**MAKALAH**

**DOKUMENTASI KRIPTOGRAFI**

****

Disusun oleh :

**FANOLO GULO**

**2015 – 81 – 006**

**PROGRAM STUDI S-1 TEKNIK INFORMATIKA**

**FAKULTAS ILMU KOMPUTER**

**UNIVERSITAS ESA UNGGUL**

**JAKARTA**

**2018**

1. **DES ( DATA ENCRYPTION STANDARD )**
   1. **Fungsi DES**

Secara umum tujuan DES diciptakan adalah untuk merahasiakan suatu pesan agar tidak bisa dibaca oleh pihak yang tidak absah. berpedoman dari tujuan umum tersebut, sehinga pesan yang dikirimkan aman dan tidak dapat dibajak.

* 1. **Cara Kerja**

Dalam DES, algoritma dekripsi tepatnya merupakan proses kebalikan (inverse) algoritma enkripsi. Dalam prakteknya proses pembalikan (proses dekripsi) ini diimplementasikan dengan membalikkan urutan sub kunci yang digunakan dalam proses enkripsi, selebihnya algoritma enkripsi dan dekripsi adalah sama. Algoritma enkripsi DES bekerja dengan mengolah blok data 8 byte (64 bit) dengan blok kunci 8 byte (64 bit). Proses penyandian dalam DES diawali dengan fungsi pengacakan bit yang dinamai IP (Initial Permutation) kemudian fungsi inti DES yang diulang sebanyak 16 kali dan terakhir ditutup dengan fungsi pengacakan bit lain yang dikenal denagn nama IP‐1 (Inverse Initial Permutation). Pada sisi lain algoritma penjadwalan sub kunci akan menghasilkan 16 sub kunci secara berurutan dari parameter kunci yang diberikan untuk digunakan pada setiap putaran fungsi inti DES. Sub kunci pertama untuk putaran pertama, sub kunci kedua untuk putaran kedua dan seterusnya hingga putaran ke 16.

Perlu diingat, kendatipun slot kunci yang disediakan digunakan berukuran 8 byte (64 bit), ternyata pada faktanya ukuran kunci yang digunakan hanya sebanyak 56 bit saja, karena bit paling signifikan (MSB) dari setiap bit diabaikan. Jadi sebenarnya ukuran kunci DES adalah 56 bit. Adapun Ilustrasi penyandian DES dalam diagram blok dapat dilihat pada gambar di samping. Algoritma penjadwalan sub kunci dibentuk dari pengacakan bit dan pemutaran kiri ruas kanan dan kiri kunci. Pertama kali, bit-bit kunci diacak dengan Permutation Choice 1 dan dibagi dua menjadi ruas kiri dan ruas kanan. Kedua ruas tersebut kemudian diputar kiri dan diacak kembali dengan Permutation Choice 2 untuk menghasilkan sub kunci. Jumlah pemutaran ke kiri ditentukan secara spesifik untuk setiap sub kunci. Rinciannya adalah sebagai berikut.

[](https://link.safelinkconverter.com/?id=aHR0cDovL3B1c2F0LW1ha2FsYWhrdS5ibG9nc3BvdC5jb20v&c=0&user=61204)

Untuk lebih lanjut, mari terlebih dahulu kita bahas detail algoritma penjadwalan sub kunci kemudian algoritma enkripsi dan terakhir algoritma dekripsi. Inti dari semua proses permutasi dalam DES adalah pengacakan bit. Sebagai contoh, jika masukan permutasi sebanyak n bit, maka akan ada sebanyak 2n kemungkinan masukan permutasi dan ada 2n kemungkinan hasil permutasi. Setiap satu kemungkinan masukan akan berpasangan dengan satu kemungkinan keluaran.

Sebelum proses penjadwalan kunci dimulai, kunci terlebih dahulu dipetakan menjadi matriks 8x8 dan diberi indeks. Dalam setiap byte, indeks paling kecil melambangkan LSB dan indeks paling besar melambangkan MSB. Sebagai contoh, indeks ke 1 melambangkan LSB byte pertama, index ke 8 melambangkan MSB byte pertama, indeks ke 9 melambangkan LSB byte kedua, indeks ke 16 melambangkan MSB byte kedua dan seterusnya hingga indeks ke 64 yang melambangkan MSB byte ke 8. Mari kita perhatikan contoh dibawah ini. Kunci = 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef.

[](https://link.safelinkconverter.com/?id=aHR0cDovL3B1c2F0LW1ha2FsYWhrdS5ibG9nc3BvdC5jb20v&c=0&user=61204)

Setelah pemetaan, hasilnya adalah sebagai berikut.

[](https://link.safelinkconverter.com/?id=aHR0cDovL3B1c2F0LW1ha2FsYWhrdS5ibG9nc3BvdC5jb20v&c=0&user=61204)

Matriks Indeks Hasil Pemetaan Pengacakan bit Permutation Choice‐1 akan mengolah 8 byte blok kunci menjadi 56 bit sub kunci yang siap diproses lebih lanjut. Untuk lebih mudahnya, proses pengacakan bit dilambangkan dengan pengacakan indeks bit yang bersangkutan. Berikut ini adalah detail Permutation Choice 1.

[](https://link.safelinkconverter.com/?id=aHR0cDovL3B1c2F0LW1ha2FsYWhrdS5ibG9nc3BvdC5jb20v&c=0&user=61204)

Setelah kunci diacak dengan Permutation Choice 1, hasil pengacakan bit tersebut kemudian dibagi 2, yakni ruas kiri dan ruas kanan, masing masing berukuran 28 bit (ditandai dengan garis tebal pada hasil Permutation Choice 1). Selanjutnya kedua ruas tersebut kemudian mengalami pemutaran kiri sebanyak jumlah yang tertera pada tabel penjadwalan jumlah pemutaran yang telah kita bahas sebelumnya. Berikut ini adalah ilustrasi pemutaran ke kiri sebanyak 1 kali (untuk ruas kiri atau ruas kanan kunci yang panjangnya 28 bit). Untuk pemutaran ke kiri dengan jumlah yang lebih besar, cukup mengulangi proses ini sebanyak yang diinginkan.

[](https://link.safelinkconverter.com/?id=aHR0cDovL3B1c2F0LW1ha2FsYWhrdS5ibG9nc3BvdC5jb20v&c=0&user=61204)

  Setelah ruas kiri dan ruas kanan diputar kiri dengan jumlah tertentu, selanjutnya hasil pemutaran tersebut digabungkan kembali menjadi 56 bit dan diacak dengan Permutation Choice‐2 untuk menghasilkan sub kunci. Rincian Permutation Choice 2 adalah sebagai berikut.

[](https://link.safelinkconverter.com/?id=aHR0cDovL3B1c2F0LW1ha2FsYWhrdS5ibG9nc3BvdC5jb20v&c=0&user=61204)

Karena jumlah keluaran Permutation Choice 48 bit sementara masukannya 56 bit, dengan demikian ada 8 bit yang “dihilangkan”. Bit-bit yang tidak muncul dalam keluaran Permutation Choice 2 diwarnai abuabu. Selanjutnya, mari kita perjelas algoritma enkripsinya. Setelah melihat diagram blok secara keseluruhan proses enkripsi, ada tiga hal yang perlu digarisbawahi dan dibahas lebih lanjut yaitu, pertama IP (Initial Permutation), kedua detail fungsi F dan IP 1 (Inverse Initial Permutation).

Selama proses enkripsi, pertama data dipetakan dan diberi indeks dengan prosedur sama persis seperti pemberian indeks pada penjadwalan kunci yang telah didiskusikan sebelumnya. Selanjutnya hasil pemetaan diacak dengan menggunakan Initial Permutation dengan rincian sebagai berikut Masukan IP

[](https://link.safelinkconverter.com/?id=aHR0cDovL3B1c2F0LW1ha2FsYWhrdS5ibG9nc3BvdC5jb20v&c=0&user=61204)

Setelah data melewati proses Initial Permutation, data yang akan disandikan kemudian dibagi menjadi dua ruas, yaitu ruas kiri dan ruas kanan yang masIng-masing lebarnya 4 byte (32 bit). Pada setiap putaran, ruas kanan dan sub kunci yang bersangkutan diproses dalam fungsi F dan hasilnya di XOR dengan ruas kiri ruas kanan dan kiri dipertukarkan. Proses ini diulang sebanyak 16 kali.

Pada putaran terakhir ruas kiri dan ruas kanan dipertukarkan kembali untuk menghilangkan efek pertukaran pada putaran terakhir. Hasil ini kemudian diacak kembali dengan menggunakan IP 1 (Inverse Initial Permutation). XOR merupakan fungsi Boolean yang didefinisikan dengan table benaran berikut.

[](https://link.safelinkconverter.com/?id=aHR0cDovL3B1c2F0LW1ha2FsYWhrdS5ibG9nc3BvdC5jb20v&c=0&user=61204)

Initial Permutatrion dan Inverse Initial Permutation memiliki karakter saling menetralkan. Dalam notasi matematika, IP−1(IP(A))= A. Inverse Initial Permutation didefinisikan sebagai berikut.

[](https://link.safelinkconverter.com/?id=aHR0cDovL3B1c2F0LW1ha2FsYWhrdS5ibG9nc3BvdC5jb20v&c=0&user=61204)

Fungsi F merupakan fungsi inti kompleks yang terdiri dari beberapa proses. Fungsi F menerima dua parameter, yaitu sub kunci dan ruas kanan data yang akan dienkripsi. Berikut ini adalah diagram blok detail fungsi F. Pada fungsi F, ruas kanan (32 bit) diacak sekaligus diperluas dengan permutasi E menjadi 48 bit. Hasil pengacakan tersebut kemudian di XOR dengan sub kunci yang telah ditetapkan dengan putaran yang bersangkutan. Hasil XOR kemudian dipecah menjadi 8 unit yang masing‐masing lebarnya 6 bit. Setiap unit tersebut kemudian disubstitusikan dalam SBOX S1 hingga S8. 6 bit paling kiri disubstitusikan ke dalam S1 dan 6 bit paling kanan disubstitusikan ke dalam S8. Hasil setiap substitusi kemudian digabungkan menjadi data selebar 48 bit yang kemudian diacak dan diperpendek dengan permutasi P menjadi 32 bit. Hasil permutasi P kemudian dinyatakan sebagai keluaran fungsi F yang nantinya akan di XOR kan dengan ruas kiri data yang akan dienkripsi.

[](https://link.safelinkconverter.com/?id=aHR0cDovL3B1c2F0LW1ha2FsYWhrdS5ibG9nc3BvdC5jb20v&c=0&user=61204)

Sekarang mari kita bahas detail fungsi F satu per satu. Pertama, permutasi E memetakan 32 bit masukan menjadi 48 bit keluaran. Karena lebar keluaran lebih besar dari lebar masukan, maka ada beberapa bit masukan yang digandakan untuk mengisi kekosongan. Permutasi E didefinisikan sebagai berikut

[](https://link.safelinkconverter.com/?id=aHR0cDovL3B1c2F0LW1ha2FsYWhrdS5ibG9nc3BvdC5jb20v&c=0&user=61204)

Kedua, DES memiliki 8 buah SBOX (S1 hingga S8) yang memiliki masukan selebar 6 bit dan keluaran selebar 4 bit. Karena lebar keluaran SBOX lebih kecil daripada lebar masukannya, maka adakemungkinan beberapa kombinasi masukan yang berbeda akan menghasilkan keluaran yang sama. Seandainya masukan setiap SBOX adalah 1 2 3 4 5 6 x x x x x x maka S1 hingga hingga S8 didefinisikan sebagai berikut.

[](https://link.safelinkconverter.com/?id=aHR0cDovL3B1c2F0LW1ha2FsYWhrdS5ibG9nc3BvdC5jb20v&c=0&user=61204)

[](https://link.safelinkconverter.com/?id=aHR0cDovL3B1c2F0LW1ha2FsYWhrdS5ibG9nc3BvdC5jb20v&c=0&user=61204)

Ketiga, hasil substitusi SBOX kemudian digabungkan menjadi 32 bit dan diacak dengan permutasi P dan hasil permutasi P merupakan keluaran fungsi F yang nantinya di XOR dengan ruas kiri. Permutasi P didefinisikan sebagai berikut.

[](https://link.safelinkconverter.com/?id=aHR0cDovL3B1c2F0LW1ha2FsYWhrdS5ibG9nc3BvdC5jb20v&c=0&user=61204)

Dalam DES, algoritma yang digunakan dalam proses enkripsi sama persis dengan algoritma yang digunakan dalam proses dekripsi, hanya saja penggunaan sub kuncinya saja yang berbeda. Dalam proses dekripsi, urutan sub kunci yang digunakan merupakan kebalikan urutan sub kunci yang digunakan dalam proses enkripsi.

* 1. **Kelebihan & Kekurangan**

Data Encryption Standard ( DES ) pun seperti sistem yang lainnya, memiliki kekurangan dan kelebihan. Salah satu kekurangan DES adalah proses yang lebih lama dalam melakukan proses dekripsi dan enkripsi.

Kelebihan DES yaitu, sistem sandi lebih kompleks sehingga membutuhkan waktu yang tidak cepat untuk menembus enkripsi DES dan sulit untuk diketahui dari pihak luar. Tetapi setelah berkembangannya jaman DES tidak digunakan karena ukuran kunci yang terlalu kecil, sehingga mudah untuk ditembus

* 1. **Source Code Aplikasi**

function encryptByDES(message, key) {

// For the key, when you pass a string,

// it's treated as a passphrase and used to derive an actual key and IV.

// Or you can pass a WordArray that represents the actual key.

// If you pass the actual key, you must also pass the actual IV.

var keyHex = CryptoJS.enc.Utf8.parse(key);

// console.log(CryptoJS.enc.Utf8.stringify(keyHex), CryptoJS.enc.Hex.stringify(keyHex));

// console.log(CryptoJS.enc.Hex.parse(CryptoJS.enc.Utf8.parse(key).toString(CryptoJS.enc.Hex)));

// CryptoJS use CBC as the default mode, and Pkcs7 as the default padding scheme

var encrypted = CryptoJS.DES.encrypt(message, keyHex, {

mode: CryptoJS.mode.ECB,

padding: CryptoJS.pad.Pkcs7

});

// decrypt encrypt result

// var decrypted = CryptoJS.DES.decrypt(encrypted, keyHex, {

// mode: CryptoJS.mode.ECB,

// padding: CryptoJS.pad.Pkcs7

// });

// console.log(decrypted.toString(CryptoJS.enc.Utf8));

// when mode is CryptoJS.mode.CBC (default mode), you must set iv param

// var iv = 'inputvec';

// var ivHex = CryptoJS.enc.Hex.parse(CryptoJS.enc.Utf8.parse(iv).toString(CryptoJS.enc.Hex));

// var encrypted = CryptoJS.DES.encrypt(message, keyHex, { iv: ivHex, mode: CryptoJS.mode.CBC });

// var decrypted = CryptoJS.DES.decrypt(encrypted, keyHex, { iv: ivHex, mode: CryptoJS.mode.CBC });

// console.log('encrypted.toString() -> base64(ciphertext) :', encrypted.toString());

// console.log('base64(ciphertext) <- encrypted.toString():', encrypted.ciphertext.toString(CryptoJS.enc.Base64));

// console.log('ciphertext.toString() -> ciphertext hex :', encrypted.ciphertext.toString());

return encrypted.toString();

}

/\*\*

\* Decrypt ciphertext by DES in ECB mode and Pkcs7 padding scheme

\*

\* @param {String} ciphertext(base64 string)

\* @param {String} key

\* @return {String} plaintext

\*

\* @author Sun

\* @version 2013-5-15

\*/

function decryptByDES(ciphertext, key) {

var keyHex = CryptoJS.enc.Utf8.parse(key);

// direct decrypt ciphertext

var decrypted = CryptoJS.DES.decrypt({

ciphertext: CryptoJS.enc.Base64.parse(ciphertext)

}, keyHex, {

mode: CryptoJS.mode.ECB,

padding: CryptoJS.pad.Pkcs7

});

return decrypted.toString(CryptoJS.enc.Utf8);

}

1. **TRIPLE DES**
   1. **Fungsi**

Pada awalnya, ukuran kunci sandi DES yaitu 56 bit sudah mencukupi pada saat algoritme ini dibuat. Namun, dengan meningkatnya kemampuan komputasi, serangan brutal telah mungkin terjadi. Triple DES menyediakan metode yang sederhana dengan menambah ukuran kunci DES untuk mencegah serangan tersebut, tanpa memerlukan perancangan sandi blok (block cipher) yang sama sekali baru.

* 1. **Cara kerja**

Tahap pertama, plainteks yang diinputkan dioperasikan dengan kunci eksternal pertama (K1) dan melakukan proses enkripsi dengan menggunakan algoritma DES. Sehingga menghasilkan pra-cipherteks pertama. Tahap kedua, pra-cipherteks pertama yang dihasilkan pada tahap pertama, kemudian dioperasikan dengan kunci eksternal kedua (K2) dan melakukan proses enkripsi atau proses dekripsi (tergantung cara pengenkripsian yang digunakan) dengan menggunakan algoritma DES. Sehingga menghasilkan prs-cipherteks kedua. Tahap terakhir, pra-cipherteks kedua yang dihasilkan pada tahap kedua, dioperasikan dengan kunci eksternal ketiga (K3) dan melakukan proses enkripsi dengan menggunakan algoritma DES, sehingga menghasilkan cipherteks (C).

Dalam kriptografi, Triple DES adalah nama umum untuk Algoritma Data Encryption Triple (TDEA atau Triple DEA) blok cipher, yang menerapkan Standar Enkripsi Data (DES) algoritma cipher tiga kali untuk setiap blok data.

Ukuran kunci DES asli cipher dari 56 bit pada umumnya cukup ketika algoritma yang dirancang, tetapi ketersediaan daya komputasi semakin membuat serangan brute force layak. Triple DES menyediakan metode yang relatif sederhana meningkatkan ukuran kunci DES untuk melindungi terhadap serangan tersebut, tanpa perlu merancang sebuah algoritma blok cipher baru.

[](http://3.bp.blogspot.com/-BbPPuMRqcw0/T-bu1K9lNwI/AAAAAAAAAGU/j7qmHMEQNIs/s1600/3.gif)

Seperti semua blok cipher, enkripsi dan dekripsi dari beberapa blok data dapat dilakukan dengan menggunakan berbagai modus operasi, yang secara umum dapat didefinisikan secara independen dari algoritma blok cipher. Namun, ANS X9.52 menentukan secara langsung, dan NIST SP 800-67 menentukan melalui SP 800-38A bahwa beberapa mode hanya dapat digunakan dengan batasan tertentu pada mereka yang belum tentu berlaku untuk spesifikasi umum dari mode. Sebagai contoh, ANS X9.52 menetapkan bahwa untuk chaining blok cipher, vektor inisialisasi akan berbeda setiap kali, sedangkan ISO / IEC 10116 tidak. PUB FIPS 46-3 dan ISO / IEC 18033-3 mendefinisikan hanya algoritma blok tunggal, dan tidak menempatkan batasan pada mode operasi untuk beberapa blok.

* 1. **Kelebihan & kekurangan**

Triple DES memiliki kekurangan pada performansi yang lambat dalam software. Kelebihan yang dimiliki oleh Triple DES adalah membuat menjebol enkripsinya sangat lama

* 1. **Source Code Aplikasi**

function decryptByTDES(message, key){

var keyHex = CryptoJS.enc.Base64.parse(key);

var decrypted = CryptoJS.TripleDES.decrypt(message, keyHex,

{

iv: keyHex,

mode: CryptoJS.mode.CBC,

padding: CryptoJS.pad.Pkcs7

});

return decrypted.toString(CryptoJS.enc.Utf8)

}

function encryptByTDES(message, key){

//var keyHex = CryptoJS.enc.Utf8.parse(key);

var keyHex = CryptoJS.enc.Base64.parse(key);

var encryptd = CryptoJS.TripleDES.encrypt(message, keyHex,

{

iv: keyHex,

mode: CryptoJS.mode.CBC,

padding: CryptoJS.pad.Pkcs7

});

//console.log(encrypted.ciphertext.toString());

return encryptd.toString();

}

1. **AES & RSA**
   1. **Fungsi**

Tujuan utama dari kriptografi adalah melindungi sebuah informasi, begitu pula dengan AES yang dengan serangkaian tahap atau ronde yang dilakukan dengan menggunakan kunci simetris. Penggunaan AES pun bukan hanya digunakan dalam hal yang sederhana melainkan perannya sangatlah krusial dalam sebuah perangkat lunak ataupun dalam hal lain dimana AES tersebut digunakan.

* 1. **Cara Kerja pada AES**

Berikut ini adalah operasi Rijndael (AES) yang menggunakan 128 bit kunci:

1. Ekspansi kunci utama (dari 128 bit menjadi 1408 bit)
2. Pencampuran subkey
3. Ulang dari i=1 sampai i=10 Transformasi : ByteSub (subtitusi per byte), ShiftRow (Pergeseren byte perbaris), MixColumn (Operasi perkalian GF(2) per kolom)
4. Pencampuran subkey (dengan XOR)
5. Transformasi : ByteSub dan ShiftRow
6. Pencampuran subkey
   1. **Cara kerja pada RSA**

Sebagai algoritma kunci publik, RSA mempunyai dua kunci, yaitu kunci publik dan kunci rahasia. Kunci publik boleh diketahui oleh siapa saja, dan digunakan untuk proses enkripsi. Sedangkan kunci rahasia hanya pihak-pihak tertentu saja yang boleh mengetahuinya, dan digunakan untuk proses dekripsi. Keamanan sandi RSA terletak pada sulitnya memfaktorkan bilangan yang besar. Sampai saat ini RSA masih dipercaya dan digunakan secara luas di internet.

Gambar – Skema Algoritma Kunci Publik, Sandi RSA terdiri dari tiga proses, yaitu proses pembentukan kunci, proses enkripsi dan proses deskripsi. Sebelumnya diberikan terlebih dahulu beberapa konsep perhitungan matematis yang digunakan RSA.

* 1. **Kelebihan & Kekurangan AES-RSA**

Kelebihan menggunakan struktur SPN (Substitution Permutation Network) yang memiliki derajat paralelisme yang lebih besar, sehingga diharapkan lebih cepat dari pada Feistel.

Kekurangan pada umumnya (termasuk pada Rijndael) adalah berbedanya struktur enkripsi dan dekripsi sehingga diperlukan dua algoritma yang berbeda untuk enkripsi dan dekripsi. Dan tentu pula tingkat keamanan enkripsi dan dekripsinya menjadi berbeda. AES memiliki blok masukan dan keluaran serta kunci 128 bit.

* 1. **Source Code Aplikasi**

var keySize = 256;

var ivSize = 128;

var iterations = 100;

function encryptByAES (msg, pass) {

var salt = CryptoJS.lib.WordArray.random(128/8);

var key = CryptoJS.PBKDF2(pass, salt, {

keySize: keySize/32,

iterations: iterations

});

var iv = CryptoJS.lib.WordArray.random(128/8);

var encrypted = CryptoJS.AES.encrypt(msg, key, {

iv: iv,

padding: CryptoJS.pad.Pkcs7,

mode: CryptoJS.mode.CBC

});

// salt, iv will be hex 32 in length

// append them to the ciphertext for use in decryption

var transitmessage = salt.toString()+ iv.toString() + encrypted.toString();

return transitmessage;

}

function decryptByAES (transitmessage, pass) {

var salt = CryptoJS.enc.Hex.parse(transitmessage.substr(0, 32));

var iv = CryptoJS.enc.Hex.parse(transitmessage.substr(32, 32))

var encrypted = transitmessage.substring(64);

var key = CryptoJS.PBKDF2(pass, salt, {

keySize: keySize/32,

iterations: iterations

});

var decrypted = CryptoJS.AES.decrypt(encrypted, key, {

iv: iv,

padding: CryptoJS.pad.Pkcs7,

mode: CryptoJS.mode.CBC

})

return decrypted;

}

//var encrypted = encrypt(message, password);

//var decrypted = decrypt(encrypted, password);

//$('#encrypted').text("Encrypted:"+encrypted);

//$('#decrypted').text("Decrypted:"+decrypted.toString(CryptoJS.enc.Utf8) );

1. **MD5 & SHA1**
   1. **Fungsi**

Algoritma yang menggunakan fungsi hash satu arah yang diciptakan oleh Ron Rivest. Algoritma merupakan pengembangan dari algoritma-algoritma sebelumnya yaitu algoritma MD2 dan algoritma MD4 karena kedua algoritma ini berhasil diserang para cryptanalist.

* 1. **Cara Kerja MD5**

Cara kerja kriptografi algoritma MD5 adalah menerima input berupa pesan dengan ukuran sembarang dan menghasilkan message diggest yang memiliki panjang 128 bit. Berikut ilustrasi gambar dari pembuatan message diggest pada kriptografi algoritma MD5:



Menilik dari gambar diatas, secara garis besar pembuatan message digest ditempuh melalui empat langkah, yaitu :

1. Penambahan bit bit pengganjal

Proses pertama yang dilakukan adalah menambahkan pesan dengan sejumlah bit pengganjal sedemikian sehingga panjang pesan (dalam satuan bit) kongruen dengan 448 modulo 512. Ini berarti setelah menambahkan bit-bit pengganjal, kini panjang pesan adalah 64 bit kurang dari kelipatan 512. Hal yang perlu diingat adalah angka 512 muncul karena algoritma MD5 memproses pesan dalam blok-blok yang berukuran 512.

Apabila terdapat pesan dengan panjang 448 bit, maka pesan tersebut akan tetap ditambahkan dengan bit-bit pengganjal. Pesan akan ditambahkan dengan 512 bit menjadi 96 bit. Jadi panjang bit-bit pengganjal adalah antara 1 sampai 512. Lalu satu hal lagi yang perlu diperhatikan adalah bahwasanya bit-bit pengganjal terdiri dari sebuah bit 1 diikuti dengan sisanya bit 0.

1. Penambahan nilai panjang pesan semula

Kemudian proses berikutnya adalah pesan ditambah lagi dengan 64 bit yang menyatakan panjang pesan semula. Apabila panjang pesan lebih besar dari 264 maka yang diambil adalah panjangnya dalam modulo 264. dengan kata lain, jika pada awalnya panjang pesan sama dengan K bit, maka 64 bit yang ditambahkan menyatakan K modulo 264. sehingga setelah proses kedua ini selesai dilakukan maka panjang pesan sekarang adalah 512 bit.

1. Inisialisasi penyangga MD5

Pada algoritma MD5 dibutuhkan empat buah penyangga atau buffer, secara berurut keempat nama penyangga diberi nama A, B, C dan D. Masing-masing penyangga memiliki panjang 32 bit. Sehingga panjang total :



Keempat penyangga ini menampung hasil antara dan hasil akhir. Setiap penyangga diinisialisasi dengan nilai-nilai (dalam notasi Hexadesimal) sebagai berikut :



1. Pengolahan pesan dalam blok berukuran 512 bit

Proses berikutnya adalah pesan dibagi menjadi L buah blok yang masing-masing panjangnya 512 bit (Y0 sampai YL-1). Setelah itu setiap blok 512 bit diproses bersama dengan penyangga MD yang menghasilkan keluaran 128 bit, dan ini disebut HMD5. Berikut ini gambaran dari proses HMD5 :



Dari gambar diatas dapat kita lihat bahwa proses HMD5 terdiri dari 4 buah putaran, dan masing-masing putaran melakukan opersi dasar MD5 sebanyak 16 kali. Dimana disetiap operasi dasar memakai sebuah elemen T. Sehingga setiap putaran memakai 16 elemen tabel T.

* 1. **Cara Kerja SH1**

Pesan diberi tambahan untuk membuat panjangnya menjadi kelipatan 512 bit ( l x 512 ). Jumlah bit asal adalah k bit. Tambahkan bit secukupnya sampai 64 bit kurangnya dari kelipatan 512 ( 512 – 64 = 448 ), yang disebut juga kongruen dengan 448 ( mod 512 ). Kemudian tambahkan 64 bit yang menyatakan panjang pesan. Inisiasi 5 md variable dengan panjang 32 bit yaitu a,b,c,d,e. Pesan dibagi menjadi blok-blok berukuran 512 bit dan setiap blok diolah. Kemudian keluaran setiap blok digabungkan dengan keluaran blok berikutnya, sehingga diperoleh output ( diggest ). Fungsi kompresi yang digunakan oleh algoritma sha-1 adalah sebagai berikut : A,b,c,d,e ← ( e + f (t,b,c,d) + s5 (a) + wt + kt),a,s30(b),c,d.]

* 1. **Kelebihan & kekurangan**

1. Keamanan terhadapserangan brute-force. Hal yang paling penting adalah bahwa SHA-1 menghasilkan diggest 32-bit lebih panjang dari MD5. Dengan brute-force maka SHA-1 lebihkuat dibanding MD5.
2. Keamanan terhadap kriptanalisis. Kelemahan MD5 ada pada design sehingga lebih mudah dilakukan kriptanalisis dibandingkan SHA-1
3. Kecepatan. Kedua algoritma bekerja pada modulo 232 sehingga keduanya bekerja baik pada arsitektur 32 bit. SHA-1 mempunyai langkah lebih banyak dibandingkan MD5 ( 80 dibanding MD5 64 ) dan harus memproses 160 bit buffer dibanding DM5 128 bit buffer, sehingga SHA-1 bekerja lebih lambat dibanding MD5 pada perangkat keras yang sama.
4. Simplicity. Kedua algoritma simple untuk dijelaskan dan mudah untuk diiemplementasikan karena tidak membutuhkan program yang besar atau table subtitusi yang besar pula.
   1. **Source Code Aplikasi**

<https://github.com/NaufalHSyahputra/CryptoTools/blob/master/proses/sha1.js>

<https://github.com/NaufalHSyahputra/CryptoTools/blob/master/proses/md5.js>

<https://github.com/NaufalHSyahputra/CryptoTools/blob/master/proses/sha1-core.js>

**SHA1 :**

/\*

\* [js-sha1]{@link https://github.com/emn178/js-sha1}

\*

\* @version 0.6.0

\* @author Chen, Yi-Cyuan [emn178@gmail.com]

\* @copyright Chen, Yi-Cyuan 2014-2017

\* @license MIT

\*/

/\*jslint bitwise: true \*/

(function() {

'use strict';

var root = typeof window === 'object' ? window : {};

var NODE\_JS = !root.JS\_SHA1\_NO\_NODE\_JS && typeof process === 'object' && process.versions && process.versions.node;

if (NODE\_JS) {

root = global;

}

var COMMON\_JS = !root.JS\_SHA1\_NO\_COMMON\_JS && typeof module === 'object' && module.exports;

var AMD = typeof define === 'function' && define.amd;

var HEX\_CHARS = '0123456789abcdef'.split('');

var EXTRA = [-2147483648, 8388608, 32768, 128];

var SHIFT = [24, 16, 8, 0];

var OUTPUT\_TYPES = ['hex', 'array', 'digest', 'arrayBuffer'];

var blocks = [];

var createOutputMethod = function (outputType) {

return function (message) {

return new Sha1(true).update(message)[outputType]();

};

};

var createMethod = function () {

var method = createOutputMethod('hex');

if (NODE\_JS) {

method = nodeWrap(method);

}

method.create = function () {

return new Sha1();

};

method.update = function (message) {

return method.create().update(message);

};

for (var i = 0; i < OUTPUT\_TYPES.length; ++i) {

var type = OUTPUT\_TYPES[i];

method[type] = createOutputMethod(type);

}

return method;

};

var nodeWrap = function (method) {

var crypto = eval("require('crypto')");

var Buffer = eval("require('buffer').Buffer");

var nodeMethod = function (message) {

if (typeof message === 'string') {

return crypto.createHash('sha1').update(message, 'utf8').digest('hex');

} else if (message.constructor === ArrayBuffer) {

message = new Uint8Array(message);

} else if (message.length === undefined) {

return method(message);

}

return crypto.createHash('sha1').update(new Buffer(message)).digest('hex');

};

return nodeMethod;

};

function Sha1(sharedMemory) {

if (sharedMemory) {

blocks[0] = blocks[16] = blocks[1] = blocks[2] = blocks[3] =

blocks[4] = blocks[5] = blocks[6] = blocks[7] =

blocks[8] = blocks[9] = blocks[10] = blocks[11] =

blocks[12] = blocks[13] = blocks[14] = blocks[15] = 0;

this.blocks = blocks;

} else {

this.blocks = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0];

}

this.h0 = 0x67452301;

this.h1 = 0xEFCDAB89;

this.h2 = 0x98BADCFE;

this.h3 = 0x10325476;

this.h4 = 0xC3D2E1F0;

this.block = this.start = this.bytes = this.hBytes = 0;

this.finalized = this.hashed = false;

this.first = true;

}

Sha1.prototype.update = function (message) {

if (this.finalized) {

return;

}

var notString = typeof(message) !== 'string';

if (notString && message.constructor === root.ArrayBuffer) {

message = new Uint8Array(message);

}

var code, index = 0, i, length = message.length || 0, blocks = this.blocks;

while (index < length) {

if (this.hashed) {

this.hashed = false;

blocks[0] = this.block;

blocks[16] = blocks[1] = blocks[2] = blocks[3] =

blocks[4] = blocks[5] = blocks[6] = blocks[7] =

blocks[8] = blocks[9] = blocks[10] = blocks[11] =

blocks[12] = blocks[13] = blocks[14] = blocks[15] = 0;

}

if(notString) {

for (i = this.start; index < length && i < 64; ++index) {

blocks[i >> 2] |= message[index] << SHIFT[i++ & 3];

}

} else {

for (i = this.start; index < length && i < 64; ++index) {

code = message.charCodeAt(index);

if (code < 0x80) {

blocks[i >> 2] |= code << SHIFT[i++ & 3];

} else if (code < 0x800) {

blocks[i >> 2] |= (0xc0 | (code >> 6)) << SHIFT[i++ & 3];

blocks[i >> 2] |= (0x80 | (code & 0x3f)) << SHIFT[i++ & 3];

} else if (code < 0xd800 || code >= 0xe000) {

blocks[i >> 2] |= (0xe0 | (code >> 12)) << SHIFT[i++ & 3];

blocks[i >> 2] |= (0x80 | ((code >> 6) & 0x3f)) << SHIFT[i++ & 3];

blocks[i >> 2] |= (0x80 | (code & 0x3f)) << SHIFT[i++ & 3];

} else {

code = 0x10000 + (((code & 0x3ff) << 10) | (message.charCodeAt(++index) & 0x3ff));

blocks[i >> 2] |= (0xf0 | (code >> 18)) << SHIFT[i++ & 3];

blocks[i >> 2] |= (0x80 | ((code >> 12) & 0x3f)) << SHIFT[i++ & 3];

blocks[i >> 2] |= (0x80 | ((code >> 6) & 0x3f)) << SHIFT[i++ & 3];

blocks[i >> 2] |= (0x80 | (code & 0x3f)) << SHIFT[i++ & 3];

}

}

}

this.lastByteIndex = i;

this.bytes += i - this.start;

if (i >= 64) {

this.block = blocks[16];

this.start = i - 64;

this.hash();

this.hashed = true;

} else {

this.start = i;

}

}

if (this.bytes > 4294967295) {

this.hBytes += this.bytes / 4294967296 << 0;

this.bytes = this.bytes % 4294967296;

}

return this;

};

Sha1.prototype.finalize = function () {

if (this.finalized) {

return;

}

this.finalized = true;

var blocks = this.blocks, i = this.lastByteIndex;

blocks[16] = this.block;

blocks[i >> 2] |= EXTRA[i & 3];

this.block = blocks[16];

if (i >= 56) {

if (!this.hashed) {

this.hash();

}

blocks[0] = this.block;

blocks[16] = blocks[1] = blocks[2] = blocks[3] =

blocks[4] = blocks[5] = blocks[6] = blocks[7] =

blocks[8] = blocks[9] = blocks[10] = blocks[11] =

blocks[12] = blocks[13] = blocks[14] = blocks[15] = 0;

}

blocks[14] = this.hBytes << 3 | this.bytes >>> 29;

blocks[15] = this.bytes << 3;

this.hash();

};

Sha1.prototype.hash = function () {

var a = this.h0, b = this.h1, c = this.h2, d = this.h3, e = this.h4;

var f, j, t, blocks = this.blocks;

for(j = 16; j < 80; ++j) {

t = blocks[j - 3] ^ blocks[j - 8] ^ blocks[j - 14] ^ blocks[j - 16];

blocks[j] = (t << 1) | (t >>> 31);

}

for(j = 0; j < 20; j += 5) {

f = (b & c) | ((~b) & d);

t = (a << 5) | (a >>> 27);

e = t + f + e + 1518500249 + blocks[j] << 0;

b = (b << 30) | (b >>> 2);

f = (a & b) | ((~a) & c);

t = (e << 5) | (e >>> 27);

d = t + f + d + 1518500249 + blocks[j + 1] << 0;

a = (a << 30) | (a >>> 2);

f = (e & a) | ((~e) & b);

t = (d << 5) | (d >>> 27);

c = t + f + c + 1518500249 + blocks[j + 2] << 0;

e = (e << 30) | (e >>> 2);

f = (d & e) | ((~d) & a);

t = (c << 5) | (c >>> 27);

b = t + f + b + 1518500249 + blocks[j + 3] << 0;

d = (d << 30) | (d >>> 2);

f = (c & d) | ((~c) & e);

t = (b << 5) | (b >>> 27);

a = t + f + a + 1518500249 + blocks[j + 4] << 0;

c = (c << 30) | (c >>> 2);

}

for(; j < 40; j += 5) {

f = b ^ c ^ d;

t = (a << 5) | (a >>> 27);

e = t + f + e + 1859775393 + blocks[j] << 0;

b = (b << 30) | (b >>> 2);

f = a ^ b ^ c;

t = (e << 5) | (e >>> 27);

d = t + f + d + 1859775393 + blocks[j + 1] << 0;

a = (a << 30) | (a >>> 2);

f = e ^ a ^ b;

t = (d << 5) | (d >>> 27);

c = t + f + c + 1859775393 + blocks[j + 2] << 0;

e = (e << 30) | (e >>> 2);

f = d ^ e ^ a;

t = (c << 5) | (c >>> 27);

b = t + f + b + 1859775393 + blocks[j + 3] << 0;

d = (d << 30) | (d >>> 2);

f = c ^ d ^ e;

t = (b << 5) | (b >>> 27);

a = t + f + a + 1859775393 + blocks[j + 4] << 0;

c = (c << 30) | (c >>> 2);

}

for(; j < 60; j += 5) {

f = (b & c) | (b & d) | (c & d);

t = (a << 5) | (a >>> 27);

e = t + f + e - 1894007588 + blocks[j] << 0;

b = (b << 30) | (b >>> 2);

f = (a & b) | (a & c) | (b & c);

t = (e << 5) | (e >>> 27);

d = t + f + d - 1894007588 + blocks[j + 1] << 0;

a = (a << 30) | (a >>> 2);

f = (e & a) | (e & b) | (a & b);

t = (d << 5) | (d >>> 27);

c = t + f + c - 1894007588 + blocks[j + 2] << 0;

e = (e << 30) | (e >>> 2);

f = (d & e) | (d & a) | (e & a);

t = (c << 5) | (c >>> 27);

b = t + f + b - 1894007588 + blocks[j + 3] << 0;

d = (d << 30) | (d >>> 2);

f = (c & d) | (c & e) | (d & e);

t = (b << 5) | (b >>> 27);

a = t + f + a - 1894007588 + blocks[j + 4] << 0;

c = (c << 30) | (c >>> 2);

}

for(; j < 80; j += 5) {

f = b ^ c ^ d;

t = (a << 5) | (a >>> 27);

e = t + f + e - 899497514 + blocks[j] << 0;

b = (b << 30) | (b >>> 2);

f = a ^ b ^ c;

t = (e << 5) | (e >>> 27);

d = t + f + d - 899497514 + blocks[j + 1] << 0;

a = (a << 30) | (a >>> 2);

f = e ^ a ^ b;

t = (d << 5) | (d >>> 27);

c = t + f + c - 899497514 + blocks[j + 2] << 0;

e = (e << 30) | (e >>> 2);

f = d ^ e ^ a;

t = (c << 5) | (c >>> 27);

b = t + f + b - 899497514 + blocks[j + 3] << 0;

d = (d << 30) | (d >>> 2);

f = c ^ d ^ e;

t = (b << 5) | (b >>> 27);

a = t + f + a - 899497514 + blocks[j + 4] << 0;

c = (c << 30) | (c >>> 2);

}

this.h0 = this.h0 + a << 0;

this.h1 = this.h1 + b << 0;

this.h2 = this.h2 + c << 0;

this.h3 = this.h3 + d << 0;

this.h4 = this.h4 + e << 0;

};

Sha1.prototype.hex = function () {

this.finalize();

var h0 = this.h0, h1 = this.h1, h2 = this.h2, h3 = this.h3, h4 = this.h4;

return HEX\_CHARS[(h0 >> 28) & 0x0F] + HEX\_CHARS[(h0 >> 24) & 0x0F] +

HEX\_CHARS[(h0 >> 20) & 0x0F] + HEX\_CHARS[(h0 >> 16) & 0x0F] +

HEX\_CHARS[(h0 >> 12) & 0x0F] + HEX\_CHARS[(h0 >> 8) & 0x0F] +

HEX\_CHARS[(h0 >> 4) & 0x0F] + HEX\_CHARS[h0 & 0x0F] +

HEX\_CHARS[(h1 >> 28) & 0x0F] + HEX\_CHARS[(h1 >> 24) & 0x0F] +

HEX\_CHARS[(h1 >> 20) & 0x0F] + HEX\_CHARS[(h1 >> 16) & 0x0F] +

HEX\_CHARS[(h1 >> 12) & 0x0F] + HEX\_CHARS[(h1 >> 8) & 0x0F] +

HEX\_CHARS[(h1 >> 4) & 0x0F] + HEX\_CHARS[h1 & 0x0F] +

HEX\_CHARS[(h2 >> 28) & 0x0F] + HEX\_CHARS[(h2 >> 24) & 0x0F] +

HEX\_CHARS[(h2 >> 20) & 0x0F] + HEX\_CHARS[(h2 >> 16) & 0x0F] +

HEX\_CHARS[(h2 >> 12) & 0x0F] + HEX\_CHARS[(h2 >> 8) & 0x0F] +

HEX\_CHARS[(h2 >> 4) & 0x0F] + HEX\_CHARS[h2 & 0x0F] +

HEX\_CHARS[(h3 >> 28) & 0x0F] + HEX\_CHARS[(h3 >> 24) & 0x0F] +

HEX\_CHARS[(h3 >> 20) & 0x0F] + HEX\_CHARS[(h3 >> 16) & 0x0F] +

HEX\_CHARS[(h3 >> 12) & 0x0F] + HEX\_CHARS[(h3 >> 8) & 0x0F] +

HEX\_CHARS[(h3 >> 4) & 0x0F] + HEX\_CHARS[h3 & 0x0F] +

HEX\_CHARS[(h4 >> 28) & 0x0F] + HEX\_CHARS[(h4 >> 24) & 0x0F] +

HEX\_CHARS[(h4 >> 20) & 0x0F] + HEX\_CHARS[(h4 >> 16) & 0x0F] +

HEX\_CHARS[(h4 >> 12) & 0x0F] + HEX\_CHARS[(h4 >> 8) & 0x0F] +

HEX\_CHARS[(h4 >> 4) & 0x0F] + HEX\_CHARS[h4 & 0x0F];

};

Sha1.prototype.toString = Sha1.prototype.hex;

Sha1.prototype.digest = function () {

this.finalize();

var h0 = this.h0, h1 = this.h1, h2 = this.h2, h3 = this.h3, h4 = this.h4;

return [

(h0 >> 24) & 0xFF, (h0 >> 16) & 0xFF, (h0 >> 8) & 0xFF, h0 & 0xFF,

(h1 >> 24) & 0xFF, (h1 >> 16) & 0xFF, (h1 >> 8) & 0xFF, h1 & 0xFF,

(h2 >> 24) & 0xFF, (h2 >> 16) & 0xFF, (h2 >> 8) & 0xFF, h2 & 0xFF,

(h3 >> 24) & 0xFF, (h3 >> 16) & 0xFF, (h3 >> 8) & 0xFF, h3 & 0xFF,

(h4 >> 24) & 0xFF, (h4 >> 16) & 0xFF, (h4 >> 8) & 0xFF, h4 & 0xFF

];

};

Sha1.prototype.array = Sha1.prototype.digest;

Sha1.prototype.arrayBuffer = function () {

this.finalize();

var buffer = new ArrayBuffer(20);

var dataView = new DataView(buffer);

dataView.setUint32(0, this.h0);

dataView.setUint32(4, this.h1);

dataView.setUint32(8, this.h2);

dataView.setUint32(12, this.h3);

dataView.setUint32(16, this.h4);

return buffer;

};

var exports = createMethod();

if (COMMON\_JS) {

module.exports = exports;

} else {

root.sha1 = exports;

if (AMD) {

define(function () {

return exports;

});

}

}

})();

**MD5 :**

var encryptbyMD5 = function (string) {

function RotateLeft(lValue, iShiftBits) {

return (lValue<<iShiftBits) | (lValue>>>(32-iShiftBits));

}

function AddUnsigned(lX,lY) {

var lX4,lY4,lX8,lY8,lResult;

lX8 = (lX & 0x80000000);

lY8 = (lY & 0x80000000);

lX4 = (lX & 0x40000000);

lY4 = (lY & 0x40000000);

lResult = (lX & 0x3FFFFFFF)+(lY & 0x3FFFFFFF);

if (lX4 & lY4) {

return (lResult ^ 0x80000000 ^ lX8 ^ lY8);

}

if (lX4 | lY4) {

if (lResult & 0x40000000) {

return (lResult ^ 0xC0000000 ^ lX8 ^ lY8);

} else {

return (lResult ^ 0x40000000 ^ lX8 ^ lY8);

}

} else {

return (lResult ^ lX8 ^ lY8);

}

}

function F(x,y,z) { return (x & y) | ((~x) & z); }

function G(x,y,z) { return (x & z) | (y & (~z)); }

function H(x,y,z) { return (x ^ y ^ z); }

function I(x,y,z) { return (y ^ (x | (~z))); }

function FF(a,b,c,d,x,s,ac) {

a = AddUnsigned(a, AddUnsigned(AddUnsigned(F(b, c, d), x), ac));

return AddUnsigned(RotateLeft(a, s), b);

};

function GG(a,b,c,d,x,s,ac) {

a = AddUnsigned(a, AddUnsigned(AddUnsigned(G(b, c, d), x), ac));

return AddUnsigned(RotateLeft(a, s), b);

};

function HH(a,b,c,d,x,s,ac) {

a = AddUnsigned(a, AddUnsigned(AddUnsigned(H(b, c, d), x), ac));

return AddUnsigned(RotateLeft(a, s), b);

};

function II(a,b,c,d,x,s,ac) {

a = AddUnsigned(a, AddUnsigned(AddUnsigned(I(b, c, d), x), ac));

return AddUnsigned(RotateLeft(a, s), b);

};

function ConvertToWordArray(string) {

var lWordCount;

var lMessageLength = string.length;

var lNumberOfWords\_temp1=lMessageLength + 8;

var lNumberOfWords\_temp2=(lNumberOfWords\_temp1-(lNumberOfWords\_temp1 % 64))/64;

var lNumberOfWords = (lNumberOfWords\_temp2+1)\*16;

var lWordArray=Array(lNumberOfWords-1);

var lBytePosition = 0;

var lByteCount = 0;

while ( lByteCount < lMessageLength ) {

lWordCount = (lByteCount-(lByteCount % 4))/4;

lBytePosition = (lByteCount % 4)\*8;

lWordArray[lWordCount] = (lWordArray[lWordCount] | (string.charCodeAt(lByteCount)<<lBytePosition));

lByteCount++;

}

lWordCount = (lByteCount-(lByteCount % 4))/4;

lBytePosition = (lByteCount % 4)\*8;

lWordArray[lWordCount] = lWordArray[lWordCount] | (0x80<<lBytePosition);

lWordArray[lNumberOfWords-2] = lMessageLength<<3;

lWordArray[lNumberOfWords-1] = lMessageLength>>>29;

return lWordArray;

};

function WordToHex(lValue) {

var WordToHexValue="",WordToHexValue\_temp="",lByte,lCount;

for (lCount = 0;lCount<=3;lCount++) {

lByte = (lValue>>>(lCount\*8)) & 255;

WordToHexValue\_temp = "0" + lByte.toString(16);

WordToHexValue = WordToHexValue + WordToHexValue\_temp.substr(WordToHexValue\_temp.length-2,2);

}

return WordToHexValue;

};

function Utf8Encode(string) {

string = string.replace(/\r\n/g,"\n");

var utftext = "";

for (var n = 0; n < string.length; n++) {

var c = string.charCodeAt(n);

if (c < 128) {

utftext += String.fromCharCode(c);

}

else if((c > 127) && (c < 2048)) {

utftext += String.fromCharCode((c >> 6) | 192);

utftext += String.fromCharCode((c & 63) | 128);

}

else {

utftext += String.fromCharCode((c >> 12) | 224);

utftext += String.fromCharCode(((c >> 6) & 63) | 128);

utftext += String.fromCharCode((c & 63) | 128);

}

}

return utftext;

};

var x=Array();

var k,AA,BB,CC,DD,a,b,c,d;

var S11=7, S12=12, S13=17, S14=22;

var S21=5, S22=9 , S23=14, S24=20;

var S31=4, S32=11, S33=16, S34=23;

var S41=6, S42=10, S43=15, S44=21;

string = Utf8Encode(string);

x = ConvertToWordArray(string);

a = 0x67452301; b = 0xEFCDAB89; c = 0x98BADCFE; d = 0x10325476;

for (k=0;k<x.length;k+=16) {

AA=a; BB=b; CC=c; DD=d;

a=FF(a,b,c,d,x[k+0], S11,0xD76AA478);

d=FF(d,a,b,c,x[k+1], S12,0xE8C7B756);

c=FF(c,d,a,b,x[k+2], S13,0x242070DB);

b=FF(b,c,d,a,x[k+3], S14,0xC1BDCEEE);

a=FF(a,b,c,d,x[k+4], S11,0xF57C0FAF);

d=FF(d,a,b,c,x[k+5], S12,0x4787C62A);

c=FF(c,d,a,b,x[k+6], S13,0xA8304613);

b=FF(b,c,d,a,x[k+7], S14,0xFD469501);

a=FF(a,b,c,d,x[k+8], S11,0x698098D8);

d=FF(d,a,b,c,x[k+9], S12,0x8B44F7AF);

c=FF(c,d,a,b,x[k+10],S13,0xFFFF5BB1);

b=FF(b,c,d,a,x[k+11],S14,0x895CD7BE);

a=FF(a,b,c,d,x[k+12],S11,0x6B901122);

d=FF(d,a,b,c,x[k+13],S12,0xFD987193);

c=FF(c,d,a,b,x[k+14],S13,0xA679438E);

b=FF(b,c,d,a,x[k+15],S14,0x49B40821);

a=GG(a,b,c,d,x[k+1], S21,0xF61E2562);

d=GG(d,a,b,c,x[k+6], S22,0xC040B340);

c=GG(c,d,a,b,x[k+11],S23,0x265E5A51);

b=GG(b,c,d,a,x[k+0], S24,0xE9B6C7AA);

a=GG(a,b,c,d,x[k+5], S21,0xD62F105D);

d=GG(d,a,b,c,x[k+10],S22,0x2441453);

c=GG(c,d,a,b,x[k+15],S23,0xD8A1E681);

b=GG(b,c,d,a,x[k+4], S24,0xE7D3FBC8);

a=GG(a,b,c,d,x[k+9], S21,0x21E1CDE6);

d=GG(d,a,b,c,x[k+14],S22,0xC33707D6);

c=GG(c,d,a,b,x[k+3], S23,0xF4D50D87);

b=GG(b,c,d,a,x[k+8], S24,0x455A14ED);

a=GG(a,b,c,d,x[k+13],S21,0xA9E3E905);

d=GG(d,a,b,c,x[k+2], S22,0xFCEFA3F8);

c=GG(c,d,a,b,x[k+7], S23,0x676F02D9);

b=GG(b,c,d,a,x[k+12],S24,0x8D2A4C8A);

a=HH(a,b,c,d,x[k+5], S31,0xFFFA3942);

d=HH(d,a,b,c,x[k+8], S32,0x8771F681);

c=HH(c,d,a,b,x[k+11],S33,0x6D9D6122);

b=HH(b,c,d,a,x[k+14],S34,0xFDE5380C);

a=HH(a,b,c,d,x[k+1], S31,0xA4BEEA44);

d=HH(d,a,b,c,x[k+4], S32,0x4BDECFA9);

c=HH(c,d,a,b,x[k+7], S33,0xF6BB4B60);

b=HH(b,c,d,a,x[k+10],S34,0xBEBFBC70);

a=HH(a,b,c,d,x[k+13],S31,0x289B7EC6);

d=HH(d,a,b,c,x[k+0], S32,0xEAA127FA);

c=HH(c,d,a,b,x[k+3], S33,0xD4EF3085);

b=HH(b,c,d,a,x[k+6], S34,0x4881D05);

a=HH(a,b,c,d,x[k+9], S31,0xD9D4D039);

d=HH(d,a,b,c,x[k+12],S32,0xE6DB99E5);

c=HH(c,d,a,b,x[k+15],S33,0x1FA27CF8);

b=HH(b,c,d,a,x[k+2], S34,0xC4AC5665);

a=II(a,b,c,d,x[k+0], S41,0xF4292244);

d=II(d,a,b,c,x[k+7], S42,0x432AFF97);

c=II(c,d,a,b,x[k+14],S43,0xAB9423A7);

b=II(b,c,d,a,x[k+5], S44,0xFC93A039);

a=II(a,b,c,d,x[k+12],S41,0x655B59C3);

d=II(d,a,b,c,x[k+3], S42,0x8F0CCC92);

c=II(c,d,a,b,x[k+10],S43,0xFFEFF47D);

b=II(b,c,d,a,x[k+1], S44,0x85845DD1);

a=II(a,b,c,d,x[k+8], S41,0x6FA87E4F);

d=II(d,a,b,c,x[k+15],S42,0xFE2CE6E0);

c=II(c,d,a,b,x[k+6], S43,0xA3014314);

b=II(b,c,d,a,x[k+13],S44,0x4E0811A1);

a=II(a,b,c,d,x[k+4], S41,0xF7537E82);

d=II(d,a,b,c,x[k+11],S42,0xBD3AF235);

c=II(c,d,a,b,x[k+2], S43,0x2AD7D2BB);

b=II(b,c,d,a,x[k+9], S44,0xEB86D391);

a=AddUnsigned(a,AA);

b=AddUnsigned(b,BB);

c=AddUnsigned(c,CC);

d=AddUnsigned(d,DD);

}

var temp = WordToHex(a)+WordToHex(b)+WordToHex(c)+WordToHex(d);

return temp.toLowerCase();

}

1. **Digital Signature**
   1. **Fungsi**

Sistem ini berfungsi untuk memberikan jaminan bahwa dokumen digital adalah otentik. Menambahkan sistem Digital signature proses transfer dokumen akan terjamin otentikasi dan kerahasiannya. Dengan demikian dapat menghindari kemungkinan terjadinya pemalsuan dokumen terkait.

* 1. **Cara Kerja**

Cara kerja Digital Signature dengan memanfaatkan dua buah kunci, yaitu kunci publik dan kunci privat. Kunci publik digunakan untuk mengenkripsi data, sedangkan kunci privat digunakan untuk mendekripsi data. Pertama, dokumen di-hash dan menghasilkan Message Digest. Kemudian, Message Digest dienkripsi oleh kunci publik menjadi Digital Signature. Untuk membuka Digital Signature tersebut diperlukan kunci privat. Bila data telah diubah oleh pihak luar, maka Digital Signature juga ikut berubah sehingga kunci privat yang ada tidak akan bisa membukanya. Ini merupakan salah satu syarat keaman jaringan, yaitu Authenticity. Artinya adalah, keaslian data dapat terjamin dari perubahan-perubahan yang dilakukan pihak luar. Dengan cara yang sama, pengirim data tidak dapat menyangkal data yang telah dikirimkannya. Bila Digital Signature cocok dengan kunci privat yang dipegang oleh penerima data, maka dapat dipastikan bahwa pengirim adalah pemegang kunci privat yang sama. Ini berarti Digital Signature memenuhi salah satu syarat keamanan jaringan, yaitu Nonrepudiation atau non-penyangkalan.

* 1. **Kelebihan & Kekurangan**

Kelemahan :

* Kunci harus dikirim melalui saluran yang aman. Kedua entitas yang berkomunikasi harus menjaga kerahasiaan kunci ini.
* Kunci harus sering diubah, mungkin pada setiap sesi komunikasi.

Kelebihan :

* Algoritma ini dirancang sehingga proses enkripsi/dekripsi membutuhkan waktu yang singkat.
* Ukuran kunci relatif lebih pendek.
* Algoritmanya bisa menghasilkan cipher(sebuah algoritma untuk menampilkan enkripsi dan kebalikannya dekripsi, serangkaian langkah yang terdefinisi yang diikuti sebagai prosedur) yang lebih kuat.
* Autentikasi pengiriman pesan langsung diketahui dari ciphertext yang diterima, karena kunci hanya diketahui oleh pengirim dan penerima pesan saja.

Kelemahan :

* Kunci harus dikirim melalui saluran yang aman. Kedua entitas yang berkomunikasi harus menjaga kerahasiaan kunci ini.
* Kunci harus sering diubah, mungkin pada setiap sesi komunikasi.

Kelebihan :

* Algoritma ini dirancang sehingga proses enkripsi/dekripsi membutuhkan waktu yang singkat.
* Ukuran kunci relatif lebih pendek.
* Algoritmanya bisa menghasilkan cipher(sebuah algoritma untuk menampilkan enkripsi dan kebalikannya dekripsi, serangkaian langkah yang terdefinisi yang diikuti sebagai prosedur) yang lebih kuat.
* Autentikasi pengiriman pesan langsung diketahui dari ciphertext yang diterima, karena kunci hanya diketahui oleh pengirim dan penerima pesan saja.

Kelemahan :

* Kunci harus dikirim melalui saluran yang aman. Kedua entitas yang berkomunikasi harus menjaga kerahasiaan kunci ini.
* Kunci harus sering diubah, mungkin pada setiap sesi komunikasi.
  1. **Source Code Aplikasi**

**Indeks.html :**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<!-- Always force latest IE rendering engine (even in intranet) & Chrome Frame

Remove this if you use the .htaccess -->

<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">

<meta name="viewport" content="initial-scale=1.0, target-densitydpi=device-dpi" /><!-- this is for mobile (Android) Chrome -->

<meta name="viewport" content="initial-scale=1.0, width=device-height"><!-- mobile Safari, FireFox, Opera Mobile -->

<style type="text/css">

div {

margin-top:1em;

margin-bottom:1em;

}

input {

padding: .5em;

margin: .5em;

}

select {

padding: .5em;

margin: .5em;

}

</style>

</head>

<body>

<div><a href="unmini.html">unmini.html</a></div>

<div><a href="noconflict.html">noconflict.html</a></div>

</body>

</html>

**noconflict.html :**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<!-- Always force latest IE rendering engine (even in intranet) & Chrome Frame

Remove this if you use the .htaccess -->

<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">

<meta name="viewport" content="initial-scale=1.0, target-densitydpi=device-dpi" /><!-- this is for mobile (Android) Chrome -->

<meta name="viewport" content="initial-scale=1.0, width=device-height"><!-- mobile Safari, FireFox, Opera Mobile -->

<script src="libs/modernizr.js"></script>

<!--[if lt IE 9]>

<script type="text/javascript" src="libs/flashcanvas.js"></script>

<![endif]-->

<style type="text/css">

div {

margin-top:1em;

margin-bottom:1em;

}

input {

padding: .5em;

margin: .5em;

}

select {

padding: .5em;

margin: .5em;

}

#signatureparent {

color:darkblue;

background-color:darkgrey;

/\*max-width:600px;\*/

padding:20px;

}

/\*This is the div within which the signature canvas is fitted\*/

#signature {

border: 2px dotted black;

background-color:lightgrey;

}

/\* Drawing the 'gripper' for touch-enabled devices \*/

html.touch #content {

float:left;

width:92%;

}

html.touch #scrollgrabber {

float:right;

width:4%;

margin-right:2%;

background-image:url()

}

html.borderradius #scrollgrabber {

border-radius: 1em;

}

</style>

</head>

<body>

<div>

<div id="content">

<div id="signatureparent">

<div>jSignature inherits colors from parent element. Text = Pen color. Background = Background. (This works even when Flash-based Canvas emulation is used.)</div>

<div id="signature"></div></div>

<div id="tools"></div>

<div><p>Display Area:</p><div id="displayarea"></div></div>

</div>

<div id="scrollgrabber"></div>

</div>

<script src="libs/jquery.js"></script>

<script type="text/javascript">

jQuery.noConflict()

</script>

<script>

/\* @preserve

jQuery pub/sub plugin by Peter Higgins (dante@dojotoolkit.org)

Loosely based on Dojo publish/subscribe API, limited in scope. Rewritten blindly.

Original is (c) Dojo Foundation 2004-2010. Released under either AFL or new BSD, see:

http://dojofoundation.org/license for more information.

\*/

(function($) {

var topics = {};

$.publish = function(topic, args) {

if (topics[topic]) {

var currentTopic = topics[topic],

args = args || {};

for (var i = 0, j = currentTopic.length; i < j; i++) {

currentTopic[i].call($, args);

}

}

};

$.subscribe = function(topic, callback) {

if (!topics[topic]) {

topics[topic] = [];

}

topics[topic].push(callback);

return {

"topic": topic,

"callback": callback

};

};

$.unsubscribe = function(handle) {

var topic = handle.topic;

if (topics[topic]) {

var currentTopic = topics[topic];

for (var i = 0, j = currentTopic.length; i < j; i++) {

if (currentTopic[i] === handle.callback) {

currentTopic.splice(i, 1);

}

}

}

};

})(jQuery);

</script>

<script src="libs/jSignature.min.noconflict.js"></script>

<script>

(function($){

$(document).ready(function() {

// This is the part where jSignature is initialized.

var $sigdiv = $("#signature").jSignature({'UndoButton':true})

// All the code below is just code driving the demo.

, $tools = $('#tools')

, $extraarea = $('#displayarea')

, pubsubprefix = 'jSignature.demo.'

var export\_plugins = $sigdiv.jSignature('listPlugins','export')

, chops = ['<span><b>Extract signature data as: </b></span><select>','<option value="">(select export format)</option>']

, name

for(var i in export\_plugins){

if (export\_plugins.hasOwnProperty(i)){

name = export\_plugins[i]

chops.push('<option value="' + name + '">' + name + '</option>')

}

}

chops.push('</select><span><b> or: </b></span>')

$(chops.join('')).bind('change', function(e){

if (e.target.value !== ''){

var data = $sigdiv.jSignature('getData', e.target.value)

$.publish(pubsubprefix + 'formatchanged')

if (typeof data === 'string'){

$('textarea', $tools).val(data)

} else if($.isArray(data) && data.length === 2){

$('textarea', $tools).val(data.join(','))

$.publish(pubsubprefix + data[0], data);

} else {

try {

$('textarea', $tools).val(JSON.stringify(data))

} catch (ex) {

$('textarea', $tools).val('Not sure how to stringify this, likely binary, format.')

}

}

}

}).appendTo($tools)

$('<input type="button" value="Reset">').bind('click', function(e){

$sigdiv.jSignature('reset')

}).appendTo($tools)

$('<div><textarea style="width:100%;height:7em;"></textarea></div>').appendTo($tools)

$.subscribe(pubsubprefix + 'formatchanged', function(){

$extraarea.html('')

})

$.subscribe(pubsubprefix + 'image/svg+xml', function(data) {

try{

var i = new Image()

i.src = 'data:' + data[0] + ';base64,' + btoa( data[1] )

$(i).appendTo($extraarea)

} catch (ex) {

}

var message = [

"If you don't see an image immediately above, it means your browser is unable to display in-line (data-url-formatted) SVG."

, "This is NOT an issue with jSignature, as we can export proper SVG document regardless of browser's ability to display it."

, "Try this page in a modern browser to see the SVG on the page, or export data as plain SVG, save to disk as text file and view in any SVG-capabale viewer."

]

$( "<div>" + message.join("<br/>") + "</div>" ).appendTo( $extraarea )

});

$.subscribe(pubsubprefix + 'image/svg+xml;base64', function(data) {

var i = new Image()

i.src = 'data:' + data[0] + ',' + data[1]

$(i).appendTo($extraarea)

var message = [

"If you don't see an image immediately above, it means your browser is unable to display in-line (data-url-formatted) SVG."

, "This is NOT an issue with jSignature, as we can export proper SVG document regardless of browser's ability to display it."

, "Try this page in a modern browser to see the SVG on the page, or export data as plain SVG, save to disk as text file and view in any SVG-capabale viewer."

]

$( "<div>" + message.join("<br/>") + "</div>" ).appendTo( $extraarea )

});

$.subscribe(pubsubprefix + 'image/png;base64', function(data) {

var i = new Image()

i.src = 'data:' + data[0] + ',' + data[1]

$('<span><b>As you can see, one of the problems of "image" extraction (besides not working on some old Androids, elsewhere) is that it extracts A LOT OF DATA and includes all the decoration that is not part of the signature.</b></span>').appendTo($extraarea)

$(i).appendTo($extraarea)

});

$.subscribe(pubsubprefix + 'image/jsignature;base30', function(data) {

$('<span><b>This is a vector format not natively render-able by browsers. Format is a compressed "movement coordinates arrays" structure tuned for use server-side. The bonus of this format is its tiny storage footprint and ease of deriving rendering instructions in programmatic, iterative manner.</b></span>').appendTo($extraarea)

});

if (Modernizr.touch){

$('#scrollgrabber').height($('#content').height())

}

})

})(jQuery)

</script>

</body>

</html>

**unmini.html :**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<!-- Always force latest IE rendering engine (even in intranet) & Chrome Frame

Remove this if you use the .htaccess -->

<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">

<meta name="viewport" content="initial-scale=1.0, target-densitydpi=device-dpi" /><!-- this is for mobile (Android) Chrome -->

<meta name="viewport" content="initial-scale=1.0, width=device-height"><!-- mobile Safari, FireFox, Opera Mobile -->

<script src="libs/modernizr.js"></script>

<!--[if lt IE 9]>

<script type="text/javascript" src="libs/flashcanvas.js"></script>

<![endif]-->

<style type="text/css">

div {

margin-top:1em;

margin-bottom:1em;

}

input {

padding: .5em;

margin: .5em;

}

select {

padding: .5em;

margin: .5em;

}

#signatureparent {

color:darkblue;

background-color:darkgrey;

/\*max-width:600px;\*/

padding:20px;

}

/\*This is the div within which the signature canvas is fitted\*/

#signature {

border: 2px dotted black;

background-color:lightgrey;

}

/\* Drawing the 'gripper' for touch-enabled devices \*/

html.touch #content {

float:left;

width:92%;

}

html.touch #scrollgrabber {

float:right;

width:4%;

margin-right:2%;

background-image:url()

}

html.borderradius #scrollgrabber {

border-radius: 1em;

}

</style>

</head>

<body>

<div>

<div id="content">

<div id="signatureparent">

<div>jSignature inherits colors from parent element. Text = Pen color. Background = Background. (This works even when Flash-based Canvas emulation is used.)</div>

<div id="signature"></div></div>

<div id="tools"></div>

<div><p>Display Area:</p><div id="displayarea"></div></div>

</div>

<div id="scrollgrabber"></div>

</div>

<script src="libs/jquery.js"></script>

<script>

/\* @preserve

jQuery pub/sub plugin by Peter Higgins (dante@dojotoolkit.org)

Loosely based on Dojo publish/subscribe API, limited in scope. Rewritten blindly.

Original is (c) Dojo Foundation 2004-2010. Released under either AFL or new BSD, see:

http://dojofoundation.org/license for more information.

\*/

(function($) {

var topics = {};

$.publish = function(topic, args) {

if (topics[topic]) {

var currentTopic = topics[topic],

args = args || {};

for (var i = 0, j = currentTopic.length; i < j; i++) {

currentTopic[i].call($, args);

}

}

};

$.subscribe = function(topic, callback) {

if (!topics[topic]) {

topics[topic] = [];

}

topics[topic].push(callback);

return {

"topic": topic,

"callback": callback

};

};

$.unsubscribe = function(handle) {

var topic = handle.topic;

if (topics[topic]) {

var currentTopic = topics[topic];

for (var i = 0, j = currentTopic.length; i < j; i++) {

if (currentTopic[i] === handle.callback) {

currentTopic.splice(i, 1);

}

}

}

};

})(jQuery);

</script>

<script src="src/jSignature.js"></script>

<script src="src/plugins/jSignature.CompressorBase30.js"></script>

<script src="src/plugins/jSignature.CompressorSVG.js"></script>

<script src="src/plugins/jSignature.UndoButton.js"></script>

<script src="src/plugins/signhere/jSignature.SignHere.js"></script>

<script>

$(document).ready(function() {

// This is the part where jSignature is initialized.

var $sigdiv = $("#signature").jSignature({'UndoButton':true})

// All the code below is just code driving the demo.

, $tools = $('#tools')

, $extraarea = $('#displayarea')

, pubsubprefix = 'jSignature.demo.'

var export\_plugins = $sigdiv.jSignature('listPlugins','export')

, chops = ['<span><b>Extract signature data as: </b></span><select>','<option value="">(select export format)</option>']

, name

for(var i in export\_plugins){

if (export\_plugins.hasOwnProperty(i)){

name = export\_plugins[i]

chops.push('<option value="' + name + '">' + name + '</option>')

}

}

chops.push('</select><span><b> or: </b></span>')

$(chops.join('')).bind('change', function(e){

if (e.target.value !== ''){

var data = $sigdiv.jSignature('getData', e.target.value)

$.publish(pubsubprefix + 'formatchanged')

if (typeof data === 'string'){

$('textarea', $tools).val(data)

} else if($.isArray(data) && data.length === 2){

$('textarea', $tools).val(data.join(','))

$.publish(pubsubprefix + data[0], data);

} else {

try {

$('textarea', $tools).val(JSON.stringify(data))

} catch (ex) {

$('textarea', $tools).val('Not sure how to stringify this, likely binary, format.')

}

}

}

}).appendTo($tools)

$('<input type="button" value="Reset">').bind('click', function(e){

$sigdiv.jSignature('reset')

}).appendTo($tools)

$('<div><textarea style="width:100%;height:7em;"></textarea></div>').appendTo($tools)

$.subscribe(pubsubprefix + 'formatchanged', function(){

$extraarea.html('')

})

$.subscribe(pubsubprefix + 'image/svg+xml', function(data) {

try{

var i = new Image()

i.src = 'data:' + data[0] + ';base64,' + btoa( data[1] )

$(i).appendTo($extraarea)

} catch (ex) {

}

var message = [

"If you don't see an image immediately above, it means your browser is unable to display in-line (data-url-formatted) SVG."

, "This is NOT an issue with jSignature, as we can export proper SVG document regardless of browser's ability to display it."

, "Try this page in a modern browser to see the SVG on the page, or export data as plain SVG, save to disk as text file and view in any SVG-capabale viewer."

]

$( "<div>" + message.join("<br/>") + "</div>" ).appendTo( $extraarea )

});

$.subscribe(pubsubprefix + 'image/svg+xml;base64', function(data) {

var i = new Image()

i.src = 'data:' + data[0] + ',' + data[1]

$(i).appendTo($extraarea)

var message = [

"If you don't see an image immediately above, it means your browser is unable to display in-line (data-url-formatted) SVG."

, "This is NOT an issue with jSignature, as we can export proper SVG document regardless of browser's ability to display it."

, "Try this page in a modern browser to see the SVG on the page, or export data as plain SVG, save to disk as text file and view in any SVG-capabale viewer."

]

$( "<div>" + message.join("<br/>") + "</div>" ).appendTo( $extraarea )

});

$.subscribe(pubsubprefix + 'image/png;base64', function(data) {

var i = new Image()

i.src = 'data:' + data[0] + ',' + data[1]

$('<span><b>As you can see, one of the problems of "image" extraction (besides not working on some old Androids, elsewhere) is that it extracts A LOT OF DATA and includes all the decoration that is not part of the signature.</b></span>').appendTo($extraarea)

$(i).appendTo($extraarea)

});

$.subscribe(pubsubprefix + 'image/jsignature;base30', function(data) {

$('<span><b>This is a vector format not natively render-able by browsers. Format is a compressed "movement coordinates arrays" structure tuned for use server-side. The bonus of this format is its tiny storage footprint and ease of deriving rendering instructions in programmatic, iterative manner.</b></span>').appendTo($extraarea)

});

if (Modernizr.touch){

$('#scrollgrabber').height($('#content').height())

}

})

</script>

</body>

</html>