SKRIPSI

PERLINDUNGAN *PASSWORD* DENGAN ENTROPI PERSONAL

SAMUEL CHRISTIAN

NPM: 2011730002

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS
UNIVERSITAS KATOLIK PARAHYANGAN
2015

UNDERGRADUATE THESIS

PROTECTING SECRET KEYS WITH PERSONAL ENTROPY

SAMUEL CHRISTIAN

NPM: 2011730002

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES
PARAHYANGAN CATHOLIC UNIVERSITY
2015

LEMBAR PENGESAHAN

PERLINDUNGAN *PASSWORD* DENGAN ENTROPI PERSONAL

SAMUEL CHRISTIAN

NPM: 2011730002

Bandung, 1 April 2015

Menyetujui,

Pembimbing Utama	Pembimbing Pendamping
Mariskha Tri Aditia, PDEng	

Ketua Tim Penguji	Anggota Tim Penguji
Plato	**Euclid**

Mengetahui,

Ketua Program Studi

Thomas Anung Basuki, Ph.D.

PERNYATAAN

Dengan ini saya yang bertandatangan di bawah ini menyatakan bahwa skripsi dengan judul:

PERLINDUNGAN *PASSWORD* DENGAN ENTROPI PERSONAL

adalah benar-benar karya saya sendiri, dan saya tidak melakukan penjiplakan atau pengutipan dengan cara-cara yang tidak sesuai dengan etika keilmuan yang berlaku dalam masyarakat keilmuan.

Atas pernyataan ini, saya siap menanggung segala risiko dan sanksi yang dijatuhkan kepada saya, apabila di kemudian hari ditemukan adanya pelanggaran terhadap etika keilmuan dalam karya saya, atau jika ada tuntutan formal atau non-formal dari pihak lain berkaitan dengan keaslian karya saya ini.

Dinyatakan di Bandung,
Tanggal 1 April 2015

Meterai

Samuel Christian
NPM: 2011730002

ABSTRAK

Abstrak indo

Kata-kata kunci: Lintasan, Homotopy, Fréchet, Lintasan Median, Penyangga

ABSTRACT

Abstrak inggris

Keywords: Trajectory, Homotopy, Fréchet, Median Trajectory, Buffer

Dipersembahkan untuk Zorg, the evil lord, just kidding LOL =P

KATA PENGANTAR

haleluya

Bandung, April 2015

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Enkripsi adalah proses merahasiakan sebuah informasi dengan cara menyandikan informasi tersebut sehingga informasi tersebut tidak dapat dibaca oleh pihak yang tidak berwenang. Dalam proses enkripsi, dibutuhkan sebuah kunci rahasia (*private key*), untuk menyandikan informasi sehingga tidak bisa dibaca dan untuk mengembalikan informasi sehingga bisa kembali dibaca. Proses enkripsi ini memiliki kelemahan, yaitu jika kunci yang digunakan untuk enkripsi hilang maka berakibat informasi yang dienkripsi tidak bisa dikembalikan seperti semula.

Pada umumnya, untuk mengembalikan kunci yang hilang ini, beberapa sistem memiliki mekanisme dengan menyediakan sebuah pertanyaan keamanan yang pertanyaan dan jawabannya sudah dirancang oleh pengguna. Jika pengguna menjawab pertanyaan keamanan ini dengan benar maka pengguna bisa mendapatkan kembali kunci yang hilang. Tetapi,

1.2 Rumusan Masalah

Rumusan masalah pada penelitian ini berupa:

- Bagaimana cara melindungi *password* dengan *secret sharing* shamir?
- Bagaimana cara mengimplementasikan *secret sharing* shamir pada perangkat lunak?

1.3 Tujuan

Tujuan penelitian ini berupa:

- Mempelajari cara kerja *secret sharing* shamir dalam melindungi *password*.
- Membangun perangkat lunak yang mengimplementasikan *secret sharing* shamir

1.4 Batasan Masalah

Batasan masalah pada penelitian ini berupa:

- Setiap pertanyaan selalu dijawab dengan jawaban yang relevan dengan pertanyaan.

1.5 Metodologi Penelitian

Metodologi dalam penelitian ini berupa:

- Melakukan studi literatur mengenai *secret sharing* shamir
- Melakukan studi literatur mengenai algoritma enkripsi *data encryption standard* (DES)
- Melakukan studi literatur mengenai *secure-hash-algorithm-512* (SHA-512)
- Melakukan analisis dan perancangan mengenai perangkat lunak yang akan dibangun
- Implementasi terhadap hasil analisis dan perancangan perangkat lunak
- Melakukan pengujian perangkat lunak

1.6 Sistematika Pembahasan

Sistematika pembahasan dalam penelitian ini berupa:

- Bab Pendahuluan
  Bab 1 berisi latar belakang, rumusan masalah, tujuan penelitian, batasan masalah, metodologi penelitian, dan sistematika pembahasan.
- Bab Dasar Teori
  Bab 2 berisi mengenai teori-teori dasar, antara lain kriptografi, algoritma enkripsi, algoritma fungsi *hash*, otentikasi, *secret sharing*, probabilitas, dan entropi.
- Bab Analisis
  Bab 3 berisi analisis meliputi perhitungan, proses, *flow chart*, *use case*, skenario, diagram aktivitas, dan rancangan awal diagram kelas.
- Bab Perancangan
  Bab 4 berisi tahapan penjelasan rancangan perangkat lunak meliputi algoritma, kelas diagram lengkap, dan rancangan tampilan perangkat lunak.
- Bab Implementasi dan Pengujian
  Bab 5 berisi tahapan implementasi pada perangkat lunak meliputi tampilan dari perangkat lunak, pengujian terdiri dari kasus, skenario, dan hasil observasi, dan kesimpulan
- Bab Kesimpulan dan Saran
  Bab 6 berisi kesimpulan serta beberapa saran untuk pengembangan lebih lanjut dari penelitian yang dilakukan dan perangkat lunak yang dibangun.

BAB 2

DASAR TEORI

2.1 Kriptografi

Kriptografi merupakan kata berasal dari bahasa Yunani, yaitu kripto dan graphia. Kripto berarti rahasia dan graphia berarti tulisan. Jadi, kriptografi adalah ilmu sekaligus seni untuk untuk menjaga keamanan pesan. Keamanan pesan diperoleh dengan menyandikan pesan menjadi tidak bermakna atau tidak memiliki arti. Zaman sekarang ini, kerahasiaan informasi menjadi suatu hal yang penting. Informasi yang sifatnya rahasia atau personal perlu dilindungi dari orang-orang yang tidak berhak untuk membacanya. Kriptografi digunakan untuk menyamarkan informasi rahasia itu dari orang atau pihak yang tidak berhak untuk membaca atau melihatnya.

Kriptografi memiliki 4 layanan utama:

1. Kerahasiaan Data (*data confidentiality*)
   Layanan ini menjamin bahwa data atau pesan yang dikirimkan tidak diketahui oleh pihak atau orang lain yang tidak berhak untuk membaca atau melihatnya.

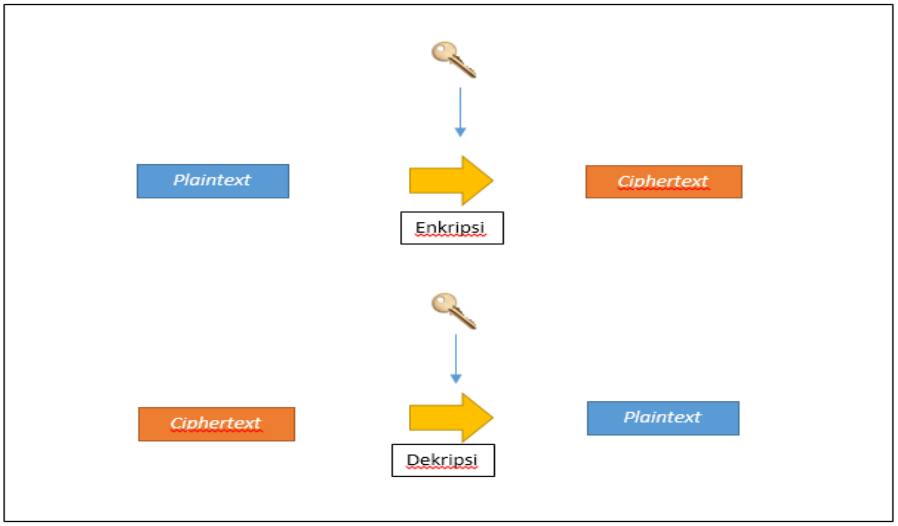
2. Integritas Data (*data integrity*)
   Layanan yang menjamin data atau pesan yang dikirimkan tidak boleh diubah tanpa seijin pemilik pesan, menjamin keaslian dari data atau pesan yang dikirimkan.

3. Otentikasi (*authentication*)
   Layanan yang digunakan untuk menvalidasi identitas seseorang, entitas, atau asal informasi yang dikirim. Autentikasi dibagi ke dalam 2 jenis dilihat dari subjek yang diotentikasinya, yaitu otentikasi entitas dan otentikasi pesan.

4. Non-repudiasi (*nonrepudiation*)
   Layanan yang menjamin bahwa tidak ada penyangkalan baik oleh pengirim atau penerima pesan.

Dalam kriptografi, pesan yang dirahasiakan disebut *plaintext*, sedangkan pesan hasil dari penyandian disebut *ciphertext*. Pesan yang telah disandikan dapat dikembalikan lagi ke pesan aslinya hanya oleh orang yang berhak. Orang yang berhak adalah orang yang mengetahui cara penyandian atau memiliki kunci penyandian. Proses menyandikan *plaintext* menjadi *ciphertext* disebut enkripsi (*encryption*) dan proses mengembalikan *ciphertext* menjadi *plaintext* disebut dekripsi (*decryption*). Dalam proses enkripsi dan dekripsi menggunakan kunci (*key*) yaitu sekumpulan huruf, angka, atau

simbol. Kunci ini sifatnya rahasia dan tidak boleh diketahui pihak lain yang tidak berwenang. Diagram dibawah ini menunjukkan proses enkripsi dan dekripsi.



Gambar 2.1: Proses enkripsi dan dekripsi

Dalam proses enkripsi, *plaintext* akan dipetakan pada fungsi enkripsi E menjadi *ciphertext* didasarkan pada kunci k seperti notasi di bawah ini:

$$E_k(plaintext) = ciphertext$$

Kemudian dalam proses dekripsi, untuk mengembalikan *ciphertext* ke *plaintext* maka *ciphertext* akan dipetakan pada fungsi dekripsi D didasarkan pada kunci k seperti notasi di bawah ini:

$$D_k(ciphertext) = plaintext$$

Sebagai contoh, *plaintext* yang akan dikirimkan sebagai berikut adalah kriptografi disandikan menjadi *ciphertext* dengan fungsi enkripsi E didasarkan pada kunci k menjadi:

$$E_k(kriptografi) = u8@37md$$

ciphertext diatas, meskipun tidak dirahasiakan, namun isinya sudah tidak jelas dan tidak dapat dimengerti maksudnya. Hanya orang yang berhak yang dapat mengembalikan *ciphertext* menjadi *plaintext*. Kemudian untuk mengembalikan *ciphertext* ke *plaintext*, *ciphertext* akan dipetakan pada fungsi dekripsi D didasarkan pada kunci k menjadi:

$$D_k(u8@37md) = kriptografi$$

Algoritma kriptografi ini dibagi menjadi 2 jenis, yaitu:

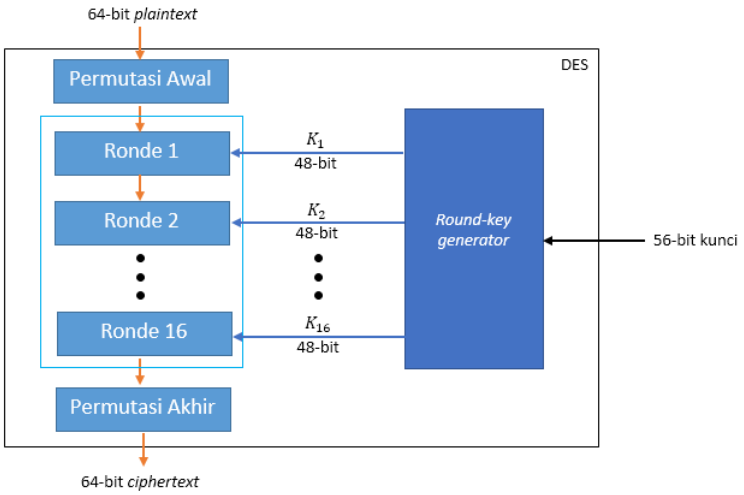
1. Kriptografi kunci simetris (*symmetric key cryptography*)
   Kriptografi kunci simetris atau penyandian kunci simetris menggunakan hanya satu kunci

rahasia. Pengirim pesan mengenkripsi pesan dengan kunci k, kemudian penerima pesan juga akan mendekripsi pesan yang diterima dengan kunci k. Dalam hal ini, pengirim dan penerima keduanya harus memiliki kunci k. Kelemahan dari kriptografi kunci simetris adalah baik pengirim dan penerima harus memiliki kunci yang sama, sehingga pengirim harus mencara cara lain untuk memberitahukan kunci kepada penerima. Contoh algoritma kunci simetris antara lain adalah *Data Encryption Standard* (DES), *Advanced Encryption Standard* (AES), *Twofish*, dan *Blowfish*.

2. Kriptografi kunci asimetris (*asymmetric key cryptography*)
   Dalam kriptografi kunci asimetris atau penyandian kunci simetris kunci rahasia yang digunakan ada 2 buah kunci, yaitu kunci publik (*public key*) dan kunci pribadi (*private key*). Kunci publik tidak rahasia dan dapat diketahui secara umum, sedangkan kunci pribadi harus dirahasiakan oleh pengirim pesan. Pengirim pesan akan mengenkripsi pesan yang dikirim dengan kunci publik penerima pesan, kemudian penerima pesan akan mendekripsi pesan menggunakan kunci pribadinya yang hanya diketahui oleh dirinya saja. Contoh algoritma kunci asimetris antara lain adalah Rivest-Shamir-Adleman (RSA), ElGamal, Diffie-Helman, *Digital Signature Algorithm*, dan *Elliptic Curve Digital Signature Algorithm* (ECDSA).

2.1.1 *Data Encryption Standard* (DES)

Data Encryption Standard (DES) adalah algoritma kriptografi kunci simetris, yaitu menggunakan kunci yang sama pada proses enkripsi dan dekripsinya. Masukkan dari DES berupa 64-*bit plaintext* dan keluarannya berupa 64-*bit ciphertext* dengan 64-*bit* kunci. Proses enkripsi terdiri dari 2 proses permutasi, yaitu permutasi awal dan permutasi akhir, dan 16 ronde sandi feistel. Setiap ronde menggunakan kunci 48-*bit* yang berbeda-beda. Kunci dari setiap ronde ini akan didapat dari round-key generator yang berdasarkan pada 64-*bit* kunci sandi. Gambar di bawah ini menunjukkan proses enkripsi dari DES.



Gambar 2.2: Proses enkripsi dengan DES

Permutasi Awal dan Permutasi Akhir

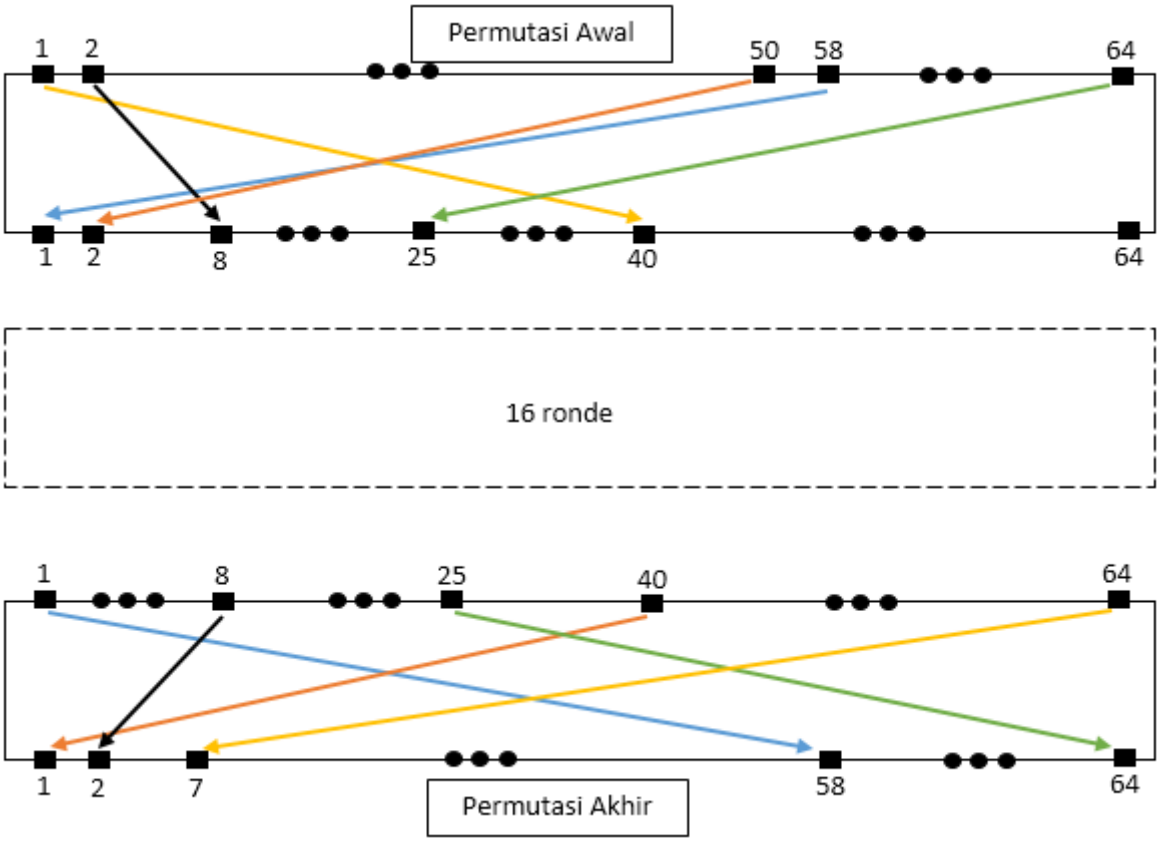
Permutasi awal dan akhir dalam DES menggunakan matriks permutasi. Kedua permutasi ini menggunakan 64-*bit* masukan dan matriks yang terdiri 64 nilai yang sudah ditentukan sebelumnya. Matriks ini nilainya selalu sama untuk setiap proses enkripsi dan tidak ada aturan tertentu untuk membuatnya. Gambar di bawah ini menunjukkan salah satu contoh matriks permutasi awal dan matriks permutasi akhir.

Permutasi Awal							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Permutasi Akhir							
40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Gambar 2.3: Matriks Permutasi

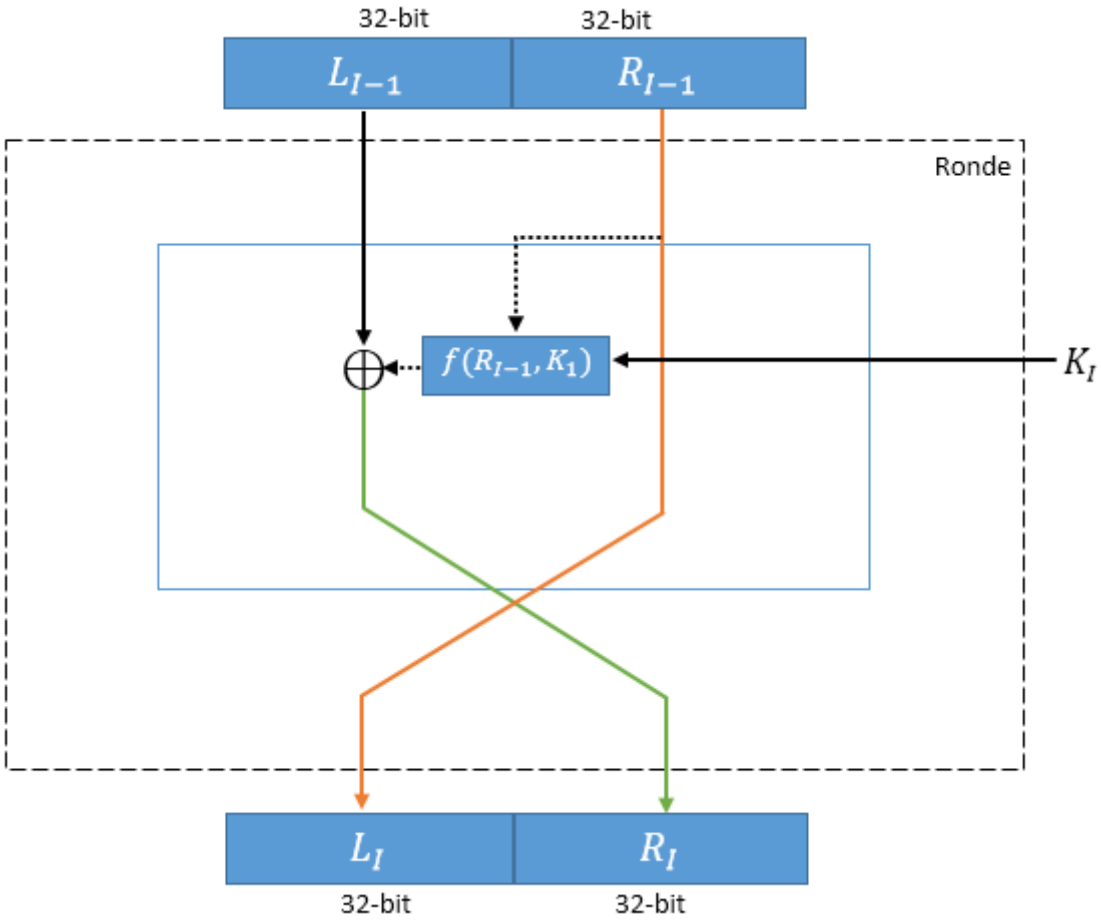
Sebagai contoh, *bit* ke-58 dari input akan menjadi *bit* ke-1 pada output permutasi awal, *bit* ke-50 akan menjadi *bit* ke-2 pada output permutasi awal dan seterusnya. Cara yang sama akan diterapkan pada proses permutasi akhir nanti setelah 16 ronde sandi feistel.



Gambar 2.4: Proses Permutasi

Ronde

DES menggunakan 16 ronde. Setiap ronde dari DES adalah sandi feistel yang ditunjukkan pada gambar di bawah.



Gambar 2.5: Ronde dalam DES

Setiap ronde menggunakan L_{I-1} dan R_{I-1} dari ronde sebelumnya (atau dari permutasi awal) dan memrosesnya menjadi L_I dan R_I untuk nanti dilanjutkan ke proses yang berikutnya (atau permutasi akhir). $f(R_{I-1}, K_I)$ merupakan fungsi DES. Plaintext (64-*bit*) akan dibagi 2 menjadi bagian kiri (L) dan bagian kanan (R). Bagian kanan akan diproses lagi oleh fungsi $f(R_{I-1}, K_I)$. Kemudian, hasil dari fungsi $f(R_{I-1}, K_I)$ akan di XOR dengan bagian kiri. Hasil dari XOR akan menjadi R_I dan hasil dari fungsi $f(R_{I-1}, K_I)$ akan menjadi L_I. Pada ronde ke-16 tidak akan terjadi pertukaran antara bagian L_{I-1} dan R_{I-1}.

Fungsi DES

Fungsi DES merupakan inti dari algoritma DES. Fungsi DES menerima masukkan kunci ronde 48-*bit* dan bagian kanan dari plaintext, R_{I-1} dan menghasilkan 32-*bit* keluaran. Fungsi ini terdiri dari 4 bagian, yaitu ekspansi *P-box*, operasi XOR, substitusi *S-box*, dan permutasi.

1. Ekspansi *P-box*
   Pada bagian ini, karena R_{I-1} adalah input dengan panjang 32-*bit* dan kunci ronde yang

digunakan sepanjang 48-*bit*, maka R_{I-1} perlu diekspansi menjadi 48-*bit*. R_{I-1} akan dibagi menjadi 8 bagian masing-masing 4-*bit*. Kemudian setiap bagian 4-*bit* ini akan diekspansi menjadi 6-*bit*. Ekspansi ini dilakukan berdasarkan aturan yang sudah ditentukan terlebih dahulu, yaitu

(a) *Bit* keluaran ke-2, 3, 4, dan 5 diisi oleh *bit* masukkan ke-1, 2, 3, dan 4.

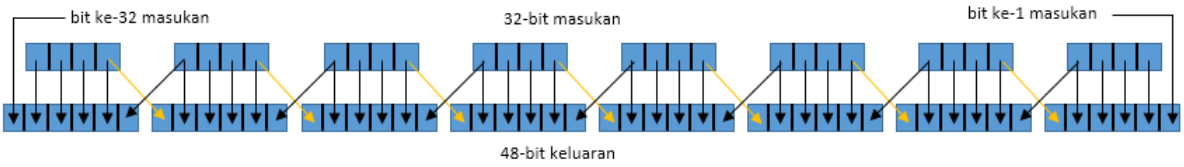
(b) *Bit* keluaran ke-1 akan diisi oleh *bit* masukkan ke-4 pada bagian sebelumnya.

(c) *Bit* keluaran ke-6 akan diisi oleh *bit* masukkan ke-1 pada bagian sesudahnya.

(d) Untuk bagian ke-1, *bit* keluaran ke-1 akan diisi oleh *bit* ke-32 masukkan.

(e) Untuk bagian ke-8, *bit* keluaran ke-6 akan diisi oleh *bit* ke-1 masukkan.

Ekspansi ini dilakukan berdasarkan nilai-nilai yang ada dalam *P-box*. Gambar di bawah ini menunjukkan proses ekspansi dan *P-box*.



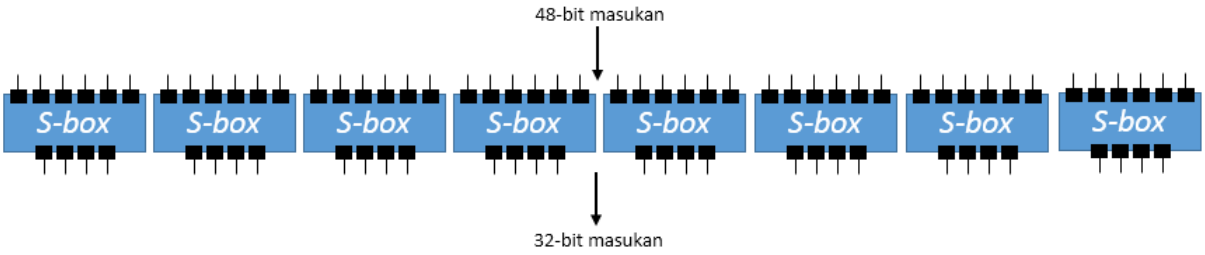
Gambar 2.6: Proses permutasi ekspansi

P-box					
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	31	31	32	1

Gambar 2.7: *P-box*

2. Operasi XOR
   Setelah dilakukan ekspansi dengan menggunakan *P-box*, dilakukan operasi XOR antara R_{I-1} dengan kunci ronde. Perlu diketahui, bahwa panjang R_{I-1} dan kunci ronde sudah sama, yaitu 48-*bit* dan kunci ronde hanya digunakan dalam proses ini saja.

3. Substitusi *S-box*
   DES menggunakan 8 buah *S-box*, masing-masing dengan 6-*bit* masukkan dan 4-*bit* keluaran. Gambar di bawah ini menunjukkan proses substitusi *S-box* dan salah satu contoh dari *S-box*.



Gambar 2.8: Proses substitusi *S-box*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	10	3	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Gambar 2.9: *S-box*

Pada bagian ini, akan dilakukan operasi substitusi berdasarkan aturan pada *S-box*. Setiap *S-box* adalah matriks berukuran 4x16 (4 baris, 16 kolom). Setiap isi dari *S-box* akan berbeda-beda. Kombinasi biner dari *bit* ke-1 dan ke-6 masukan menunjukkan posisi baris yang akan dipilih dan kombinasi biner dari *bit* masukan sisanya menunjukkan posisi kolom yang akan dipilih. Setelah itu, angka yang ditunjuk oleh baris dan kolom kombinasi biner ini akan diubah juga menjadi biner.

Sebagai contoh, misalkan masukan untuk *S-box* diatas adalah 110011. Maka, baris yang dipilih adalah 11 dalam biner dan 3 dalam desimal dan kolom yang dipilih adalah 1001 dalam biner dan 9 dalam desimal. Berarti, nilai yang didapat berdasarkan *S-box* diatas adalah 11 dan dalam biner adalah 1011. Maka, keluaran dari 110011 adalah 1011.

4. Permutasi
   Bagian terakhir dari fungsi DES adalah permutasi langsung dengan masukan 32-*bit* dan keluaran 32-*bit*. Proses nya sama dengan permutasi yang dilakukan pada awal proses enkripsi tetapi ukuran matriks permutasi 4x8 (4 baris, 8 kolom). Gambar di bawah ini menunjukkan matriks permutasi langsung.

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

Gambar 2.10: Matriks Permutasi Langsung

Sama halnya seperti proses permutasi yang dilakukan di awal enkripsi, angka 16 menunjukkan bahwa *bit* ke-16 masukan akan menjadi *bit* ke-1 keluaran, *bit* ke-7 masukan akan menjadi *bit* ke-2 keluaran, dan seterusnya.

2.1.2 Pembangunan Kunci Ronde

Pembangunan kunci ronde adalah algoritma untuk membentuk kunci yang akan digunakan pada setiap ronde. Pembangunan kunci ronde akan menghasilkan 16 kunci masing-masing dengan panjang 48-*bit* dari masukan 56-*bit* kunci sandi. Tetapi, sebenarnya panjang sandi kunci ini adalah 64-*bit* dengan 8-*bit parity bit*. *Parity bit* akan dihilangkan sebelum proses pembangkitan kunci sandi 56-*bit*. Proses pembangkitan kunci sandi 56-*bit* bisa dilihat pada gambar di bawah.

Gambar 2.11: *Round-Key Generator*

Parity Drop

Proses yang dilakukan sebelum pembangkitan kunci sandi adalah penghilangan *parity bit* pada kunci sandi 64-*bit*. Bit yang dihilangkan adalah *bit* ke-8, 16, 24, 32, Ĕ, 64 sehingga hasil akhirnya adalah 56-*bit* sandi kunci yang nanti kunci ini akan digunakan untuk membuat kunci setiap ronde. Matriks permutasi yang digunakan pada proses ini ditunjukkan pada gambar di bawah.

57	49	41	33	25	17	9	1
58	50	42	34	26	18	10	2
59	51	43	35	27	19	11	3
60	52	44	36	63	55	47	39
31	23	15	7	62	54	46	38
30	22	14	6	61	53	45	37
29	21	13	5	28	20	12	4

Gambar 2.12: Matriks permutasi untuk *Parity drop*

Shift Left

Pada tahap ini, kunci yang akan dibentuk akan dipisah menjadi 2 bagian masing-masing 28-*bit*. Setiap bagian akan digeser ke arah kiri secara sirkular sebanyak 1 atau 2 *bit*. Ronde ke-1, 2, 9, dan 16 akan digeser sebanyak 1 *bit*, selain itu hanya akan digeser sebanyak 2 *bit*. Kemudian, kedua bagian ini akan disatukan kembali menjadi satu bagian dengan panjang 56-*bit*.

Kompresi *P-box*

Kompresi dengan *P-box* adalah tahap permutasi untuk mengubah kunci 56-*bit* menjadi 48-*bit* agar bisa digunakan sebagai ronde kunci. Kompresi ini menggunakan matriks permutasi *P-box* dengan ukuran 6x8 seperti gambar yang ditunjukkan di bawah ini.

14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
32	29	36	50	42	46	53	34

Gambar 2.13: Matriks kompresi *P-box*

2.2 Fungsi *Hash*

Fungsi *hash* adalah algoritma kriptografi yang membuat kompresi dari pesan (*message*) atau inti dari pesan, dimana kompresi ini dapat berfungsi sebagai *fingerprint* atau disebut juga *digest*. Fungsi *hash* juga berfungsi untuk menjaga integritas sebuah pesan agar ketika ada perubahan pada pesan tersebut dapat langsung diketahui. Fungsi *hash* akan memroses message (m) menjadi *message digest* atau *digest* (h). Bentuk umum dari fungsi *hash*, yaitu:

$$h = H(m)$$

Fungsi *hash* memiliki 4 kriteria utama, yaitu:

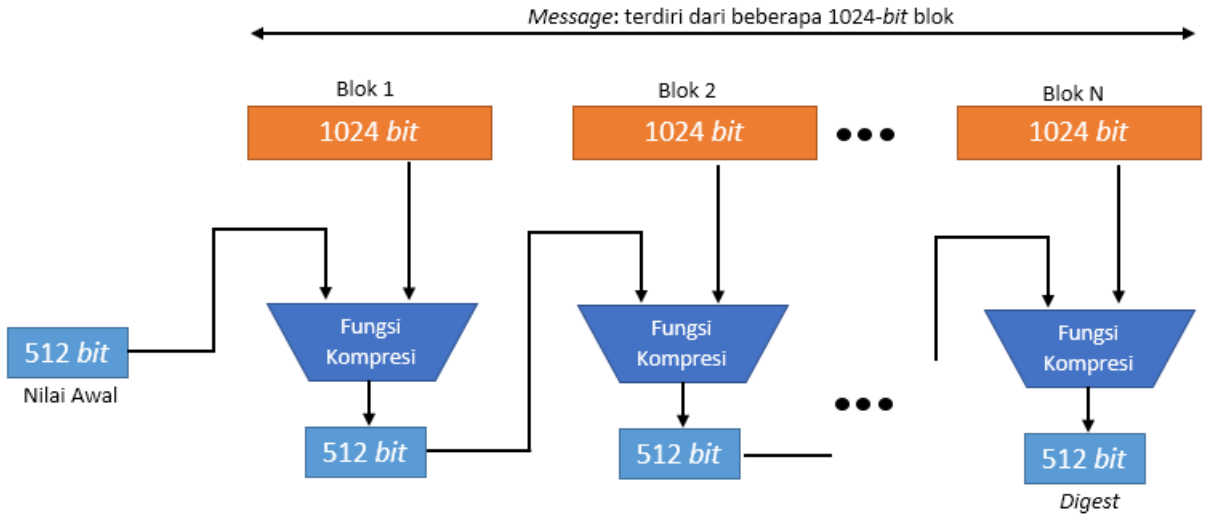
- Semua nilai hash memiliki panjang yang sama.
- Mudah untuk menghitung nilai *hash* untuk setiap *message* masukkan.

- Nilai *hash* tidak bisa dikembalikan menjadi *message*.
- Tidak ada 2 atau lebih *message* yang memiliki nilai *hash* yang sama.
- Saat *message* berubah maka nilai hashnya juga akan berubah.

Contoh fungsi *hash* antara lain SHA-0, SHA-1, SHA-512, MD-2, dan MD-5.

2.2.1 *Secure Hashing Algorithm* 512 (SHA-512)

Secure hashing algorithm 512 atau SHA-512 adalah algoritma fungsi *hash* yang menghasilkan 512-*bit message digest*. SHA-512 merupakan algoritma fungsi *hash* dalam versi SHA yang menghasilkan *message digest* paling panjang (512-*bit*). SHA-512 membuat 512-*bit* digest yang diambil dari beberapa blok *message*. Setiap blok *message* panjangnya 1024 bit. Gambar di bawah ini menunjukkan cara pembuatan *message digest*.



Gambar 2.14: Pembuatan *message digest*

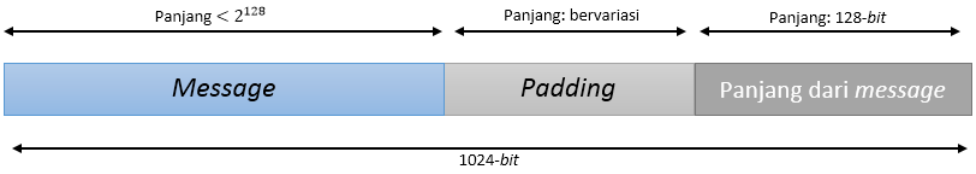
Persiapan *Message*

SHA-512 menerima masukan *message* dengan panjang kurang dari $2^{128} - bit$, berarti jika panjang *message* sama dengan 2^{128} atau lebih maka SHA-512 tidak memroses *message* tersebut. Hal ini tidak akan menjadi masalah karena ukuran $2^{128} - bit$ cukup besar untuk tempat penyimpanan komputer manapun.

Length Field dan *Padding*

Sebelum *message digest* atau *digest* dapat dibuat, SHA-512 membutuhkan tambahan 128-*bit* yang menunjukkan panjang dari *message*. Panjang dari *message* ini direpresentasikan oleh 128-*bit* tambahan.

Sebelum menambahkan bagian 128-*bit* yang menunjukkan panjang dari *message* maka kita perlu melapisi (*padding*) *message* asli agar panjang totalnya bisa mencapai 1024-*bit*. Panjang dari bagian



Gambar 2.15: *Length field* dan *padding* dalam SHA-512

padding yang harus ditambahkan ini ditunjukkan dengan cara sebagai berikut, dengan P adalah *padding*, M adalah *message*.

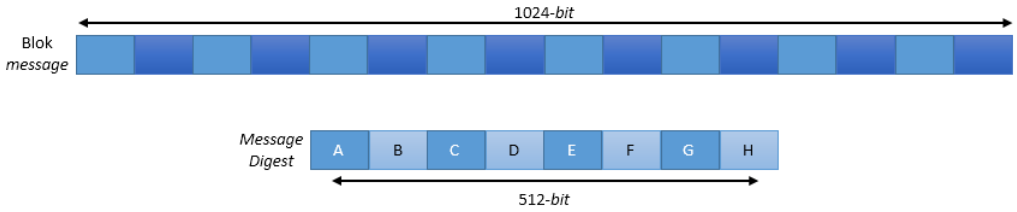
$$(M + P + 128) = 0 \mod 1024$$

$$P = (-M - 128) \mod 1024$$

Cara menuliskan *padding* diawali dengan angka 1 kemudian diikuti oleh beberapa angka 0 (nol) sampai panjangnya mencapai P.

Words

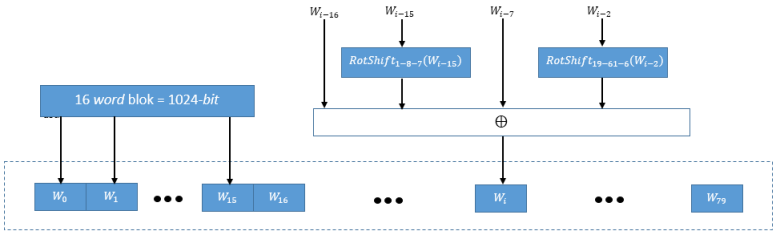
SHA-512 beroperasi dengan ukuran *words*. Panjang dari satu *word* adalah 64-*bit*. Ini berarti, setelah proses *padding* ditambahkan pada *message*, setiap blok *message* akan terdiri dari 16 buah 64-*bit word*. *Message digest* atau *digest* juga akan terdiri dari 64-*bit word* namun panjangnya hanya 8 *word* dan setiap *word* nya dinamakan A, B, C, D, E, F, G, dan H.



Gambar 2.16: Blok *message* dan *message digest* dalam *word*

Ekspansi *Word*

Sebelum diproses, setiap blok *message* harus diekspansi, panjang setiap blok ini adalah 1024-*bit* terdiri dari 16 buah 64-*bit word*. Pada tahap pemrosesan, dibutuhkan 80 buah 64-*bit word* maka dari itu, 16 *word* ini harus diekspansi menjadi 80 *word*, W_0 sampai W_79. Gambar di bawah ini akan menunjukkan cara ekspansi *word*.



Gambar 2.17: Ekspansi *word*

$$RotShift_{l-m-n}(x) : RotR_l(x) \oplus RotR_m(x) \oplus ShL_n(x)$$

$RotR_i(x)$: Rotasi ke kanan dari x sebanyak i *bit*.
$ShL_i(x)$: *Shift left* x sebanyak i *bit* ditambah (*padding*) dengan 0.

Inisialisasi *Message Digest*

SHA-512 menggunakan 8 buah konstanta dalam proses insialisasi *message digest*. Konstanta ini akan diberi nama A_0 sampai H_0. Setiap nilai dari konstanta didapat dari 8 bilangan prima pertama (2, 3, 5, 7, 11, 13, 17, 19). Setiap nilai merupakan nilai pecahan atau angka di belakang koma dari akar kuadrat bilangan prima yang bersangkutan.

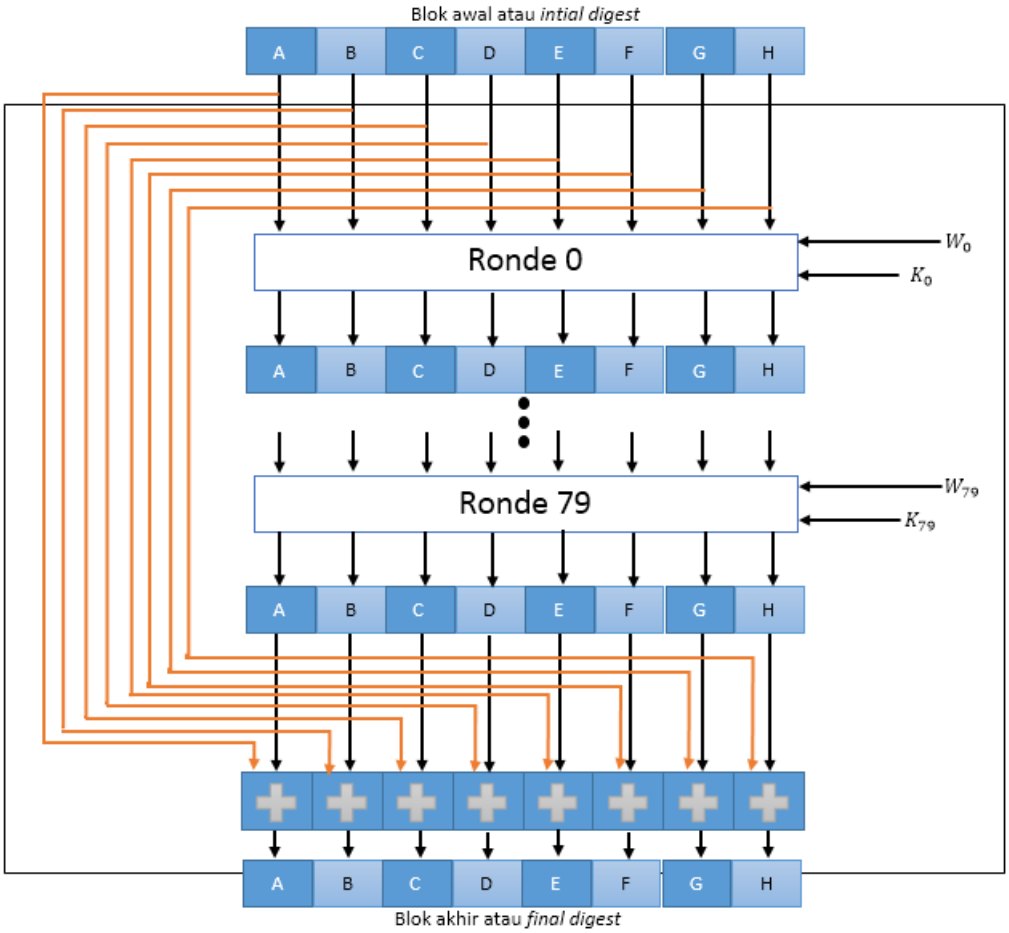
Sebagai contoh, H_0 didapat dari nilai $\sqrt{19} = 4.35889894354$. Kemudian, nilai ini akan dikonversi ke biner sepanjang 64-*bit* maka akan didapatkan $(100.0101 \quad 1011 \quad 11101001)_2 = (4.5BE0CD19137E2179)_{16}$. SHA-512 hanya mengambil nilai yang di belakang koma saja jadi nilai dari $H_0 = 5BE0CD19137E2179$. Gambar di bawah ini menunjukkan nilai dari A_0 sampai H_0 dalam heksadesimal.

Konstanta	Nilai	Konstanta	Nilai
A_0	6A09E667F3BCC908	E_0	510E527FADE682D1
B_0	BB67AE8584CAA73B	F_0	9B05688C2B3E6C1F
C_0	3C6EF372EF94F828	G_0	1F83D9ABFB41BD6B
D_0	A54FE53A5F1D36F1	H_0	5BE0CD19137E2179

Gambar 2.18: Konstanta inisialisasi dalam SHA-512

Fungsi Kompresi

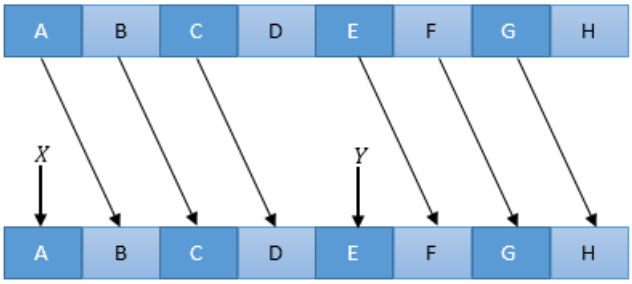
SHA-512 membuat *message digest* sepanjang 512-*bit* (8 *word*) dari beberapa blok *message* yang masing-masing panjangnya 1024-*bit*. Pemrosesan setiap blok *message* ini dilakukan sebanyak 80 ronde. Setiap ronde akan terdiri dari W_i digabungkan dengan K_i untuk membuat W_i baru yang akan digunakan pada ronde selanjutnya. Pada akhir ronde 79, nilai W_0 sampai W_7 pada ronde 79 akan ditambahkan oleh nilai W_0 sampai W_7 pada ronde yang paling awal (ronde 0) dan dimodulo oleh 2^{64}. Hasil yang didapatkan disebut *message digest* atau *final digest*. Gambar di bawah ini menunjukkan proses fungsi kompresi pada SHA-512.



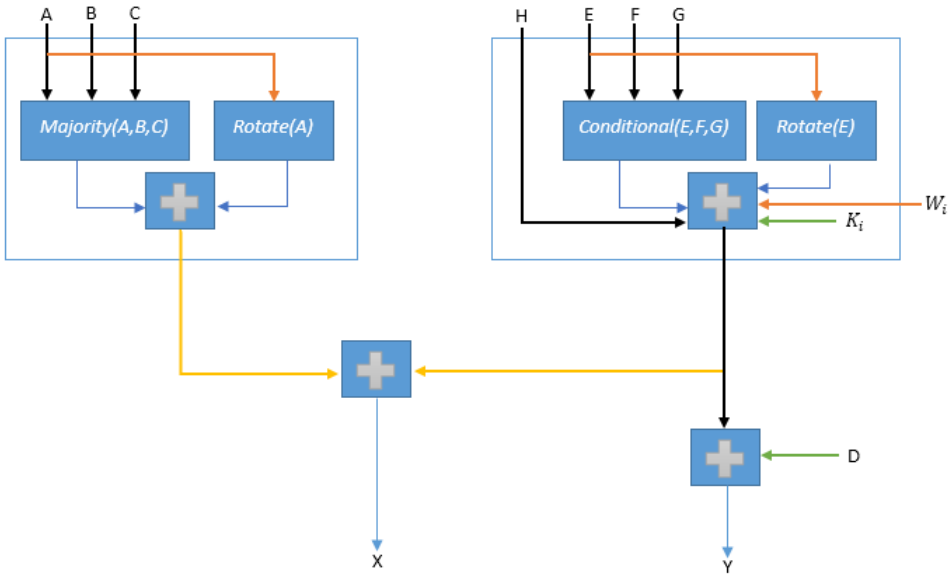
Gambar 2.19: Fungsi kompresi dalam SHA-512

Ronde

Pada setiap ronde, 8 nilai baru pada 64-*bit word* dibuat dari 8 nilai 64-*bit word* pada ronde sebelumnya. Nilai baru ini dinamakan *buffer*. Enam dari delapan *buffer* mendapatkan nilai dari *buffer* ronde sebelumnya. Sedangkan untuk 2 *buffer* lagi nilai keduanya didapatkan dari sebuah fungsi kompleks yang berhubungan dengan *buffer* sisanya serta gabungan tambahan dari W_i dan K_i, bersangkutan dengan masing-masing ronde. Gambar di bawah ini akan menjelaskan struktur dari setiap ronde serta fungsi kompleks yang digunakan untuk mendapatkan nilai untuk 2 *buffer* (A dan E).



Gambar 2.20: Struktur ronde dalam SHA-512



Gambar 2.21: Fungsi kompleks dalam SHA-512

Nilai dari $Majority(A, B, C)$, $Conditional(E, F, G)$, $Rotate(A)$, dan $Rotate(E)$ pada gambar di atas didapat dari rumus di bawah ini.

$$Majority(x, y, z) = (x\ AND\ y) \oplus (y\ AND\ z) \oplus (z\ AND\ x)$$

$$Conditional(x, y, z) = (x\ AND\ y) \oplus (NOT\ x\ AND\ z)$$

$$Rotate(x) = RotR_{28}(x) \oplus RotR_{34}(x) \oplus RotR_{39}(x)$$

$RotR_i(x)$ adalah rotasi ke kanan x sebanyak i *bit*. Simbol kotak yang berisi tanda tambah berarti adalah penambahan antar *bit-bit* lalu kemudian hasilnya di modulo oleh 2^{64}. K_i pada fungsi kompleks diatas didapat dengan cara yang sama saat mencari nilai dari konstanta awal A_0 sampai H_0. Namun, karena yang digunakan sebanyak 80 buah dari K_0 sampai K_{79} maka banyak bilangan prima yang digunakan juga sebanyak 80 bilangan prima pertama mulai dari 2 sampai 409.

Perbedaannya dengan nilai dari konstanta awal adalah nilai K_i pada fungsi kompleks didapat dari akar kubik bilangan prima yang bersangkutan. Selanjutnya, proses yang sama dengan cara menentukan nilai konstanta awal tetap digunakan untuk menentukan nilai K_i. Sebagai contoh, $\sqrt[3]{409} = 7.42291412044$, kemudian akan dikonversi ke dalam biner menjadi $(111.0110\ 1100\ 0100\ 0100\ 0111)_2$ dan dikonversi ke dalam heksadesimal menjadi $(7.6C44198C4A475817)_{16}$, kemudian nilai yang diambil hanya nilai di belakang koma, maka nilai dari $K_{79} = 6C44198C4A475817$.

2.3 Otentikasi

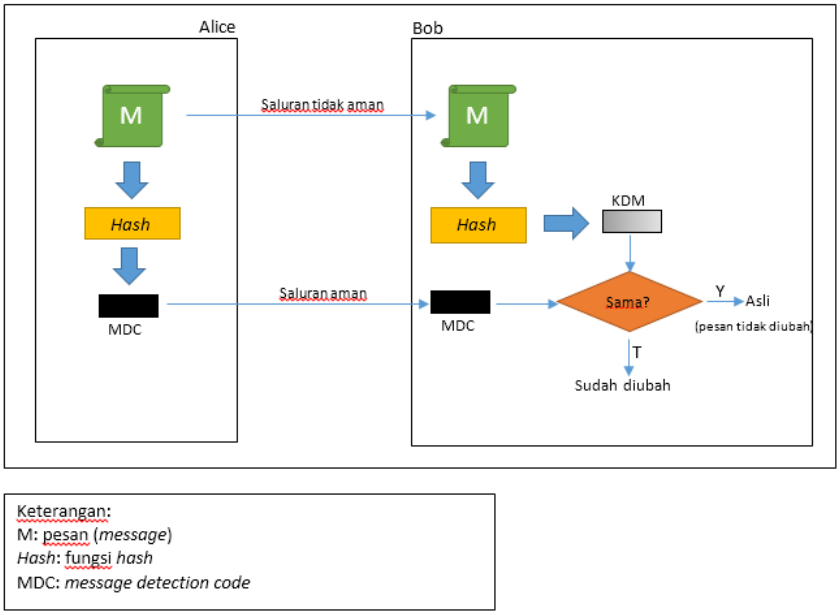
Otentikasi adalah proses untuk menentukan apakah sebuah entitas diijinkan untuk mengakses sumber daya yang dimiliki sebuah sistem. Otentikasi juga merupakan proses untuk memastikan keaslian data. Otentikasi dibagi menjadi 2 jenis, yaitu:

2.3.1 Otentikasi Pesan (*Message Authentication*)

Sebuah pesan inti (*message digest*) dapat digunakan untuk menjamin integritas sebuah pesan, memastikan pesan tidak diubah. Namun, pesan inti tidak bisa memastikan keaslian pengirim pesan. Ketika Alice mengirim pesan kepada Bob, Bob harus bisa tahu bahwa pesan yang dikirim berasal dari Alice. Supaya Bob bisa memastikan bahwa Alice yang mengirim pesan, maka Alice harus bisa menyediakan bukti bahwa Alice adalah pengirimnya. Contoh dari otentikasi ini antara lain adalah *modification detection code* (MDC) dan *message authentication code* (MAC).

Modification Detection Code

Modification detection code merupakan sebuah pesan inti yang dapat memastikan integritas dari sebuah pesan, bahwa pesan tidak diubah saat proses pengiriman. Alice dapat membuat pesan inti, yaitu kode deteksi modifikasi dan mengirimkannya bersama dengan pesan asli kemudian Bob akan membuat juga secara terpisah kode deteksi modifikasi berdasarkan pesan asli yang dikirim Alice dan membandingkan dengan kode deteksi modifikasi yang dikirim Alice. Jika sama, maka pesan yang dikirim oleh Alice asli dan tidak berubah saat pengiriman. Proses diatas dapat ditunjukkan pada diagram di bawah ini.

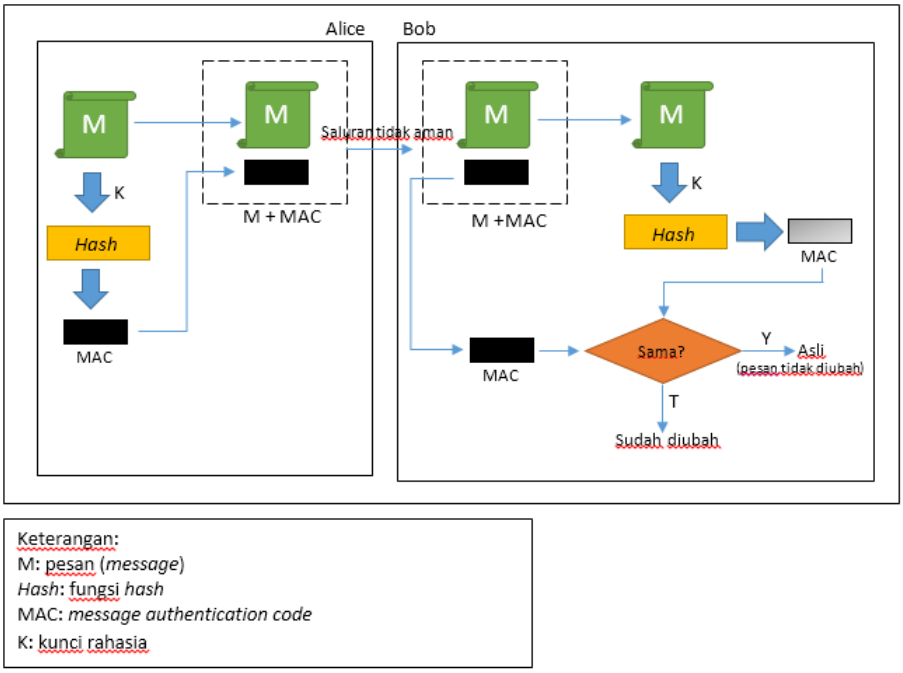


Gambar 2.22: *Modification detection code*

Diagram diatas menunjukkan bahwa pesan yang dikirim Alice dapat dikirim melalui saluran yang tidak aman, maksudnya pihak lain diluar Alice dan Bob bisa membacanya bahkan mengubah isinya, tapi MDC yang dibuat oleh Alice harus dikirimkan pada Bob di saluran aman. Dengan cara ini, MDC tidak akan bisa diubah isinya dan ketika pesan yang dikirim melalui saluran tidak aman diubah maka Bob akan tahu bahwa pesan sudah diubah oleh pihak lain dengan membandingkan hasil MDC dari pesan yang dia terima dengan MDC yang Alice kirimkan melalui saluran aman.

Message Authentication Code

Untuk memastikan bahwa memang Alice yang mengirimkan pesan kepada Bob maka kode deteksi pesan yang digunakan harus diubah menjadi *message authentication code* (MAC). Perbedaan MAC dengan MDC adalah MAC menggunakan sebuah kunci rahasia atau kunci pribadi yang hanya diketahui oleh Alice dan Bob saja. Diagram di bawah ini akan menunjukkan cara kerja MAC dalam mengotentikasi pesan.



Gambar 2.23: *Modification authentication code*

Alice menggunakan fungsi *hash* untuk membuat MAC dengan menggunakan gabungan dari pesan dan kunci rahasia yang hanya diketahui olehnya dan Bob saja. Kemudian, Alice mengirimkan pesan beserta dengan MAC pesan tersebut kepada Bob. Bob akan menerima pesan dan memisahkan MAC dengan pesan dan kemudian Bob akan membuat MAC dari pesan yang diterima dari Alice dengan menggunakan fungsi hash yang sama seperti Alice gunakan dan tentu saja MAC yang dibuat merupakan gabungan dari pesan dan kunci rahasia. Melalui cara ini, jika hasil keluaran MAC yang Bob buat sendiri dibandingkan dengan MAC yang Alice kirimkan sama, maka Bob bisa yakin dengan pasti bahwa pesan yang dia terima berasal dari Alice dan pesan tersebut asli, tidak diubah oleh pihak lain. Terdapat beberapa modifikasi dari *message authentication code* antara lain adalah *nested MAC*, *hashed MAC*, dan CMAC.

2.3.2 Otentikasi Entitas (*Entity Authentication*)

Otentikasi entitas merupakan sebuah teknik yang dirancang untuk memastikan kebenaran identitas seseorang. Entitas yang dimaksud disini bisa berupa orang, pengguna (*user*), atau sebuah *server*. Entitas yang identitasnya perlu dibuktikan kebenarannya disebut penuntut (*claimant*) dan entitas yang bertindak untuk memastikan kebenaran identitas dari penuntut disebut pemeriksa (*verifier*). Ketika Bob hendak memastikan kebenaran identitas dari Alice, Bob adalah pemeriksa dan Alice

adalah penuntut.

Ada dua perbedaan antara otentikasi pesan dan otentikasi entitas:

1. Otentikasi pesan atau otentikasi sumber data tidak terjadi secara langsung sedangkan otentikasi entitas terjadi secara langsung.
2. Otentikasi pesan hanya mengotentikasi satu pesan saja, ketika pesan berikutnya dikirimkan maka proses otentikasi pesan akan dilakukan kembali sedangkan otentikasi entitas akan terjadi selama satu durasi dalam sebuah sesi.

Dalam otentikasi entitas, penuntut harus bisa membuktikan kebenaran identitas dirinya kepada pemeriksa. Ada beberapa cara pembuktian yang bisa dilakukan oleh penuntut, yaitu:

- Sesuatu yang diketahui (*something known*). Hal ini diketahui oleh penuntut dan juga pemeriksa. Contohnya antara lain adalah *password*, nomor PIN, kunci rahasia, dan kunci pribadi.
- Sesuatu yang dimiliki (*something possessed*). Hal ini dimiliki oleh penuntut dan bisa memastikan kebenaran identitas dari penuntut. Contohnya antara lain adalah paspor, KTP, kartu kredit, dan SIM.
- Sesuatu yang melekat (*something inherent*). Hal ini menempel atau sebagai bagian dari penuntut. Contohnya antara lain adalah sidik jari, suara, tanda tangan, pola retina, dan tulisan tangan.

Beberapa teknik dalam otentikasi entitas antara lain, yaitu *password*, *zero-knowledge*, *challenge-response*, dan biometrik.

2.3.3 *Password*

Salah satu teknik otentikasi entitas yang paling sederhana dan mudah adalah otentikasi menggunakan *password* atau kata kunci rahasia. *Password* ini adalah sesuatu yang diketahui hanya oleh penuntut. *Password* digunakan saat penuntut (selanjutnya akan dinamakan pengguna) perlu untuk mengakses sistem untuk menggunakan sumber daya dari sistem. Setiap pengguna akan diberikan *username* yang sifatnya publik dan *password* yang sifatnya rahasia atau pribadi. Ada dua skema otentikasi menggunakan *password* ini, yaitu *password* tetap (*fixed password*) dan *password* satu-kali (*one-time password*).

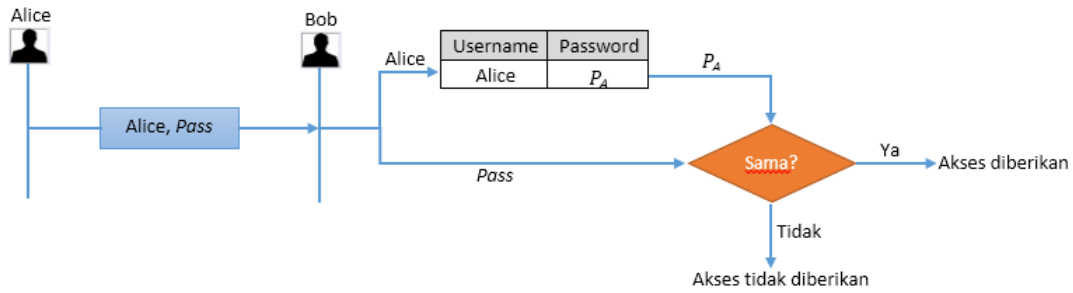
Password Satu-Kali

Password satu-kali adalah *password* yang digunakan hanya satu kali. Jadi, setiap kali pengguna akan mengakses sistem untuk setiap sesi waktu yang berbeda, maka *password* yang digunakan akan selalu berbeda-beda. Beberapa teknik yang digunakan dalam *password* satu-kali antara lain menggunakan *list*, *sequentially updated password*, dan *password* satu-kali *lamport*.

Password Tetap

Password tetap (*fixed password*) adalah *password* yang digunakan berulang-ulang kali setiap saat pengguna akan mengakses sistem. *Password* ini akan selalu sama untuk setiap kali pengguna akan mengakses sistem. Ada beberapa skema dari *password* tetap ini.

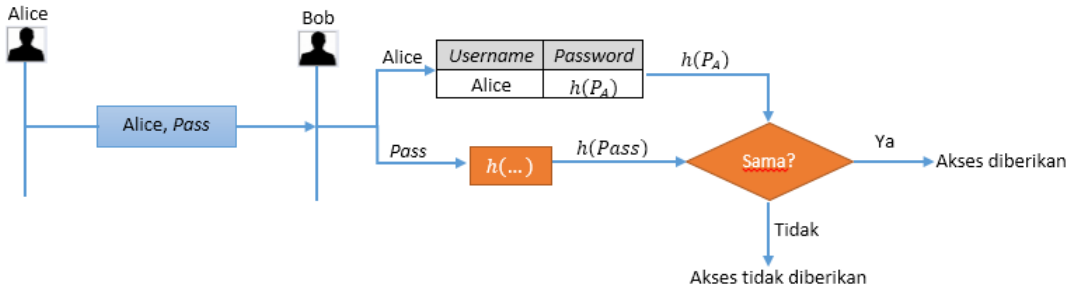
- Skema 1
  Dalam skema ini, sistem menyimpan setiap *password* dalam sebuah tabel basis data yang diurutkan berdasarkan nama pengguna (*username*). Saat pengguna akan mengakses sistem, maka pengguna akan memasukkan *username* dan *password* dalam bentuk *plaintext*. Kemudian, sistem akan mencari *password* berdasarkan *username*. Jika *password* yang dimasukkan pengguna sesuai dengan *password* yang ada dalam tabel basis data maka pengguna diberikan akses masuk ke dalam sistem, jika tidak sesuai maka pengguna tidak diberikan akses masuk ke dalam sistem. Kekurangan dari skema ini adalah karena *password* disimpan dalam tabel basis data maka setiap orang yang memiliki akses ke dalam basis data dapat mengetahui setiap *password* yang disimpan dan tentu saja hal ini melanggar salah satu layanan dari kriptografi yaitu, kerahasiaan data (*data confidentiality*).



Gambar 2.24: *Username* dan *Password*

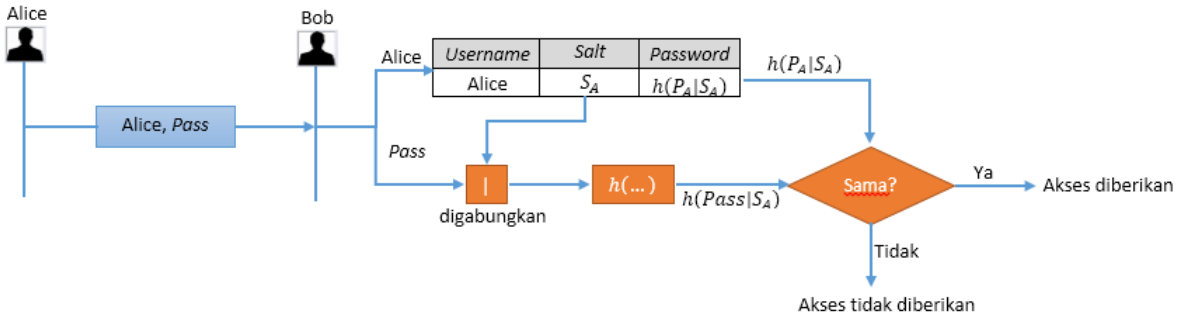
- Skema 2
  Skema yang lebih aman dibandingkan skema diatas adalah skema penyimpanan password yang menggunakan fungsi *hash*. Sistem tidak lagi menyimpan *password* dengan bentuk *plaintext* dalam tabel basis data, namun sistem menyimpan *password* dalam bentuk hashnya dalam tabel basis data, sehingga saat sistem dibobol maka penyerang tidak dapat mengetahui *password* secara langsung karena fungsi *hash* adalah fungsi satu arah dan mustahil untuk mengembalikan ke *plaintext*nya. Selain itu, orang-orang yang memiliki akses ke dalam basis data tidak bisa mengetahui *password* dari setiap *username* yang disimpan dalam tabel. Ketika *password* dibuat, sistem akan menghitung nilai *hash* dari *password* dan menyimpan nilai hash tersebut dalam tabel basis data. Saat pengguna akan mengakses sistem, pengguna memasukkan *username* dan *password* dalam bentuk *plaintext* kemudian sistem akan menghitung nilai hash dari *password* tersebut dan menyesuaikan dengan nilai *hash* dari *password* yang bersangkutan dalam tabel basis data. Jika sesuai, maka pengguna diberikan ijin masuk ke dalam sistem, jika tidak maka pengguna tidak diberikan ijin masuk.

- Skema 3
  Skema ketiga menggunakan teknik *salting*. Ketika *password* dibuat, sebuah *string* acak, disebut *salt*, akan ditambahkan pada *password*. Setelah ditambahkan, *password* yang bersangkutan akan dihitung nilai *hash*nya. Sistem akan menyimpan *username*, *salt*, dan *hash* atau *digest* dari *password* dalam tabel basis data. Ketika pengguna akan mengakses sistem, sistem akan mengambil *salt* berdasarkan *username* yang dimasukkan kemudian menambahkan *salt* tersebut pada *password* yang dimasukkan dan menghitung nilai *hash*nya kemudian menyesuaikan



Gambar 2.25: *Password hashing*

dengan yang ada pada tabel basis data. Jika sesuai, maka akses diberikan, jika tidak maka akses akan ditolak.



Gambar 2.26: *Password salting*

2.4 *Secret Sharing*

Enkripsi digunakan untuk menjaga keamanan informasi dan setiap proses enkripsi disertai oleh sebuah kunci rahasia atau *password*. *Password* ini tentu saja harus selalu diingat oleh manusia atau disimpan dalam memori komputer. Namun, hal ini tidak sepenuhnya aman karena bisa saja manusia lupa atau terjadi kerusakan komputer atau bencana alam sehingga *password* ini hilang dan informasi yang dienkripsi tidak akan bisa diakses atau dikembalikan.

Secret sharing adalah metode untuk membagi informasi (rahasia) menjadi beberapa bagian. Bagian-bagian tersebut disebut *share* dan setiap bagian dibagikan kepada beberapa partisipan. Untuk mendapatkan kembali informasi, maka dibutuhkan sejumlah *share*.

Shamir mendasarkan metode *secret sharing* dalam sebuah masalah sebagai berikut:

"Sebelas orang ilmuwan mengerjakan sebuah projek rahasia. Mereka menyimpan dokumen dalam sebuah kabinet dimana kabinet ini hanya akan bisa dibuka jika enam atau lebih ilmuwan hadir. Berapa banyak kunci minimal yang dibutuhkan untuk membuka kabinet? Berapa banyak kunci yang harus dimiliki oleh masing-masing ilmuwan?"

Tentu saja jawabannya minimal sebanyak 462 kunci untuk membuka kabinet dan minimal 252 kunci yang harus dimiliki oleh setiap ilmuwan. Angka ini sangat tidak masuk akal dan menyulitkan

apalagi jika banyak ilmuwannya bertambah. Dalam skema *secret sharing* shamir, suatu data D akan dibagi menjadi n buah *share* dengan ketentuan sebagai berikut:

- Jika bagian yang ada sebanyak k bagian atau lebih akan membuat D mudah untuk dibentuk kembali.
- Jika bagian yang ada hanya sebanyak *k-1* atau kurang maka D tidak akan dapat dibentuk kembali.

Skema diatas dinamakan skema *threshold*(k,n).

2.4.1 Skema *Threshold*(k,n)

Enkripsi dilakukan untuk melindungi kerahasiaan sebuah data atau informasi, namun karena setiap proses enkripsi memerlukan kunci maka diperlukan cara lain untuk melindungi kerahasiaan kunci tersebut. Salah satu cara yang paling aman dalam melindungi kunci kriptografi adalah dengan menyimpannya dalam sebuah lokasi yang aman dan tersembunyi (komputer, otak manusia, atau lemari besi).

Namun cara ini tidak sepenuhnya aman karena bisa saja terjadi bencana alam, kematian, penipuan dan sabotase menyebabkan kunci yang disimpan menjadi rusak atau hilang sehingga data yang dienkripsi menjadi tidak bisa diakses dan hilang sepenuhnya. Solusi lainnya adalah dengan memperbanyak kunci yang ada dan menyimpannya di tempat yang berbeda-beda, satu di memori komputer, satu diingat oleh manusia, dan yang lainnya disimpan di tempat yang berbeda-beda. Namun, cara ini juga tidak aman karena masih tidak terhindar dari bencana alam, *human error*, dan malapetaka lainnya.

Untuk mengatasi permasalahan ini maka dibutuhkan sebuah mekanisme atau skema, skema tersebut dinamakan skema *threshold*(k,n). Skema *threshold(k,n)* didasarkan pada interpolasi polinomial: diberikan k titik pada bidang kartersius $(x_1, y_1), (x_2, y_2), ..., (x_k, y_k)$ untuk setiap x_i maka hanya akan ada 1 saja nilai $q(x_i)$ dalam derajat $k-1$ dimana $q(x_i) = y_i$ untuk seluruh x_i. Misalkan diasumsikan bahwa D adalah representasi berupa angka dari kunci yang akan dipecah menjadi beberapa bagian atau *share* menggunakan skema *threshold(k,n)*, kemudian untuk membagi D menjadi D_i *share*, dipilih $k-1$ derajat polinomial untuk membentuk fungsi $q(x)$:

$$q(x) = a_0 + a_1 * x + a_2 * x^2 + ... + a_{k-1} * x^{k-1}$$

$$a_0 = D$$

Nilai a_1 sampai a_{k-1} dipilih secara acak kemudian dihitung nilai dari D_1 sampai D_n dimana n adalah banyaknya *share*:

$$D_1 = q(1), D_2 = q(2), ..., D_i = q(i), ..., D_n = q(n)$$

Untuk setiap nilai dari D_n yang diketahui, maka bisa dicari nilai koefesien dari fungsi $q(x)$ dengan interpolasi atau penyelesaian sistem persamaan linear dari situ bisa didapatkan nilai dari $q(0)$ sehingga nilai D nanti bisa diketahui. Namun, D bisa dihitung jika k atau lebih *share* diketahui, jika hanya $k-1$ atau kurang *share* yang diketahui, maka nilai D tidak bisa dihitung sehingga data tidak bisa dikembalikan.

Namun, dengan nilai k yang tinggi juga akan sulit untuk mendapatkan data, karena banyak *share* yang dikumpulkan. Dengan mengurangi nilai k, akan mempermudah untuk mendapatkan kembali data sehingga mengatasi akibat atau hilangnya beberapa *share*. Pemilihan k dan n yang tepat dapat menjaga kerahasiaan data.

2.5 Probabilitas

Probabilitas atau peluang merupakan salah cara dalam ilmu matematika untuk mengukur tingkat kepercayaan akan suatu kejadian. Teori probabilitas sangat luas penggunaannya, baik dalam kehidupan sehari-hari maupun dalam percobaan-percobaan ilmiah. Teori probabilitas ini seringkali digunakan oleh para pengambil keputusan untuk memprediksi suatu kejadian sehingga nantinya bisa mengambil keputusan yang tepat.

Seluruh kemungkinan keluaran yang akan terjadi dalam probabilitas disebut ruang sampel sedangkan masing-masing kemungkinan yang dapat terjadi dalam ruang sampel dinamakan elemen atau anggota dari ruang sampel. Ruang sampel dilambangkan dengan huruf S dan elemen dilambangkan dengan huruf x_i. Dalam notasi matematikanya:

$$S = x_1, x_2, x_3, ..., x_n$$

Sedangkan probabilitas akan kemungkinan bahwa kejadian x_i akan terjadi dilambangkan dengan $P(x_i)$. Dalam rumus matematikanya:

$$P(x_i) = \frac{n}{N}$$

dimana n adalah banyaknya kemunculan kejadian x_i dalam sebuah ruang sampel S dan N adalah banyaknya kejadian yang terjadi dalam ruang sampel S.

2.6 Distribusi Binom

Setiap eksperimen atau percobaan yang dilakukan secara berkali-kali pasti memiliki dua keluaran, yaitu sukses atau gagal. Untuk setiap keluaran yang didapatkan (baik sukses maupun gagal) bisa ditetapkan sebagai sukses. Proses ini dinamakan proses bernouli dan setiap eksperimen yang dilakukan untuk setiap proses bernouli dinamakan percobaan bernouli. Ada beberapa syarat sebuah eksperimen bisa dinamakan percobaan bernouli:

1. Eksperimen harus diulang sebanyak n kali.
2. Setiap keluaran dari perulangan bisa dianggap sukses atau gagal.
3. Probabilitas bahwa keluarannya sukses, p, harus selalu sama untuk setiap kali perulangan.
4. Setiap perulangan tidak dipengaruhi dengan perulangan sebelumnya atau independen.

Maka, untuk banyak perulangan yang sukses, X, dalam n perulangan percobaan bernouli dinamakan variabel acak binom sedangkan untuk probabilitas dari variabel acak ini dinamakan distribusi binom dilambangkan dengan $P(x, n, p)$, dimana p merupakan probabilitas keluaran sukses dan

$1-p$, yaitu q merupakan probabilitas keluaran gagal. Karena percobaan dilakukan sebanyak n kali maka probabilitas keseluruhannya menjadi $p^n q^{(n-x)}$, kemudian karena sebanyak x kejadian sukses yang diambil dari keseluruhan ruang sampel n, maka probabilitas p dan q akan dikalikan dengan kombinasi n dan x, dituliskan menjadi $\begin{pmatrix} n \\ x \end{pmatrix}$. Jadi, dalam notasi matematikanya:

$$P(x,n,p) = \begin{pmatrix} n \\ x \end{pmatrix} p^n q^{(n-x)} \qquad x = 0,1,2,...,n$$

2.7 Entropi

Entropi adalah rata-rata banyak informasi yang dimiliki oleh sebuah pesan. Pesan yang dimaksud disini adalah kejadian yang diharapkan atau elemen dari sebuah ruang sampel atau kejadian. Maka dari itu, entropi bisa dijadikan alat ukur dari tingkat ketidakpastian yang dimiliki oleh sebuah pesan atau sumber informasi. Misalkan sebuah pesan X terdiri dari huruf-huruf. Untuk setiap huruf x yang ada dalam pesan X memiliki banyak kemunculan n dan X terdiri dari huruf-huruf sebanyak N. Kemudian untuk setiap huruf yang bukan x, y, memiliki banyak kemunculan m. Maka, nilai entropi yang dimiliki oleh pesan X untuk huruf x, $H(x)$:

$$H(x) = -\frac{n}{N}\log_2(\frac{n}{N}) - \frac{m}{N}\log_2(\frac{m}{N})$$

atau untuk setiap huruf x dalam pesan X, probabilitas kemunculannya adalah p dan untuk setiap huruf bukan x dalam pesan X probabilitas kemunculannya adalah q, sehingga rumus nilai entropinya menjadi:

$$H(x) = -p\log_2 p - q\log_2 q$$

dimana $q = 1-p$ dan menggunakan logaritma basis 2 karena probabilitas kemunculannya biner, yaitu huruf x dan huruf bukan x.

BAB 3

ANALISIS

Pada bab ini akan dibahas analisis terhadap teori-teori yang telah dibahas sebelumnya. Analisis akan meliputi studi kasus untuk algoritma *secret sharing* shamir yang telah dibahas sebelumnya, algoritma *secret sharing* shamir yang dikembangkan, dan perancangan perangkat lunak.

3.1 Studi Kasus

Bagian ini akan berisi studi kasus mengenai algoritma *secret sharing* shamir yang telah dibahas sebelumnya.

3.1.1 *Secret Sharing* Shamir

Pada bagian ini akan dijelaskan proses pembangunan *share* untuk *secret sharing* shamir pada sebuah angka rahasia S dan proses pembangunan kembali S dari *share-share* yang ada. Untuk kasus ini, dipilih $S = 1234$.

Proses pembangunan *share*

Langkah yang perlu dilakukan pertama adalah memilih banyak *share* n yang diinginkan dan banyak minimal *share* yang diperlukan untuk mengembalikan S, yaitu k. Untuk kasus ini, akan dipilih $n = 8$ dan $k = 3$.

Langkah selanjutnya adalah memilih $k - 1$ angka acak yang nanti akan digunakan sebagai koefesien fungsi polinomial $f(x)$. Karena pada kasus ini $k = 3$ maka ada 2 angka acak yang dipilih, misalkan 237 dan 55. Maka fungsi $f(x)$ untuk menghitung nilai setiap *share*:

$$f(x) = 1234 + 237x + 55x^2$$

Kemudian, akan dihitung nilai dari setiap $f(x_i)$ dari $f(1)$ sampai $f(8)$ karena $n = 8$:

$$f(1) = 1234 + 237 * 1 + 55 * 1 * 1 = 1526$$

$$f(2) = 1234 + 237 * 2 + 55 * 2 * 2 = 1928$$

$$f(3) = 1234 + 237 * 3 + 55 * 3 * 3 = 2440$$

$$f(4) = 1234 + 237 * 4 + 55 * 4 * 4 = 3062$$

$$f(5) = 1234 + 237 * 5 + 55 * 5 * 5 = 3794$$

$$f(6) = 1234 + 237 * 6 + 55 * 6 * 6 = 4636$$

$$f(7) = 1234 + 237 * 7 + 55 * 7 * 7 = 5588$$

$$f(8) = 1234 + 237 * 8 + 55 * 8 * 8 = 6650$$

Maka didapatkan nilai untuk setiap S_1 sampai S_8.

$$S_1 = 1526, S_2 = 1928, S_3 = 2440, S_4 = 3062, S_5 = 3794, S_6 = 4636, S_7 = 5588, S_8 = 6650$$

Proses pembangunan kembali (rekonstruksi S)

Karena pada kasus ini $k = 3$ maka untuk mengembalikan S maka hanya dibutuhkan 3 *share* aja. Misalkan *share* yang dipilih adalah S_2, S_4, dan S_5.

Langkah selanjutnya adalah membentuk rumus dasar dari fungsi $f(x)$. Karena pada kasus ini, $k = 3$, maka rumus dasar dari $f(x)$:

$$f(x) = c + bx + ax^2$$

Setelah rumus dasar dari fungsi $f(x)$ dibentuk langkah selanjutnya adalah menghitung nilai masing-masing fungsi berdasarkan *share* yang diketahui, dalam kasus ini adalah S_2, S_4, dan S_5, maka nilai masing-masing fungsi $f(x)$:

$$f(2) = c + 2b + 4a = 1928$$

$$f(4) = c + 2b + 16a = 3062$$

$$f(5) = c + 5b + 25a = 3794$$

Langkah selanjutnya adalah menyelesaikan persamaan linear diatas, sehingga nanti bisa didapatkan nilai c dimana seperti yang diketahui nilai c merupakan S karena S adalah konstanta tanpa koefesien dari $f(x)$ ($f(0) = c$).

Proses penyelesaian persamaan linear untuk pembangunan kembali (rekonstruksi S)

Pada bagian ini akan dijelaskan proses penyelesaian persamaan linear menggunakan eliminasi gauss. Pada kasus ini, persamaan linear yang diperoleh:

$$c + 2b + 4a = 1928$$

$$c + 2b + 16a = 3062$$

$$c + 5b + 25a = 3794$$

Langkah awal yang perlu dilakukan dalam eliminasi gauss adalah transformasi persamaan linear ke matriks. Maka, dari persamaan linear diatas matriksnya:

$$\begin{bmatrix} 1 & 2 & 4 & 1928 \\ 1 & 4 & 16 & 3062 \\ 1 & 5 & 25 & 3794 \end{bmatrix}$$

3.2 Diagram Kelas

DAFTAR REFERENSI

LAMPIRAN A

THE PROGRAM