan waktu sangat lama.

d. Adanya kemungkinan dua kunci menghasilkan bilangan prima yang sama, terutama jika panjang kuncinya adalah sama. Hal ini mengakibatkan, kriptanalis semakin sulit untuk menebak bilangan prima yang benar. Sebagai contoh, pada contoh soal di bab 3, andaikan kunci yang digunakan adalah “ykhbp” yang memiliki jumlah = 57, atau bilangan prima terdekat adalah 59, sama dengan yang diharapkan pada soal, akan tetapi karena kunci yang dimiliki oleh kriptanalis salah, maka hasil yang ia peroleh juga jauh dari yang diharapkan, karena dengan kunci “ykhbp”, teks hasil dekripsinya menjadi “zosorryxnfxaftfhpfb”, sangat berbeda dengan teks yang seharusnya.

e. Dari kekuatan tersebut, diperoleh satu kekuatan lagi, yaitu untuk mencoba mendekripsi suatu teks terenkripsi, setiap orang harus mencoba setiap kunci mulai dari 1 karakter sampai dengan n karakter

(dengan n adalah panjang kunci yang sesungguhnya), andaikan panjang kuncinya pun diketahui, seseorang tetap harus melakukan *brute force* percobaan untuk setiap kunci yang mungkin. Jika panjang kunci adalah 10 karakter dan itu diketahui oleh kriptanalis, maka ia harus mencoba lebih dari 140 trilyun kemungkinan kunci, jika setiap detiknya ia dapat menghasilkan 1 juta kunci, maka waktu yang dibutuhkan adalah 140 juta detik atau setara dengan 106 tahun pemecahan kode. Dari sini, jelas bahwa dengan prosesor secanggih apapun, pengenkripsian pesan akan sangat sulit dilakukan oleh siapapun yang tidak memiliki kewenangan dalam hal tersebut.

Disamping keunggulan-keunggulan tersebut, ternyata algoritma ini juga memiliki kelemahan, diantaranya adalah:

a. Hasil dekripsi yang benar pun tidak akan sama dengan pesan aslinya. Hal ini terlihat juga dari contoh, dengan pesan asli “My name is Zakka Fauzan” ternyata setelah dienkripsi dan didekripsi ulang akan menghasilkan pesan “mynameiszakkafauzan”, andaikan pesan tersebut jauh lebih panjang, maka penerima pesan harus menghabiskan waktu yang cukup lama untuk memisah-misahkan setiap kata pada kalimat yang bersesuaian, juga untuk perkiraan penggunaan huruf kapital yang tepat. Hal ini
b. Kesulitan menghasilkan bilangan prima bukan hanya dialami oleh kriptanalis, meskipun kriptanalis tentunya lebih sulit karena belum mengetahui kunci yang tepat, akan tetapi juga dialami oleh pengirim pesan atau penerima pesan. Andaikan pesan yang dikirimkan sangat panjang, mungkin memerlukan waktu yang cukup lama untuk melakukan pembangkitan bilangan prima dengan cepat, akan tetapi hal ini, menurut penulis, bukan merupakan sesuatu hal yang cukup signifikan, karena untuk percobaan enkripsi dan dekripsi terhadap kurang lebih 1000 karakter saja dapat dilakukan kurang dari satu detik.
c. Pergeseran sedikit kunci dapat membangkitkan bilangan prima yang sama, misalnya “lampu” yang berjumlah 58 membangkitkan bilangan prima 59, sama dengan “kampu” yang berjumlah 57 juga akan menghasilkan bilangan prima 59, sehingga hal ini dapat sedikit memudahkan kriptanalis, meskipun dari percobaan yang dilakukan penulis, hal ini tidak berdampak cukup signifikan.

5. KEMUNGKINAN KRIPTANALISIS

Segala kekuatan dari algoritma ini, bagaimanapun juga, karena kekuatan kunci yang dimiliki. Hal ini mengakibatkan, algoritma ini sangat mudah dipecahkan jika pengiriman pesan selalu dilakukan dengan menggunakan kunci yang sama, karena dengan *known-plaintext attack* pemecahan kode akan sangat mudah dilakukan. Sebagai contoh, pada kasus yang sama pada soal, jika diketahui bahwa “My name is Zakka Fauzan” adalah plainteks dan “xyzpgwrthpezfpvushd” adalah cipherteks yang dihasilkan, maka kriptanalis dengan cepat dapat mengetahui kunci yang digunakan adalah “lampu”, karena pencarian dari kunci cukup menggunakan n karakter terdepan saja (dengan n adalah panjang dari kunci), andaikan pun panjang dari kunci tidak diketahui, maka percobaan pencarian dapat dilakukan sebanyak 1 + 2 + … + n kali saja, tidak akan membutuhkan waktu yang lama untuk pencarian kunci terhadap plainteks dan cipherteks jika memang kunci yang digunakan selalu sama untuk algoritma ini.

Selain dari hal diatas, kemungkinan kriptanalisis kedua adalah penebakan berdasarkan tanda baca, karena pada algoritma ini, tanda baca akan dibiarkan begitu saja, sehingga kriptanalis dapat memperkirakan awal kalimat, apa isi di dalam tanda kutip, dan lain-lain, hal ini mungkin dapat mempercepat percobaan kriptanalisis oleh kriptanalis.

6. KESIMPULAN

Dari semua hal diatas, diketahui bahwa ternyata algoritma vigenere minimum-prime key-adding cipher ini ternyata masih memiliki kelemahan mudah dipecahkan dengan menggunakan *known-plaintext attack*, akan tetapi hal tersebut dapat ditanggulangi dengan menggunakan kunci yang berbeda di setiap pengiriman pesan, atau penambahan algoritma baru, agar untuk karakter pertama sampai karakter ke-n (dengan n adalah panjang dari kunci) pun sudah dilakukan penambahan dengan bilangan tertentu. Salah satu cara yang mungkin dapat dilakukan adalah penambahan terhadap bilangan prima tidak hanya dilakukan di karakter ke-(n+1) dan seterusnya saja, akan tetapi penambahan terhadap bilangan prima juga dilakukan di karakter pertama, sehingga mungkin akan dirasakan jauh lebih sulit untuk menghasilkan karakter plainteks meskipun sudah pernah diketahui contoh plainteks dan cipherteks untuk kasus yang berbeda dengan kunci yang sama.

Selain dari itu, dapat disimpulkan bahwa algoritma

ini memiliki kekuatan yang cukup tinggi, memiliki kompleksitas yang tidak terlalu tinggi dibandingkan apabila penciptaan dari bilangan prima dilakukan di setiap karakternya (di algoritma ini, pembangkitan bilangan prima hanya dilakukan satu kali setiap n karakter [n adalah panjang dari kunci]), serta memiliki kekuatan yang secara keseluruhan hampir setara dengan penggunaan algoritma *one-time pad* dan memiliki kesederhanaan dan kemudahan implementasi yang hampir setara dengan vigenere cipher biasa.

Untuk semakin mempersulit algoritma ini, ada baiknya jika simbol-simbol dari teks yang bukan ‘a’-‘z’ dan ‘A’-‘Z’ juga ikut dienkripsi, sehingga kriptanalis tidak memiliki kemungkinan percobaan pemecahan teks terenkripsi dengan menggunakan pembacaan tanda koma. Selain dari hal itu, karakter yang dihasilkan mungkin lebih baik bukan hanya dari ‘a’-‘z’ saja, akan tetapi seluruh karakter yang ada di ASCII (ada 256 karakter), sehingga semakin menimbulkan kebingungan bagi kriptanalis untuk melakukan percobaan kriptanalisis terhadap teks terenkripsi.

Selain dari kedua hal tersebut, untuk menambah kemudahan dari penerima pesan, ada baiknya spasi ikut terenkripsi sehingga nantinya begitu didekripsi, spasi ikut terdekripsi menjadi spasi kembali, atau jika spasi tidak terenkripsi, posisi-posisi dari spasi harus dicatat, mungkin di *header* dari teks terenkripsi, dan ikut terenkripsi bersama dengan teks yang memang seharusnya dienkripsi. Selain itu, tidak perlu dilakukan perubahan dari teks dengan huruf kapital menjadi huruf kecil, karena nantinya dengan penggunaan keseluruhan karakter ASCII, penggunaan karakter apapun akan sangat menyulitkan bagi kriptanalis untuk melakukan usaha pemecahan teks terenkripsi.

DAFTAR REFERENSI

[1] R.Munir, *Diktat Kuliah IF5054 Kriptografi*, Program Studi Teknik Informatika Institut Teknologi Bandung, 2006.

LAMPIRAN KODE PROGRAM

Decrypting.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace KriptoMakalah
{
    class Decrypting
    {
        #region private attribute
        private string sentence;
        private int position;
        private int[] keyInt;
        private int[] dataPrime;
        #endregion

        #region get and set
        public string Sentence
        {
            get { return sentence; }
            set { sentence = value; }
        }

        public int Position
        {
            get { return position; }
            set { position = value; }
        }

        public int[] KeyInt
        {
            get { return keyInt; }
            set { keyInt = value; }
        }

        public int[] DataPrime
        {
            get { return dataPrime; }
            set { dataPrime = value; }
        }
        #endregion

        public Decrypting(string newSentence) {
            sentence = newSentence;
            position = 0;
        }

        public string decryptResult(string key)
        {
            Kripto k = new Kripto(key);

            DataPrime = new int[k.Key.Length - 1];

            KeyInt = new int[k.Key.Length];
            for (int i = 0; i < KeyInt.Length; i++)
            {
                KeyInt[i] = k.Key[i] - 'a';
            }

            bool firstRound = true;
            bool manyRounds = false;
            string sentenceResult = string.Empty;
            foreach (char c in sentence)
            {
                if (c >= 'a' && c <= 'z')
                {
                    string str = c.ToString();
                    char min = 'a';

                    int npn = nextPrimeNumber(firstRound, k, position, manyRounds);
                    int decode = c - KeyInt[position] - npn;
```

```
                    while (decode < min)
                    {
                        decode += 26;
                    }
                    sentenceResult += (char)decode;

                    if (position > 0)
                        DataPrime[position - 1] = npn;

                    position = (position + 1) % KeyInt.Length;

                    if (position == 0 && !firstRound)
                        manyRounds = true;
                    else if (position == 0)
                        firstRound = false;

                }
                else
                    sentenceResult += (char)c;
            }
            return sentenceResult;
        }

        public int nextPrimeNumber(bool firstRound, Kripto k, int position, bool manyRounds)
        {
            if (firstRound)
            {
                return 0;
            }
            else
            {
                if (position != KeyInt.Length - 1 && DataPrime[position] != 0 && manyRounds)
                {
                    return (DataPrime[position]);
                }
                else
                {
                    k.generateNextPrime();

                    if (manyRounds)
                    {
                        DataPrime[DataPrime.Length - 1] = k.LastPrime;
                    }

                    return k.LastPrime;
                }
            }
        }
    }
}
```

Crypting.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Text.RegularExpressions;

namespace KriptoMakalah
{
    class Crypting
    {
        #region private attribute
        private string sentence;
        private int position;
        private int[] keyInt;
        private int[] dataPrime;

        #endregion

        #region get and set
        public string Sentence
        {
            get { return sentence; }
            set { sentence = value; }
        }
```

```
public int Position
{
    get { return position; }
    set { position = value; }
}

public int[] KeyInt
{
    get { return keyInt; }
    set { keyInt = value; }
}

public int[] DataPrime
{
    get { return dataPrime; }
    set { dataPrime = value; }
}
#endregion

public Crypting(string newSentence) {
    sentence = newSentence.Replace(" ", string.Empty).ToLower();
    //sentence = newSentence;
    position = 0;
}

public string encryptResult(string key) {
    Kripto k = new Kripto(key);

    DataPrime = new int[k.Key.Length - 1];

    KeyInt = new int[k.Key.Length];
    for (int i = 0; i < KeyInt.Length; i++) {
        KeyInt[i] = k.Key[i] - 'a';
    }

    bool firstRound = true;
    bool manyRounds = false;
    string sentenceResult = string.Empty;
    foreach (char c in sentence) {
        if (c >= 'a' && c <= 'z')
        {
            string str = c.ToString();
            char max = 'z';

            int npn = nextPrimeNumber(firstRound, k, position, manyRounds);
            int code = c + KeyInt[position] + npn;

            while (code > max)
            {
                code -= 26;
            }
            sentenceResult += (char)code;

            if(position > 0)
                DataPrime[position-1] = npn;

            position = (position + 1) % KeyInt.Length;

            if (position == 0 && !firstRound)
                manyRounds = true;
            else if (position == 0)
                firstRound = false;

        }
        else
            sentenceResult += (char)c;
    }
    return sentenceResult;
}

public int nextPrimeNumber(bool firstRound, Kripto k, int position, bool manyRounds) {
    if (firstRound)
    {
        return 0;
    }
    else {
        if (position != KeyInt.Length - 1 && DataPrime[position] != 0 && manyRounds)
```

```
                {
                    return (DataPrime[position]);
                }
                else
                {
                    k.generateNextPrime();

                    if (manyRounds){
                        DataPrime[DataPrime.Length - 1] = k.LastPrime;
                    }

                    return k.LastPrime;
                }
            }
        }
    }
}
```

Kripto.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace KriptoMakalah
{
    class Kripto
    {
        #region private attribute
        private string key;
        private int sumKey = 0;
        private int lastPrime = 0;
        #endregion

        #region get and set
        public string Key
        {
            get { return key; }
            set { key = value; }
        }

        public int SumKey
        {
            get { return sumKey; }
            set { sumKey = value; }
        }

        public int LastPrime
        {
            get { return lastPrime; }
            set { lastPrime = value; }
        }
        #endregion

        public Kripto(string newKey){
            key = newKey;
            foreach (char x in key) {
                sumKey += x - 'a';
            }
        }

        public void generateNextPrime(){
            int bound = sumKey - 1;
            if (lastPrime != 0){
                bound = lastPrime;
            }

            getNextPrime(bound);
        }

        private void getNextPrime(int bound) {
            bool found = false;
            for (int i = bound + 1; !found; i++) {
                found = checkPrime(i);
                if (found) {
                    lastPrime = i;
                }
```

```
            }
        }

        private bool checkPrime(int x){
            for (int i = 2; i < Math.Sqrt(x); i++){
                if (x % i == 0) {
                    return false;
                }
            }
            return true;
        }
    }
}
```