# Penerapan Kriptografi Dalam Pengamanan Penyimpanan Password

**Kamal Mahmudi**

Mahasiswa Jurusan Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Labtek V, Jalan Ganeca 10 Bandung
Email: if17111@students.if.itb.ac.id

**ABSTRAK**

**Aplikasi *multiuser* baik aplikasi berbasis web maupun aplikasi berbasis desktop saat ini sudah bukan barang baru lagi. Bersifat *multiuser* berarti memungkinkan setiap *user* memiliki hak penggunaan yang berbeda juga memungkinkan setiap *user* memiliki barang/item yang dimiliki secara privat. Untuk mengenali user yang sedang mengakses aplikasi, sistem melakukan otentikasi terlebih dahulu yang secara umum dilakukan dengan pengecekan *password*.**

***User* yang baik tentunya paham akan kerahasiaan *password* tersebut dan menyimpannya seaman mungkin. Pengamanan yang lemah pada saat *user* mengirimkan *password*-nya untuk diotentikasi ataupun saat *password*-nya disimpan pada basis data seolah membuat usaha *user* dalam menjaga kerahasiaan *password*-nya menjadi sia-sia.**

**Makalah ini akan membahas penerapan kriptografi klasik pada pengamanan penyimpanan password, melihat kekurangan dan kelebihan dari berbagai *cipher*, serta mencoba memperbaiki kekurangannya sehingga menghasilkan *cipher* yang layak untuk diterapkan pada pengamanan penyimpanan *password*.**

**Kata kunci:** basis data, cipher, enkripsi satu arah, kriptografi klasik, otentikasi, password

## 1. PENDAHULUAN

Secara umum *user* tidak pernah meragukan aplikasi dalam menjaga data *user* dan *password* yang telah disimpan pada basis data aplikasi. Faktanya tidak jarang aplikasi menyimpan *password* tersebut apa adanya yang dimungkinkan karena adanya unsur kesengajaan, kelalaian, atau bahkan keterbatasan pengetahuan *programmer* aplikasi tersebut. Dapat dibayangkan dampak apa yang mungkin muncul apabila terdapat pihak yang dapat melihat basis data aplikasis tersebut.

Fakta lain yang lebih sering terjadi adalah *programmer* melakukan enkripsi terlebih dahulu sebelum menyimpan *password* pada basis data aplikasi. Dengan hal ini berarti apabila terdapat pihak yang dapat melihat basis data, belum tentu dia dapat mengetahui *password* yang disimpan. Pertanyaan yang muncul kemudian adalah seberapa kuat enkripsi yang dilakukan? Seberapa umum pula enkripsi yang digunakan? Karena kebanyakan *programmer* tidak mau direpotkan membuat modul enkripsi, sehingga memilih menggunakan enkripsi yang telah disediakan oleh bahasa pemorgraman yang digunakan. Dampaknya tidak sedikit pula pihak yang berusaha mengalahkan enkripsi tersebut dan mempublikasikan karyanya.

Membuat modul enkripsi sendiri bukan berarti tidak mempunyai risiko. Selain kekuatan enkripsi yang menjadi pertanyaan, keberadaan pihak-pihak yang memiliki kemampuan untuk mengakses *source code* aplikasi tersebut memperbesar kemungkinan enkripsi tersebut dapat dipatahkan dengan mempelajari metode enkripsi yang digunakan.

Hingga saat ini, tentunya sudah banyak solusi yang dikembangkan. Jika tidak, tentunya aplikasi *multiuser* tidak akan diminati, padahal aplikasi *social networking* merupakan aplikasi *multiuser* yang paling banyak diminati dunia dari berbagai kalangan dan latar belakang yang berbeda-beda.

Meskipun demikian, mengembangkan solusi sendiri mempunyai nilai tambah yang tidak bisa dilupakan. Dengan mengembangkan metode enkripsi buatan sendiri membuat kriptanalisis harus berfikir dari awal untuk memecahkannya,

bandingkan bila metode enkripsi yang digunakan pernah digunakan dan dianalisis sebelumnya, kriptanalisis dapat menggunakan hasil analisis tersebut untuk mempercepat langkah awalnya bahkan mungkin bisa membawanya pada suatu kesimpulan. Hal ini menjadikan kriptanalisis berfikir dua kali untuk melakukan analasis, membandingkan keuntungan yang dia peroleh apabila dapat memecahkan enkripsi tersebut dengan usaha yang telah dia lakukan untuk memecahkannya.

## 2. PENGAMANAN PENYIMPANAN PASSWORD

Ada banyak cara yang dapat dilakukan *programmer* dalam menyimpan *password* yang digunakan dalam otentikasi pada basis data. Setiap cara tentu memiliki kekurangan dan kelebihan yang berbeda dari cara yang lain.

### 2. 1. Mengandalkan Sistem Utama

Mengandalkan sistem utama di sini maksudnya adalah *programmer* menyerahkan segala faktor keamanan pada sistem utama. *Password user* akan disimpan apa adanya dengan asumsi sistem utama aman dari segala bentuk tindakan yang merugikan *user*. Cara ini bisa dikatakan cara yang paling mudah dilakukan, bahkan *programmer* tidak perlu memikirkan/melakukan sesuatu yang berhubungan dengan sisi keamanan. Kelemahan cara ini adalah tidak menutup kemungkinan terdapat pihak dalam yang dapat dengan mudah mengakses basis data aplikasi melakukan tindakan yang merugikan user, selain itu apabila sistem utama berhasil diretas oleh *user* yang tidak memiliki hak akses, *user* tersebut dapat dengan mudah mengekstrak data yang ada pada basis data.

### 2. 2. Cipher Subtitusi

Secara umum cipher subtitusi dapat dikelompokkan menjadi empat jenis, cipher abjad-tunggal, cipher subtitusi homofonik, cipher abjad-majemuk, dan cipher subtitusi poligram. Pada cipher abjad-tunggal, satu karakter pada pada plainteks disubtitusi dengan satu karakter yang bersesuaian. Pada cipher subtitusi homofonik satu karakter pada plainteks dapat disubtitusi dengan salah satu dari beberapa karakter yang bersesuaian. Cipher abjad-majemuk dibuat dari sejumlah cipher abjad-tunggal dengan kunci yang berbeda-beda yang kebanyakannya bersifat periodik. Berbeda dengan jenis lainnya, cipher subtitusi poligram melakukan subtitusi blok dengan ukuran tertentu dari plainteks menjadi blok baru pada cipherteks.

Penggunaan cipher subtitusi pada pengamanan penyimpanan *password* mampu meningkatkan keamanan *password* yang disimpan. Apabila sistem utama berhasil diretas oleh *user* yang tidak memiliki hak akses, *user* tersebut belum tentu dapat mengekstrak data yang ada pada basis data. Namun, belum tentu bukan berarti menutup kemungkinan data tersebut dapat diekstrak. Dengan tambahan usaha, misalnya analisis frekuensi, *user* tersebut bisa saja mampu menganalisis algoritma enkripsi yang digunakan serta kunci yang digunakan dan membuat algoritma dekripsi untuk mendekripsi data yang ada pada basis data.

Kelemahan lain dari cara ini adalah *source code* dari aplikasi ini dengan jelas menunjukkan algoritma enkripsi yang digunakan bahkan juga memberi tahu kunci yang digunakan untuk melakukan enkripsi. Seperti halnya pada cara mengandalkan sistem utama, cara ini tidak menutup kemungkinan terdapat pihak dalam yang dapat dengan mudah mengakses *source code* dan basis data yang bisa saja membuat algoritma dekripsi dengan acuan *source code* tersebut untuk mengekstrak data yang ada pada basis data.

### 2. 3. Cipher Transposisi

Cipher transposisi juga dikenal sebagai cipher permutasi. Hal ini dikarenakan cipher transposisi tidak melakukan perubahan nilai karakter plainteks, melainkan melakukan perubahan urutan karakter plainteks. Dalam melakukan permutasi plainteks, ada banyak metode yang dapat dilakukan, misalkan Rail Fence cipher, Route cipher, dan Columnar cipher.

Sama seperti cipher subtitusi, cipher transposisi mampu meningkatkan keamanan *password* yang disimpan yang apabila sistem utama berhasil diretas oleh *user* yang tidak memiliki hak akses, *user* tersebut belum tentu dapat mengekstrak data yang ada pada basis data. Namun, dengan melakukan analasis frekuensi, *user* tersebut dapat dengan mudah menyadari bahwa cipher transposisilah yang digunakan dikarenakan frekuensi karakter pada cipherteks menunjukkan pola yang sama dengan frekuensi karakter pada plainteks. Dengan melakukan anagram, pada

akhirnya *user* tersebut dapat mengekstrak data yang diinginkannya.

Begitu pula halnya pada kasus *source code*, cara ini tidak menutup kemungkinan terdapat pihak dalam yang dapat dengan mudah mengakses *source code* dan basis data yang bisa saja membuat algoritma dekripsi dengan acuan *source code* tersebut untuk mengekstrak data yang ada pada basis data dikarenakan *source code* dengan jelas mendeskripsikan algoritma permutasi yang digunakan.

## 2. 4. Fungsi *Hash* Satu-Arah Publik

Berbeda dengan kedua cipher sebelumnya, fungsi *hash* satu-arah memetakan string dengan panjang yang bebas menjadi string dengan panjang tertentu yang tetap. Pemetaan yang bersifat satu-arah menjadikan keluaran fungsi ini tidak dapat dikembalikan menjadi string masukan. Ada beberapa fungsi *hash* satu-arah publik yang sudah dibuat, antara lain MD5 dan SHA.

Penggunaan fungsi *hash* satu-arah pada pengamanan penyimpanan *password* meningkatkan keamanan *password* mengingat *password* disimpan dalam bentuk yang secara komputasi tidak dapat dikembalikan ke bentuk string aslinya. Pada kasus *source code*, cara ini dapat menutup kemungkinan terdapat pihak dalam yang dapat dengan mudah mengakses *source code* dan basis data yang bisa saja membuat algoritma dekripsi dengan acuan *source code* tersebut untuk mengekstrak data yang ada pada basis data dikarenakan meskipun *source code* dengan jelas menunjukkan algoritma fungsi *hash* satu-arah yang digunakan tetap saja string yang dihasilkan tidak dapat dikembalikan ke string awal.

Walaupun demikian, penggunaan fungsi *hash* satu-arah publik memiliki celah keamanan yang patut untuk dipertimbangkan. Fungsi tersebut belum tentu telah terimplementasi pada bahasa pemrograman yang digunakan *programmer* dalam membangun aplikasi. Selain itu, penggunaan fungsi *hash* satu-arah publik pada pengamanan penyimpanan *password* memiliki resiko tinggi untuk dipecahkan. Panjang *password* yang secara logika tidak akan melebihi 32 karakter membuat tidak sedikit orang membuat daftar plainteks dan *hash* yang dihasilkan, sehingga apabila terdapat *hash password*, seseorang cukup melakukan pencarian pada daftar tersebut untuk mendapatkan string semula. Fungsi *hash* satu-arah publik biasanya dapat ditemukan aplikasi dekripsi *hash password* di internet. Beberapa aplikasi *online* yang cukup terkenal adalah http://www.md5crack.com dan http://www.passcracking.com sementara aplikasi *offline* yang juga terkenal adalah *Cain and Abel*.

# 3. ALTERNATIF PENGAMANAN PENYIMPANAN PASSWORD

Metode-metode yang telah dibahas sebelumnya sebenarnya layak diterapkan pada kasus-kasus tertentu. Namun secara umum, penggunaan metode-metode tersebut layak untuk dipertimbangkan, mengingat masing-masingnya memiliki kelebihan dan kekurangan yang berbeda-beda. Karenanya perlu dicari sebuah alternatif lain dalam proses pengamanan penyimpanan *password*.

## 3. 1. Konsep Utama

Penggunaan cipher subtitusi ataupun cipher transposisi memiliki keunikan tersendiri misalkan hasil yang diperoleh pada algoritma yang sama belum tentu sama apabila kunci yang digunakan berbeda, namun kekuatan algoritma yang terletak pada kunci menjadi tidak aman apabila kunci dapat dengan mudah diketahui pihak yang tidak berkepentingan, sementara pada aplikasi kunci dapat dengan mudah disebutkan pada *source code* aplikasi tersebut.

Penggunaan fungsi *hash* satu-arah publik memang memiliki keunggulan pada algoritma yang memiliki kekuatan terpisah dari kunci. Namun sifatnya yang publik menjadikannya sering digunakan yang berarti sering pula orang tertarik untuk memecahkannya. Selain itu penggunaan fungsi *hash* ini masih bergantung pada ketersediaan fungsi tersebut pada bahasa pemrograman yang digunakan.

Dari kelebihan dan kekurangan masing-masing metode tersebut dapat diambil sebuah solusi alternatif yang menggabungkan kelebihan dan saling menutupi kekurangan. Agar *password* yang disimpan aman, diperlukan enkripsi satu arah. Karena salah satu titik lemah fungsi *hash* publik ada pada faktor terlalu sering digunakan sehingga banyak pihak yang tertarik melakukan serangan, maka perlu dibentuk algoritma yang bersifat fleksibel dan yang menggabungkan beberapa proses enkripsi sekaligus. Bersifat fleksibel di sini dimaksudkan agar algoritma dapat digunakan pada sistem lain namun sudah termodifikasi sehingga

algoritma yang dihasilkan benar-benar berbeda dan keamanan *password* semakin kuat karena waktu untuk memecahkannya menjadi panjang disebabkan oleh proses pemecahan yang harus dimulai dari awal untuk setiap sistemnya.

Sementara itu, penggunaan cipher subtitusi dan cipher transposisi yang memiliki kekurangan pada kemungkinan serangan yang menghubungkan statistik pada cipherteks dengan statistik pada plainteks dapat jauh dikurangi dengan menyembunyikan serta merumitkan hubungan statistik yang terjadi.

Dari pemikiran tersebut, algoritma ini akan melakukan permutasi terhadap *password* yang akan dienkripsi, melakukan kompresi atau ekspansi agar memiliki ukuran yang diinginkan, membaginya menjadi beberapa bagian, mengenkripsi tiap-tiap bagian tersebut secara individual, mengkombinasikan bagian-bagian tersebut (misalnya dengan fungsi enkripsi di mana salah satu bagian menjadi plainteks dan salah satu bagian menjadi kunci secara terus menerus) sampai hanya ada satu string yang merupakan string akhir yang akan disimpan pada basis data.

Misalkan KE(S) adalah fungsi yang melakukan kompresi/ekspansi terhadap string S dan mengeluarkan hasil berupa string, P(S) adalah fungsi yang melakukan permutasi terhadap string S dan mengeluarkan hasil berupa string, Pisah(S, n) adalah fungsi yang membagi string S menjadi *array* string yang berukuran n elemen, E(S) adalah fungsi yang melakukan enkripsi terhadap string S dan mengeluarkan hasil berupa string, Gabung(S[]) adalah fungsi yang melakukan kombinasi terhadap *array* string S dan mengeluarkan hasil berupa string dan $S_5$ merupakan hasil yang disimpan pada basis data, maka secara umum algoritma ini dapat berupa:

$S_1 = KE(S_0)$

$S_2 = P(S_1)$

$S_3[] = Pisah(S_2, n)$

Untuk $0 <= i < n$, $S_4[i] = E(S_3[i])$

$S_5 = Gabung(S_4[])$

Untuk menghindari penggunaan algoritma yang sama, detail algoritma dapat dirubah menyesuaikan kebutuhan *programmer*.

## 3. 2. Implementasi

Implementasi yang akan dirancang pada makalah ini mengambil kasus aplikasi berbasis web pada bagian perubahan *password* (bagian ini melibatkan pengecekan *password* lama dan penyimpanan *password* baru) dan enkripsi dipecah menjadi dua bagian, yaitu pada klien dan server sehingga bahasa pemrograman yang akan digunakan adalah javascript dan PHP. Enkripsi pada klien diperlukan agar *password* yang dikirim bersifat aman ketika berada pada jalur komunikasi yang tidak aman. Sementara enkripsi pada sisi server diperlukan agar keseluruhan detail algoritma tidak diketahui oleh publik meskipun detail algoritma tetap dapat dilihat dari *file* javascipt dan PHP yang ada pada server.

Berikut beberapa file yang mengimplementasikan konsep yang telah dibahas pada bagian sebelumnya:

1. clientEncryption.js merupakan kelas yang menangani enkripsi pada klien, melakukan tahap kompresi/ekspansi, permutasi, dan pembagian string. Kompresi dilakukan dengan membuang dua karakter pertama dan mengoperasikan keduanya lalu disimpan pada cipherteks hingga mencapai jumlah yang diinginkan, sementara ekspansi dilakukan dengan menambahkan karakter baru dengan subtitusi satu persatu karakter plainteks dengan kunci berupa karakter terakhir pada plainteks. Permutasi dilakukan dengan permutasi kolom dengan kunci “mahmudi”. Pembagian string hanya membagi string menjadi dua bagian sama panjang.
2. serverEncryption.php merupakan kelas yang menangani enkripsi pada server, melakukan tahap enkripsi setiap bagian secara terpisah, serta menggabungkan kembali setiap bagian menjadi string tunggal. Tahap enkripsi setiap bagian pada implementasi yang dilakukan memanfaatkan fungsi ekspansi yang sama seperti pada clientEncryption.js. Penggabungan setiap bagian dilakukan dengan menjadikan salah satu bagian sebagai plainteks sementara bagian lainnya menjadi kunci yang kemudian dilakukan cipher subtitusi.
3. index.html merupakan antarmuka *user* yang juga memanggil kelas clientEncryption.js
4. process.php merupakan file yang melakukan pengecekan hasil enkripsi *password* lama terhadap database serta menyimpan hasil enkripsi *password* baru di database apabila sesuai.

Implementasi keempat file tersebut disertakan pada akhir makalah ini.

# 4. HASIL DAN ANALISIS

## 4. 1. Hasil Pengujian

Berikut adalah hasil pengujian terhadap beberapa *password*:

1. *Password*: "hurufA6kali"
   Tahap I: "hurufA6kali%X$W22U5/d!o!n0Y,H!4:"
   Tahap II: "ua2o!AX/Yrl2!46$d,hkW!HuiUn:f%50"
   Tahap III: "ua2o!AX/Yrl2!46$", "d,hkW!HuiUn:f%50"
   Tahap IV: "ua2o!AX/Yrl2!46$WwLzYYvn!Q2Q<sL,", "d,hkW!HuiUn:f%50A,U5.z7Ow^RLqTu"
   Tahap V: " N]=[!em7!=vEgXlWjfRMP2V:*bf!]h"
2. *Password*: "hurufB6kali"
   Tahap I: "hurufB6kali%X$W23V6;e@p@o~Z<K%9A"
   Tahap II: "ua2p%BX;Zrl3@96$e<hkW@KuiVoAf%6~"
   Tahap III: "ua2p%BX;Zrl3@96$", "e<hkW@KuiVoAf%6~"
   Tahap IV: "ua2p%BX;Zrl3@96$WwL1de2u(Y0Z|8aC", "e<hkW@KuiVoAf%6~C?Y9/5$W5,bb7jC@"
   Tahap V: "=P],|#hn8@,3FlYmYnjWV]d^lL}16Ec$"
3. *Password*: "inipasswordsepanjangtigapuluhdua"
   Tahap I: "inipasswordsepanjangtigapuluhdua"
   Tahap II: "nongdseglirjausptuiwaihpdapaasnu"
   Tahap III: "nongdseglirjausp", "tuiwaihpdapaasnu"
   Tahap IV: "nongdseglirjausp>e&Lo:Y5Dl>ZzHzC", "tuiwaihpdapaasnuBv?n#Jq;Ww|ZzFsA"
   Tahap V: ":]_>4.!_$9:01A;{?,9yPD$xZ[<yMMFC"
4. *Password*: "a"
   Tahap I: "a1Oc5Ws}]/-7a1Oc5Ws}]/-7a1Oc5Ws}"
   Tahap II: "1]c-WWa}OO/57ss1]ca}O/5c-Wa}57s1"
   Tahap III: "1]c-WWa}OO/57ss1", "]ca}O/5c-Wa}57s1"
   Tahap IV: "1]c-WWa}OO/57ss1OKmWs$CAOcU=s}qE", "]ca}O/5c-Wa}57s1w=MKYQ)K:QqoG@S*"
   Tahap V: "wY3)kO/a}kS3YM}O~]yg^/-KIs~ay~9-"

## 4. 2. Analisis

Pada pengujian 1 dan 2 terlihat bahwa perubahan 1 karakter dapat mempengaruhi hasil keseluruhan, namun pada tahap I (yaitu tahap ekspansi) dapat disadari bahwa perbedaan penambahan terjadi pada karakter keenam yang baru dimana karakter keenam pada plainteks juga berbeda, bagian ini dapat menarik perhatian kriptanalis.

Pada pengujian 3, terlihat bahwa sampai tahap III string yang disimpan masih berupa plainteks yang teracak, hal ini dikarenakan panjang password sama dengan panjang hasil kompresi/ekspansi. Namun pada tahap III string telah dibagi menjadi bagian yang lebih pendek sehingga dapat diekspansi dan kembali menghasilkan string acak.

Pada pengujian 4, *password* yang sangat pendek langsung menghasilkan string acak pada setiap tahapnya.

Hasil pengamatan lanjut menunjukkan bahwa apabila panjang *password* tidak sama dengan panjang hasil kompresi/ekspansi, maka akan dihasilkan string acak tambahan untuk setiap tahap. Namun apabila sama, string acak tetap akan dihasilkan pada tahap IV di mana string dibagi menjadi lebih pendek sebelum diekspansi.

Tahap terakhir yang merupakan penggabungan setiap bagian menjadi string tunggal menggunakan cipher subtitusi dimana panjang plainteks sama dengan panjang kunci yang keduanya merupakan string acak, sehingga kelemahan cipher subtitusi yang dapat diserang dengan hubungan statistik dapat ditutupi mengingat baik plainteks, kunci, maupun cipherteks bukan merupakan kata, ataupun karakter pada bahasa percakapan manusia.

Tahap kompresi/ekspansi merupakan tahap yang paling banyak menggunakan sumber daya sistem. Namun dikarenakan panjang *password* normal tidak mungkin melebihi 32 karakter, menjadikan penggunaannya relatif kecil.

## 5. KESIMPULAN

Ada banyak cara yang dapat diterapkan dalam menjaga keamanan penyimpanan *password*. Setiap cara memiliki kelebihan dan kekurangannya masing-masing.

Penerapan kriptografi pada penyimpanan *password* dapat meningkatkan keamanan *password* yang disimpan.

Penggunaan algoritma non-publik atau algoritma publik yang termodifikasi dapat meningkatkan keamanan *password*.

Keamanan *password* yang disimpan pada basis data menjadikannya sebagai pertahanan terakhir apabila terjadi pencurian *password* yang bisa saja dilakukan oleh pihak internal maupun pihak eksternal, baik dengan memanfaatkan hak akses yang dimilikinya ataupun secara paksa.

Algoritma enkripsi yang fleksibel memudahkan *programmer* dalam menempatkan algoritma yang sama dengan detail yang berbeda pada sistem yang berbeda. Hal ini menyebabkan apabila suatu sistem berhasil ditaklukkan, sistem lain belum tentu dapat.

Implementasi dari alternatif pengamanan yang ditawarkan pada makalah ini dapat dipertimbangkan penggunaannya mengingat algoritma yang bersifat fleksibel sehingga *programmer* dapat mengganti detail seperti karakter-karakter yang akan dihasilkan pada hasil akhir enkripsi, atau algoritma yang digunakan untuk permutasi, atau bahkan menggunakan fungsi *hash* satu-arah seperti md5 untuk menggantikan fungsi enkripsi E(S) yang dibahas sebelumnya.

## REFERENSI


[1] Kenan, Kevin. *Cryptography in the Database: The Last Line of Defense*. 2005. Addison-Wesley Professional.

[2] Schneier, Bruce. *Applied Cryptography*. 1996. John Wiley & Sons, Inc.

[3] http://islab.oregonstate.edu/koc/ece575/

[4] http://www.securitydocs.com/library/3079

[5] http://www.securitydocs.com/library/3547

[6] http://www.simonsingh.net/Crypto_Corner.html

1. **clientEncryption.js**

```
var clientEncryption = function(plaintext) {
    //Penampung string hasil pengolahan setiap tahap
    this.S = new Array(4);
    //panjang string password yang diinginkan
    this.passwordLength = 32;
    //90 karakter yang diperbolehkan dikombinasikan menjadi password
    this.chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890~!@#$%^&*()_-
=,.<>?/;:[]{}|";
    //Banyaknya bagian yang dihasilkan pada tahap memecah string
    this.n = 2;

    if(plaintext != null) this.S[0] = plaintext;
    else this.S[0] = "defaultstring";

    this.setPlaintext = function(newPlaintext) {
        if(newPlaintext != null) this.S[0] = newPlaintext;
    }

    this.encrypt = function() {
        this.compressionExpansion();
        this.permutation();
        this.split();
        return this.S[3];
    }

    //melakukan tahap kompresi/ekspansi
    this.compressionExpansion = function() {
        if(this.S[0].length < this.passwordLength)
            this.S[1] = this.expand(this.S[0], this.passwordLength);
        else
            this.S[1] = this.compress(this.S[0], this.passwordLength);
    }

    //melakukan tahap permutasi dengan transposisi columnar dengan key "mahmudi"
    this.permutation = function() {
        this.S[2] = this.columnarTransposition(this.S[1], "mahmudi");
    }

    //melakukan tahap membagi string dengan jumlah tertentu
    this.split = function() {
        this.S[3] = new Array(this.n);
        var length = Math.ceil(this.S[2].length / this.n);
        for(var i = 0; i < this.n; i++)
            this.S[3][i] = this.S[2].substr(i*length, i + length);
    }

    //melakukan transposisi columnar
    this.columnarTransposition = function(plaintext, key) {
        var i, j;
        var nCharPlaintext = plaintext.length;
        var nCharKey = key.length;
        var nRow = Math.ceil(nCharPlaintext / nCharKey);

        var posKey = new Array(nCharKey);
        var temp = new Array(nCharKey);
        for(i = 0; i < nCharKey; i++)
            temp[i] = key.charAt(i);
        temp.sort();

        posKey[0] = key.indexOf(temp[0]);
        for(i = 1; i < nCharKey; i++) {
            if(temp[i] == temp[i-1])
                posKey[i] = key.indexOf(temp[i], posKey[i-1]+1);
            else
                posKey[i] = key.indexOf(temp[i]);
        }

        var cell = new Array(nRow);
```

```
            for(i = 0; i < nRow; i++) {
                cell[i] = new Array(nCharKey);
                for(j = 0; j < nCharKey; j++)
                    cell[i][j] = plaintext.charAt(i*nCharKey+posKey[j]);
            }

            var ciphertext = "";
            for(j = 0; j < nCharKey; j++)
                for(i = 0; i < nRow; i++)
                    ciphertext += cell[i][j];

            return ciphertext;
        }

        //melakukan ekspansi string
        this.expand = function(plaintext, length) {
            var ciphertext = plaintext;
            var i = 0;
            //menambahkan karakter baru ke akhir string
            //karakter baru adalah hasil operasi subtitusi karakter ke i dengan karakter terakhir
            while(ciphertext.length < length) {
                var iTh = ciphertext.charAt(i);
                var last = ciphertext.charAt(ciphertext.length-1);
                ciphertext += this.subtitutionChar(iTh, last);
                i++;
            }
            return ciphertext;
        }

        //melakukan kompresi string
        this.compress = function(plaintext, length) {
            var ciphertext = "";
            var i = 0;
            //membuang setiap 2 karakter awal,mengoperasikannya, dan menambahkan pada ciphertext sampai
    panjang tertentu
            while(plaintext.length-i > length) {
                var i2Th = plaintext.charAt(i*2);
                var i2Plus1Th = plaintext.charAt(i*2+1);
                ciphertext += this.subtitutionChar(i2Th, i2Plus1Th);
                i++;
            }
            ciphertext += plaintext.substr(i*2, plaintext.length);
            return ciphertext;
        }

        //melakukan operasi subtitusi karakter plainteks dengan parameter key
        this.subtitutionChar = function(plaintext, key) {
            plaintext = this.chars.indexOf(plaintext);
            key = this.chars.indexOf(key);
            var ciphertext = this.chars.charAt((plaintext + key) % this.chars.length);
            return ciphertext;
        }
    }
```

**2. serverEncryption.php**

```
    <?php
    class ServerEncryption {
        //Penampung string hasil pengolahan setiap tahap
        protected $S = array();
        //90 karakter yang diperbolehkan dikombinasikan menjadi password
        protected $chars= "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890~!@#$%^&*()_-
    =,.<>?/;:[]{}|";

        public function __construct($arrayString = null) {
            if($arrayString != null)
                $this->set($arrayString);
        }

        public function set($arrayString) {
            $this->S[0] = $arrayString;
```

```
    }

    public function encrypt() {
        if(isset($this->S[0])) {
            $this->encryptPartition();
            $this->join();
        }
        return $this->S[2];
    }

    //melakukan tahap enkripsi tiap partisi secara terpisah
    private function encryptPartition() {
        $this->S[1] = array();
        $i = 0;
        foreach ($this->S[0] as $partition) {
            $this->S[1][$i] = $this->expand($partition, 32);
            $i++;
        }
    }

    //melakukan tahap menggabungkan setiap partisi
    private function join() {
        $n = count($this->S[1]);
        $this->S[2] = $this->S[1][0];
        for($i = 1; $i < $n; $i++)
            $this->S[2] = $this->subtitutionBlock($this->S[2], $this->S[1][$i]);
    }

    //melakukan ekspansi string
    private function expand($plaintext, $length) {
        $ciphertext = $plaintext;
        $i = 0;
        //menambahkan karakter baru ke akhir string
        //karakter baru adalah hasil operasi subtitusi karakter ke i dengan karakter terakhir
        while(strlen($ciphertext) < $length) {
            $iTh = $ciphertext[$i];
            $last = $ciphertext[strlen($ciphertext) - 1];
            $ciphertext .= $this->subtitutionChar($iTh, $last);
            $i++;
        }
        return $ciphertext;
    }

    //melakukan operasi subtitusi blok plainteks dengan parameter key secara periodik
    private function subtitutionBlock($plaintext, $key = null) {
        if($key == null) $key = "mahmudi";
        $ciphertext = "";
        $n = strlen($key);
        for($i = 0; $i < $n; $i++) {
            $char = $plaintext[$i];
            $ciphertext .= $this->subtitutionChar($char, $key[$i % $n]);
        }
        return $ciphertext;
    }

    //melakukan operasi subtitusi karakter plainteks dengan parameter key
    private function subtitutionChar($plaintext, $key) {
        $plaintext = strpos($this->chars, $plaintext);
        $key = strpos($this->chars, $key);
        $ciphertext = $this->chars[($plaintext + $key) % strlen($this->chars)];
        return $ciphertext;
    }
}
?>
```

3. **index.html**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
```

```
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
        <link rel="stylesheet" href="style.css" type="text/css" />
        <script type="text/javascript" src="clientEncryption.js"></script>
        <script type="text/javascript">
            window.onerror = function (sMessage, sUrl, sLine) {
                alert("An error occurred:\n" + sMessage + "\nURL: " + sUrl + "\nLine: " + sLine);
                return true;
            }
            var submitForm = function() {
                var args = "?u="+document.getElementById("username").value;
                var encoder = new clientEncryption();
                var temp, i;

                var pwd = document.getElementById("pwd").value;
                encoder.setPlaintext(pwd);
                temp = encoder.encrypt();
                for(i = 0; i < encoder.n; i++)
                    args += "&p1[]=" + escape(temp[i]);


                var newPwd = document.getElementById("new-pwd").value;
                encoder.setPlaintext(newPwd);
                temp = encoder.encrypt();
                for(i = 0; i < encoder.n; i++)
                    args += "&p2[]=" + escape(temp[i]);

                window.location = "proses.php" + args;
            }
        </script>
        <title></title>
    </head>
    <body>
        <div id="container">
            <form action="javascript:submitForm();">
                <fieldset>
                    <legend>Change Password</legend>
                    <label for="username">Username</label>
                    <input id="username" />
                    <label for="pwd">Password</label>
                    <input type="password" id="pwd" />
                    <label for="new-pwd">New Password</label>
                    <input type="password" id="new-pwd" />
                    <input type="submit" class="submit" value="Change" />
                </fieldset>
            </form>
        </div>
    </body>
</html>
```

4. **process.php**

```
<?php
    include "serverEncryption.php";

    $oldPasswd = new ServerEncryption($_GET["p1"]);
    $newPasswd = new ServerEncryption($_GET["p2"]);

    $link_identifier = mysql_connect("localhost", "web", "rahasia");
    mysql_select_db("tes", $link_identifier);
    $query = "UPDATE user SET password = '".$newPasswd->encrypt()."' WHERE username = '".$_GET["u"]."'
AND password = '".$oldPasswd->encrypt()."'";
    $result = mysql_query($query, $link_identifier);

    if(mysql_affected_rows($link_identifier) > 0) echo "Password changed";
    else echo "Password not changed.<br />Probably because: <ul><li>wrong password</li><li>new
password as same as new one</li><li>internal error</li></ul>";
?>
```