BAB II
LANDASAN TEORI

2.1 Kriptografi

Menurut Stalling, ada beberapa tuntutan yang terkait dengan isu keamanan data yaitu :

1. *Confidentiality*

   Menjamin bahwa data-data tersebut hanya bisa diakses oleh pihak-pihak tertentu saja.

2. *Authentication*

   Baik pada saat mengirim atau menerima informasi, kedua belah pihak perlu mengetahui bahwa pengirim dari pesan tersebut adalah orang yang sebenarnya seperti yang diklaim.

3. *Integrity*

   Tuntutan ini berhubungan dengan jaminan setiap pesan yang dikirim pasti sampai pada penerimanya tanpa ada bagian dari pesan tersebut yang diganti, diduplikasi, dirusak, diubah urutannya, dan ditambahkan.

4. *Nonrepudiation*

   Nonrepudiation mencegah pengirim maupun penerima mengingkari bahwa mereka telah mengirimkan atau menerima suatu pesan/informasi. Jika sebuah pesan dikirim, penerima dapat membuktikan bahwa pesan tersebut memang dikirim oleh pengirim yang tertera. Sebaliknya, jika sebuah pesan diterima, pengirim dapat membuktikan bahwa pesannya telah diterima oleh pihak yang ditujunya.

5. *Access Control*

   Membatasi sumber-sumber data hanya kepada orang-orang tertentu.

6. *Availability*

   Jika diperlukan setiap saat semua informasi pada sistem komputer harus tersedia bagi semua pihak yang berhak atas informasi tersebut.

Dari keenam aspek keamanan data tersebut, empat diantaranya dapat diatasi dengan menggunakan kriptografi yaitu *confidentiality*, *integrity*, *authentication*, dan *nonrepudiation.*

Kriptografi dapat didefinisikan sebagai ilmu yang mempelajari teknik-teknik matematika yang berhubungan dengan aspek-aspek pada keamanan informasi misalnya kerahasiaan, integritas data, otentikasi pengirim / penerima data, dan otentikasi data. *Cryptanalysis* adalah studi tentang bagaimana mengalahkan (memecahkan) mekanisme kriptografi, dan *cryptology* yang berasal dari kata kryptos dan logos (bahasa Yunani) yang artinya kata tersembunyi, adalah penggabungan disiplin *cryptography* dan *cryptanalysis* [1].

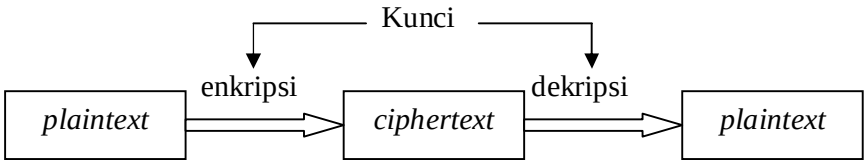
2.2 Sistem Kriptografi

Berdasarkan jumlah kunci yang digunakan, ada dua jenis sistem kriptografi yaitu sistem kriptografi simetris dan sistem kriptografi asimetris.

2.2.1 Sistem Kriptografi Simetris

Enkripsi simetris sering juga disebut sebagai enkripsi konvensional atau enkripsi kunci-tunggal (*single key*). Pada model enkripsi simetris ini digunakan

algoritma yang sama untuk proses enkripsi/dekripsi dengan memakai satu kunci yang sama.



Gambar 2.1. Model sederhana Sistem Kriptografi Simetris

Keamanan dari enkripsi simetris bergantung pada beberapa faktor, yaitu :

1. Algoritma enkripsi harus cukup kuat sehingga tidaklah praktis untuk mendekripsi suatu pesan hanya dengan memiliki *cyphertext* saja.
2. Keamanan dari enkripsi simetris adalah bergantung pada kerahasiaan kunci, bukan kerahasiaan dari algoritma enkripsi itu sendiri. Semakin panjang kunci yang dipakai maka semakin sulit untuk menebak kunci dengan menggunakan metode *brute force attacks* (mencoba semua kemungkinan kunci).

Algoritma enkripsi simetris yang popular dewasa ini adalah DES (*Data Encryption Standard*) dengan panjang kunci 56-bit, IDEA (128-bit), Twofish (sampai dengan 256-bit), Rijndael (sampai dengan 256-bit) dan lain-lain [3].

2.2.2 Sistem Kriptografi Asimetris

Sistem kriptografi asimetris biasanya lebih dikenal dengan kriptografi kunci-publik (*public-key cryptography)*. Ide kriptografi asimetris ini pertama kali dimunculkan oleh Whitfield Diffie dan Martin Hellman pada tahun 1976. Diffie dan Hellman mempostulatkan sistem ini tanpa menunjukkan algoritmanya.

Walaupun demikian mereka menjabarkan syarat-syarat yang harus dipenuhi oleh suatu algoritma *public-key* yaitu :

1. Mudah secara komputasi bagi suatu pihak B untuk mengkonstruksi sepasang kunci asimetris (kunci public KU, kunci pribadi KR).
2. Mudah secara komputasi bagi pengirim A, dengan memiliki kunci public B dan pesan yang ingin dienkripsi, M, untuk menghasilkan *ciphertext* (C) :

$$C = E_{KUb}(M)$$

3. Mudah secara komputasi bagi penerima B untuk mendekripsi *ciphertext* yang dihasilkan dengan menggunakan kunci pribadinya untuk mengembalikan pesan aslinya.

$$M = D_{KRb}(C) = D_{KRb}[E_{KUb}(M)]$$

4. Tidak bisa secara komputasi bagi pihak ketiga untuk memperoleh kunci pribadi KRb hanya dengan mengetahui kunci public KUb.
5. Tidak bisa secara komputasi bagi pihak ketiga untuk mengembalikan data asli M hanya dengan mengetahui kunci public KUb dan *ciphertext* C.

Walaupun bukanlah suatu keharusan bagi semua aplikasi *public-key*, namun persyaratan keenam bisa ditambahkan :

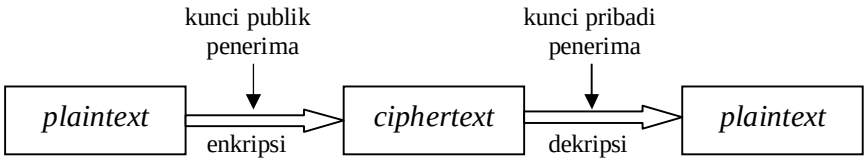
6. Fungsi enkripsi dan dekripsi bisa diterapkan dengan urutan yang dibalik :

$$M = E_{KUb}[D_{KRb}(M)]$$

Kegunaan dari persyaratan keenam adalah untuk penerapan tanda tangan digital (*digital signature*) yang digunakan memecahkan isu otentikasi (*authentication*) dalam masalah keamanan data.

Menurut Stalling, proses enkripsi *public-key* sederhana melibatkan empat tahap berikut :

1. Setiap *user* di dalam jaringan membuat sepasang kunci untuk digunakan sebagai kunci enkripsi dan dekripsi dari pesan yang akan diterima.
2. *User* mempublikasikan kunci enkripsinya dengan menempatkan kunci publiknya ke tempat umum. Pasangan kunci yang lain tetap dijaga kerahasiaannya.
3. Jika *user* A ingin mengirimkan sebuah pesan ke *user* B, ia akan mengenkripsi pesan tersebut dengan menggunakan kunci publik *user* B.
4. Pada saat *user* B ingin mengirimkan sebuah pesan ke *user* B, ia akan menggunakan kunci pribadinya sendiri. Tidak ada pihak lain yang bisa mendekripsi pesan itu karena hanya B sendiri yang mengetahui kunci pribadi B.



Gambar 2.2 Model Sederhana Sistem Kriptografi Asimetris

Sampai saat ini hanya ada beberapa sistem *cryptography* asimetris yang dipublikasikan. Yang paling berhasil sejauh ini adalah algoritma RSA yang memenuhi keenam persyaratan *public-key* di atas. Di samping itu, algoritma enkripsi *public-key* yang lain adalah LUC, DSS, Diffie-Hellman dan lain-lain.

Kunci publik adalah kunci yang tidak disembunyikan dan boleh diketahui oleh orang lain. Kunci publik digunakan dalam proses enkripsi.

Kunci *private* adalah kunci rahasia yang tidak boleh diketahui oleh orang lain. Kunci *private* digunakan dalam proses dekripsi [3].

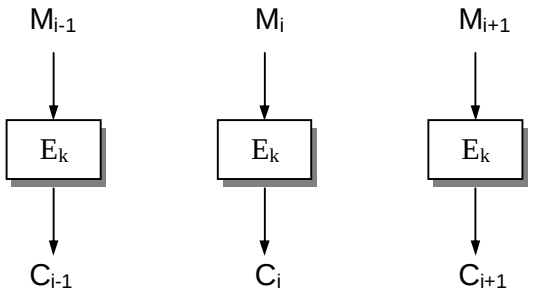
2.2.3 Block Cipher dan Mode Operasi

Sebuah *block cipher* adalah sebuah fungsi yang memetakan n-bit blok *plaintext* menjadi n-bit *ciphertext* (Menezes, 1996). Fungsi tersebut terdiri dari sebuah algoritma dan sebuah kunci. Hasil pemetaan dari *plaintext* ke *ciphertext* akan berbeda-beda tergantung pada kunci yang digunakan. Baik *cryptography* simetris maupun *cryptography* asimetris bisa merupakan *block cipher*.

Untuk *plaintext* yang panjangnya lebih besar dari n-bit perlu dipilih mode operasi untuk menentukan cara enkripsi/dekripsi *plaintext* tersebut. Ada beberapa pilihan mode operasi yang bisa diterapkan antara lain *Electronic CodeBook* (ECB), *Cipher Block Chaining* (CBC), *Cipher FeedBack* (CFB), *Output FeedBack* (OFB). Keempat mode operasi ini memiliki kelebihan dan kekurangan masing-masing. Untuk penelitian dalam skripsi ini mode operasi yang digunakan hanya ECB dan CBC saja [3].

2.2.3.1 Mode Operasi ECB (Electronic Code Book)

Mode operasi ECB membagi-bagi *plaintext* menjadi blok-blok yang panjangnya n-bit dan masing-masing blok dienkripsi secara terpisah dengan menggunakan kunci yang sama. Dengan demikian, untuk blok n-bit *plaintext* yang sama dalam satu pesan akan menghasilkan n-bit *ciphertext* yang sama pula.



$C_i = E_k (M_i)$; $M_i = E_k (C_i)$

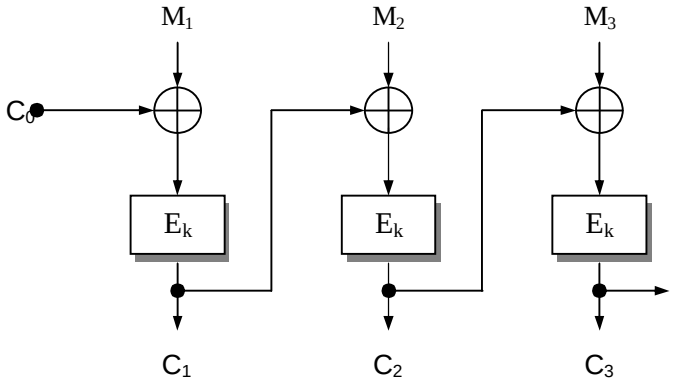
Gambar 2.3 Mode Electronic Code Book (ECB)

Dari gambar di atas dapat dilihat *plaintext* dipecah menjadi urutan yang terdiri dari blok-blok n-bit, M_{i-1}, M_i, M_{i+1} dengan hasil enkripsinya berupa blok-blok *ciphertext* n-bit, C_{i-1}, C_i, C_{i+1} [3].

2.2.3.2 Mode Operasi CBC (Cipher Block Chaining)

Salah satu alternatif untuk mencegah munculnya blok-blok *ciphertext* yang sama dari blok *plaintext* yang sama pada satu pesan adalah dengan menggunakan mode CBC. Pada skema ini setiap blok n-bit plaintext di-XOR-kan dengan blok n-bit *ciphertext* sebelumnya. Kecuali blok *plaintext* pertama di-XOR-kan dengan suatu konstanta awal atau initialization vector (IV), sebesar n-bit. Hasil dari proses XOR tersebut yang kemudian dienkripsi.

Untuk proses dekripsi, hasil dekripsi blok *ciphertext* di-XOR-kan dengan blok *ciphertext* sebelumnya untuk menghasilkan blok *plaintext*. Untuk blok pertama, hasil dekripsi blok *ciphertext* pertama di-XOR-kan dengan IV untuk menghasilkan blok *plaintext* pertama. Walaupun nilai IV tidak perlu dirahasiakan akan tetapi integritas dari nilai IV harus dilindungi [3].

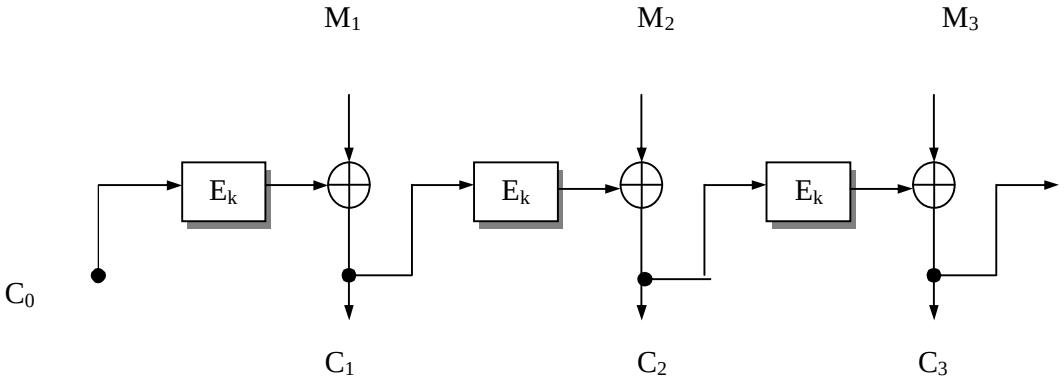


$C_i = E_k (C_{i-1} \oplus M_i)$; $M_i = C_{i-1} \oplus D_k (C_i)$

Gambar 2.4 Mode Cipher Block Chaining (CBC)

2.2.3.3 Mode Operasi CFB (Cipher Feedback)

Dalam mode *Cipher Feedback* (CFB) blok *ciphertext* sebelumnya di-enkrip dan outputnya digabungkan dengan blok *plaintext* dengan menggunakan XOR untuk menghasilkan blok *ciphertext* sekarang. Kita dapat mendefinisikan mode CFB sedemikian hingga mode tersebut menggunakan *feedback* yang lebih kecil dari 1 blok penuh data. Sebuah vektor inisialisasi C_0 digunakan sebagai sebuah "seed" untuk prosesnya, seperti terlihat pada Gambar 2.5 di bawah ini [3].

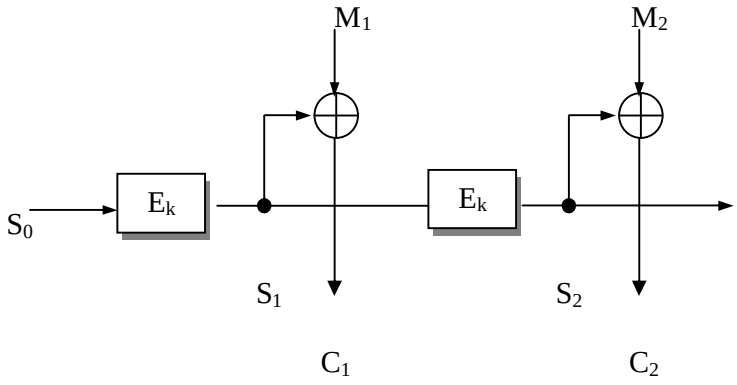


$C_i = E_k (C_{i-1}) \oplus M_i$; $M_i = E_k (C_{i-1}) \oplus C_i$

Gambar 2.5 Mode Cipher Feedback

2.2.3.4 Mode Output Feedback

Mode *Output Feedback* (OFB mode) mirip dengan mode CFB kecuali bahwa jumlah operasi XOR dengan setiap blok *plaintext* dihasilkan secara independen dari baik *plaintext* maupun *ciphertext*. Sebuah vektor Inisialisasi C_0 digunakan sebagai suatu "seed" untuk sebarisan blok data s_i, dan setiap blok data s_i diperoleh dari proses enkripsi terhadap blok data s_{i-1} sebelumnya. Proses enkripsi blok *plaintext* diperoleh dengan melakukan operasi XOR antara blok *plaintext* dengan blok data yang relevan [3].



$C_i = M_i \oplus S_i;\ M_i = C_i \oplus S_i;\ S_i = E_k (S_{i-1})$

Gambar 2.6 Mode Output Feedback

2.2.4 Stream Cipher

Stream cipher adalah jenis algoritma enkripsi simetri. *Stream cipher* dapat dibuat sangat cepat sekali, jauh lebih cepat dibandingkan dengan algoritma *block cipher* yang manapun. Algoritma *block cipher* secara umum digunakan untuk unit *plaintext* yang besar sedangkan *stream cipher* digunakan untuk blok data yang lebih kecil, biasanya ukuran bit. Proses enkripsi terhadap *plaintext* tertentu dengan

algoritma *block cipher* akan menghasilkan *ciphertext* yang sama jika kunci yang sama digunakan. Dengan *stream cipher*, transformasi dari unit *plaintext* yang lebih kecil ini berbeda antara satu dengan lainnya, tergantung pada kapan unit tersebut ditemukan selama proses enkripsi.

Satu *stream cipher* menghasilkan apa yang disebut suatu *keystream* (suatu barisan bit yang digunakan sebagai kunci). Proses enkripsi dicapai dengan menggabungkan *keystream* dengan *plaintext* biasanya dengan operasi *bitwise* XOR. Pembentukan *keystream* dapat dibuat independen terhadap *plaintext* dan *ciphertext*, menghasilkan apa disebut dengan *synchronous stream cipher*, atau dapat dibuat tergantung pada data dan enkripsinya, dalam hal mana *stream cipher* disebut sebagai *self-synchronizin*g. Kebanyakan bentuk *stream cipher* adalah *synchronous stream cipher*s.

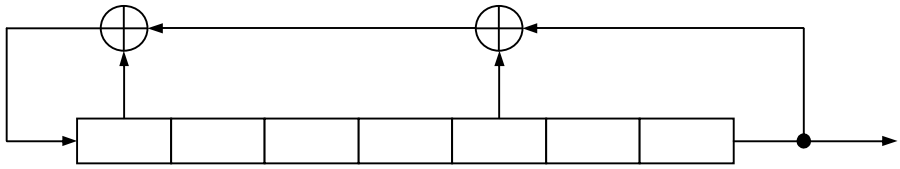
Konsentrasi dalam *stream ciphers* pada umumnya berkaitan dengan sifat-sifat teoritis yang menarik dari *one-time pad*. Suatu *one-time pad*, kadang-kadang disebut *Vernam cipher*, menggunakan sebuah string dari bit yang dihasilkan murni secara random. *Keystream* memiliki panjang sama dengan pesan *plaintext; string random* digabungkan dengan menggunakan *bitwise* XOR dengan *plaintext* untuk menghasilkan *ciphertext*. Karena *keystream* seluruhnya adalah *random*, walaupun dengan sumber daya komputasi tak terbatas seseorang hanya dapat menduga *plaintext* jika dia melihat *ciphertext*. Metode *cipher* seperti ini disebut memberikan kerahasiaan yang sempurna (*perfect secrecy*).

Metode *stream cipher* yang umum digunakan adalah RC4. Satu hal yang menarik bahwa mode operasi tertentu dari suatu *block cipher* dapat men-

transformasikan secara efektif hasil operasi tersebut ke dalam satu *keystream generator* dan dalam hal ini, *block cipher* apa saja dapat digunakan sebagai suatu *stream cipher*; seperti dalam DES, CFB atau OFB. Akan tetapi, *stream ciphers* dengan desain khusus biasanya jauh lebih cepat [1].

2.2.4.1 Linear Feedback Shift Register

Suatu *Linear Feedback Shift Register* (LFSR) adalah suatu mekanisme untuk menghasilkan sekuens bit biner. *Register* memiliki sebarisan sel yang ditentukan oleh vektor inisialisasi. Tingkah laku *register* diatur oleh sebuah *counter* (*clock*). Pada setiap saat isi sel dari *register* digeser (*shift*) ke kanan sejauh satu posisi, dan hasil operasi XOR terhadap subset dari isi sel ditempatkan pada sel paling kiri. Satu bit dari output biasanya diturunkan selama prosedur *update* ini.



Gambar 2.7 Linear Feedback Shift Register (LFSR)

LFSR cepat dan mudah untuk diimplementasikan baik secara *hardware* maupun secara *software*. Dengan pemilihan yang cermat terhadap alat kontrol (*pad*) *feedback* (bit tertentu yang digunakan, dalam Gambar 2.7 di atas *tap* adalah bit pertama dan bit kelima) sekuens yang dihasilkan dapat memiliki performansi yang baik secara statistik. Akan tetapi, sekuens yang dihasilkan oleh LFSR tunggal tidak cukup aman karena sebuah kerangka matematik yang kuat telah dikembangkan selama bertahun-tahun yang memungkinkan analisis yang mudah

terhadap hasil tersebut. Namun demikian, LFSR sangat berguna sebagai blok bangunan dalam sistem yang lebih aman [1].

2.2.4.2 Nonlinear Feedback Shift Register

Mudah untuk membayangkan suatu rangkaian umpan balik yang lebih rumit daripada yang digunakan LFSR. Masalahnya adalah tidak terdapat suatu teori matematika untuk menganalisanya. Secara khusus beberapa masalah dengan rangkaian umpan balik dari *Nonlinear Feedback Shift Register*, antara lain :

1. Mungkin terdapat prasangka atau bias seperti terdapat lebih banyak satu daripada nol atau lebih sedikit langkah yang diharapkan pada keluaran.
2. Periode maksimum dari rangkaian mungkin jauh lebih rendah dari yang diharapkan.
3. Periode rangkaian dapat berbeda untuk nilai awal yang berbeda.
4. Rangkaian dapat muncul secara sesaat tetapi kemudian terjadi dead end pada suatu nilai tunggal. Ini dapat diselesaikan dengan meng-XOR-kan fungsi linier dengan bit paling kanan.

Sebaliknya jika tidak ada teori untuk menganalisis segi keamanan umpan balik *Nonlinear Feedback Shift Register*, terdapat sedikit perangkat *cryptanalisis stream cipher* yang didasarkan padanya. Kita dapat menggunakan umpan balik *Nonlinear Feedback Shift Register* pada rancangan *stream cipher* tetapi kita harus berhati-hati [1].

2.2.5 Fungsi Hash

Sebuah nilai hash dihasilkan oleh fungsi H dengan bentuk

$$h = H(M)$$

dimana M adalah pesan dengan panjang bebas, dan H(M) atau h adalah nilai hash dengan panjang tetap. Nilai hash ini akan ditambahkan di awal pesan dan kemudian dikirimkan bersamaan. Penerima dari pesan itu akan melakukan otentikasi terhadap pesan itu dengan mengkomputasi nilai hash dan kemudian membandingkannya dengan nilai hash yang ada di awal pesan.

Agar fungsi hash bisa bertindak sebagai otentikasi pesan, fungsi hash H harus memenuhi syarat-syarat di bawah ini :

1. H dapat diterapkan pada blok data dengan ukuran berapa pun.
2. H menghasilkan keluaran dengan panjang yang tetap.
3. Relatif mudah untuk mengkomputasi H(x) untuk sembarang x.
4. Dengan mengetahui nilai hash y, tidaklah bisa secara komputasi untuk menemukan x yang memenuhi H(x) = y.
5. Dengan mengetahui x, tidaklah bisa secara komputasi untuk menemukan $y \neq x$ dengan H(y) = H(x).
6. Tidak bisa secara komputasi untuk menemukan pasangan (x,y) yang memenuhi H(x) = H(y).

Tiga syarat pertama berfungsi sebagai syarat penerapan praktis dari fungsi hash untuk otentikasi pesan.

Syarat keempat adalah syarat “satu-arah”. Mudah untuk menghasilkan kode nilai hash dengan diberikan sebuah pesan tetapi tidak bisa untuk menghasilkan pesan dengan diberikan kode nilai hash.

Syarat kelima menjamin bahwa dengan diberikan pesan asli tidak akan bisa ditemukan suatu pesan lain dengan nilai hash yang sama dengan nilai hash pesan aslinya. Syarat ini adalah untuk mencegah penipuan jika digunakan kode hash yang terenkripsi. Untuk kasus ini, pihak lawan bisa membaca pesan dan juga menghasilkan kode hashnya. Tetapi karena pihak lawan tersebut tidak memiliki kunci untuk mendekripsikan kode hash tersebut, maka ia tidak akan bisa mengubah pesan tanpa diketahui. Jika syarat ini tidak dipenuhi maka pihak lawan bisa melakukan hal-hal berikut. Pertama, mengintersepsi suatu pesan dengan kode hash terenkripsinya; kedua, menghasilkan kode hash yang tidak terenkripsi dari pesan tersebut; ketiga, membuat pesan baru yang lain dengan kode nilai hash yang sama dan kemudian mengirimkannya kembali.

Jika kelima syarat di atas dipenuhi, maka fungsi hash tersebut dinamakan fungsi hash yang lemah. Jika syarat keenam juga dipenuhi, maka fungsi hash itu dianggap fungsi hash yang kuat [3].

2.2.5.1 Message Authentication Codes

Suatu *message authentication code* (MAC) adalah sebuah label otentikasi (juga disebut suatu *checksum*) yang diturunkan dengan menggunakan suatu *authentication scheme*, bersama-sama dengan satu kunci rahasia, terhadap sebuah pesan. Tidak seperti *digital signatures*, MAC dihitung dan diverifikasi dengan

kunci yang sama, sehingga MAC hanya dapat diverifikasi oleh orang yang seharusnya menerima pesan.

Terdapat empat jenis MAC yakni :

1. *Unconditionally secure*

   Simmons dan Stinson mengusulkan satu *unconditionally secure MAC* didasarkan kepada enkripsi dengan satu *one-time pad*. *Ciphertext* dari pesan mengotentikasi dirinya sendiri, karena tidak ada orang lain yang memiliki akses ke *one-time pad*. Namun, terdapat beberapa redundansi dalam pesan. Suatu *unconditionally secure MAC* dapat juga diperoleh dengan menggunakan satu kunci rahasia *one-time*.

2. *Hash function-based*

   *Hash function-based MAC*s (sering disebut HMACs) menggunakan satu atau lebih kunci bersama-sama dengan satu fungsi hash untuk menghasilkan satu *checksum* yang ditambahkan ke akhir dari pesan. Salah satu contoh tipe ini adalah metode autentikasi pesan *keyed-MD5*.

3. *Stream cipher-based*

   Lai, Rueppel, dan Woolven mengusulkan suatu MAC yang didasarkan kepada algoritma *stream ciphers*. Dalam algoritma mereka, sebuah *stream cipher* yang dapat dibuktikan keamanannya digunakan untuk memecahkan suatu pesan atas dua bagian (substream) dan masing-masing substream diberikan kepada satu LFSR; proses *checksum* adalah keadaan akhir dari kedua LFSR.

4. *Block cipher-based*

MAC dapat juga diturunkan dari *block ciphers.* DES-CBC MAC merupakan standard USA dan internasional. Ide dasar adalah dengan melakukan enkripsi terhadap blok pesan dengan menggunakan DES-CBC dan output blok terakhir dalam *ciphertext* sebagai suatu *checksum*. Bellare et al. memberikan analisis tingkat keamanan dari MAC ini [3].

2.2.5.2 *Digital Signature*

Otentikasi pesan dengan fungsi hash memang berhasil melindungi kedua belah pihak yang saling bertukar pesan dari pihak ketiga. Tetapi, otentikasi pesan tidak bisa mencegah kemungkinan kedua belah pihak saling menyerang satu sama lain.

Sebagai contoh, A mengirim sebuah pesan yang dilengkapi dengan nilai hash yang dienkripsi dengan algoritma kunci simetris kepada B. Kemungkinan di bawah ini bisa terjadi :

1. B bisa memalsukan pesan lain dan mengaku bahwa pesan itu dikirim oleh A. B hanya perlu untuk membuat pesan lain dan menambahkan nilai hash yang dienkripsi dengan kunci yang diketahui kedua belah pihak.
2. A bisa menyangkal bahwa ia telah mengirimkan pesan kepada B karena mungkin saja B memalsukan pesan tersebut. Sama sekali tidak ada cara untuk membuktikan bahwa A memang mengirimkan pesan tersebut.

Pada situasi dimana tidak ada kepercayaan penuh antara pengirim dan penerima pesan, diperlukan suatu mekanisme yang lebih daripada sekedar otentikasi. Solusi yang paling menarik dari masalah ini adalah tanda tangan digital

(*digital signature*). Tanda tangan digital adalah suatu mekanisme otentikasi yang memungkinkan pembuat pesan menambahkan sebuah kode yang bertindak sebagai tanda tangannya. Tanda tangan tersebut menjamin integritas dan sumber dari sebuah pesan.

Tanda tangan digital dalam banyak hal mirip dengan tanda tangan biasa. Ada beberapa fungsi dari tandatangan digital yaitu :

1. Untuk memverifikasi pembuat pesan serta waktu ditandatanganinya pesan tersebut.
2. Untuk mengotentikasi isi pesan pada waktu ditandatangani pesan tersebut.
3. Harus bisa diverifikasi oleh pihak ketiga untuk menyelesaikan perselisihan mengenai integritas dan sumber pesan.

Dari fungsi-fungsi di atas maka dapat disusun persyaratan-persyaratan yang harus dipenuhi oleh sebuah tanda tangan digital yaitu :

1. Tanda tangan tersebut haruslah berupa pola bit yang tergantung pada pesan yang ditandatangani.
2. Tanda tangan tersebut harus menggunakan beberapa informasi yang menunjukkan pengirimnya, baik untuk mencegah pemalsuan pesan atau pengingkaran.
3. Relatif mudah untuk mengenali dan memverifikasi tanda tangan digital.
4. Relatif mudah untuk mengenali dan memverifikasi tanda tangan digital.
5. Tidak bisa secara komputasi untuk memalsukan sebuah tanda tangan digital, baik dengan mengkonstruksi sebuah pesan baru baru dari tanda tangan digital

yang ada ataupun dengan mengkonstruksi sebuah tanda tangan digital dengan menggunakan pesan yang diberikan.

6. Duplikat tanda tangan digital tersebut gampang disimpan.

Persyaratan-persyaratan di atas dapat dipenuhi dengan mengkombinasikan fungsi hash dengan enkripsi public-key. Jadi setelah pengirim pesan A menambahkan nilai hash dari pesan tersebut, selanjutnya nilai hash itu akan dienkripsi dengan menggunakan kunci pribadi pengirim dan kemudian mengirimkan pesan itu kepada B. B kemudian mendekripsi nilai hash tersebut dengan menggunakan kunci publik A. Karena hanya A yang bisa mengenkripsi nilai hash tersebut maka pastilah pesan itu berasal dari A [3].

2.3 Aplikasi Kriptografi

2.3.1 *Privacy*

Privacy (kerahasiaan) mungkin merupakan aplikasi paling nyata dari kriptografi. Kriptografi dapat digunakan untuk mengimplementasikan *privacy* hanya dengan mengenkrip informasi yang diinginkan untuk tetap *private*. Agar seseorang dapat membaca data *private* ini dia harus mendekrip terlebih dahulu. Kadang-kadang informasi tertentu bukan untuk diakses oleh siapapun juga, dan dalam hal ini informasi dapat disimpan sedemikian rupa sehingga membalik proses merupakan sesuatu yang secara virtual tidak mungkin. Misalnya, dalam sistem *multi-user*, tidak ada satu orangpun dimungkinkan untuk mengetahui daftar *password* dari masing-masing *user* dalam sistem. Biasanya nilai hash dari *password* yang disimpan bukan *password* itu sendiri. Hal ini memungkinkan *user*

dari sistem yakin betul tentang informasi pribadi disimpan betul-betul aman dari gangguan orang lain karena dengan memasukkan *password* harus diverifikasi terlebih dahulu (dengan menghitung fungsi hashnya dan membandingkan dengan nilai hash yang tersimpan) [3].

2.3.2 *Digital Signature* dan *Authentication*

Authentication adalah suatu proses untuk membuktikan dan memverifikasi informasi tertentu. Kadang-kadang seseorang ingin memverifikasi asal dokumen, indentitas pengirim, waktu dan tanggal penandatanganan dan/atau pengiriman, identitas komputer atau user dan lain-lain. Suatu *digital signature* adalah cara *cryptography* dimana dengan cara tersebut beberapa hal di atas dapat diverifikasi. Tanda tangan dijital dari suatu dokumen adalah potongan informasi yang didasarkan kepada dokumen dan kunci rahasia penanda-tangan. Tanda tangan ini biasanya diciptakan melalui penggunaan fungsi hash dan fungsi tanda tangan privat (enkripsi kunci rahasia penanda tangan), tetapi masih ada metode lain.

Setiap hari orang menandatangani nama mereka dalam surat, bukti penggunaan kartu kredit, dan dokumen lainnya, yang menunjukkan bahwa mereka setuju dengan isi dokumen tersebut. Dalam hal ini, mereka melakukan otentikasi bahwa mereka dalam kenyataannya adalah pemilik atau pengirim atau sumber dari dokumen. Hal ini memungkinkan orang lain untuk melakukan verifikasi bahwa pesan khusus benar-benar berasal dari si penanda-tangan dokumen. Akan tetapi, cara ini bukanlah bebas dari kemungkinan pencurian atau pemalsuan dari pihak ketiga, karena orang dapat “mengangkat” tanda-tangan dari dalam dokumen dan

menempatkannya ke dokumen lain, dengan demikian menghasilkan dokumen asli tapi palsu (aspal). Tanda tangan konvensional (dengan tinta) juga tidak aman dari pemalsuan karena dimungkinkan untuk mereproduksi sebuah tanda tangan pada dokumen lain atau mengubah dokumen setelah dokumen ditanda-tangani. Tanda tangan dijital dan tulisan tangan tergantung pada fakta bahwa sulit untuk mendapatkan dua orang dengan tanda tangan sama [3].

2.3.3 *Key Agreement Protocol*

Suatu *key agreement protocol*, juga dikenal dengan *key exchange protocol*, adalah sebarisan langkah yang dilakukan bila dua atau lebih pihak perlu sepakat atas suatu kunci yang digunakan untuk suatu *secret-key cryptosystem*. Protokol ini memungkinkan orang menggunakan kunci secara bersama dengan bebas dan aman melalui suatu medium yang tidak aman, tanpa perlu terlebih dahulu ada pembentukan kunci rahasia bersama.

Misalkan Ali dan Bob ingin menggunakan satu *secret-key cryptosystem* untuk berkomunikasi secara aman. Mereka terlebih dahulu harus memutuskan apa kunci yang mereka gunakan. Bob bukannya menelepon Ali dan mendiskusikan apa kunci yang mereka gunakan, yang bisa saja didengar oleh pihak ketiga, mereka memutuskan menggunakan satu protokol kesepakatan kunci (*key agreement protocol*). Dengan menggunakan *key agreement protocol*, Ali dan Bob dapat saling mempertukarkan kunci dalam lingkungan yang tidak aman. Salah satu contoh protokol ini adalah Diffie-Hellman *key agreement*. Dalam beberapa kasus, *public-key cryptography* digunakan dalam suatu *key agreement protocol*.

Contoh lainnya adalah penggunaan amplop dijital (*digital envelopes*) untuk *key agreement*.

2.3.4 *Identification* (Identifikasi)

Identification (identifikasi) adalah suatu proses melalui mana seseorang yakin tentang identitas orang lain atau entittas tertentu. Dalam kehidupan kita sehari-hari kita mengidentifikasikan anggota keluarga kita, kawan, dan teman sejawat dengan karakteristik fisik mereka, seperti suara, muka atau karakteristik lainnya. Karakteristik ini disebut *biometrics*, yang hanya dapat digunakan pada jaringan dengan perangkat khusus. Entitas dalam sebuah jaringan dapat juga mengidentifikasikan entitas lain dengan menggunakan metode kriptografi.

Otentikasi dan identifikasi adalah dua hal yang berbeda. Identifikasi mengharuskan *verifier* (pelaku verifikasi) membandingkan informasi yang diberikan terhadap semua entitas yang diketahuinya, sedangkan otentikasi membutuhkan pengecekan informasi tentang entitas tunggal yang diberikan dan diidentifikasikan sebelumnya. Selanjutnya, identifikasi harus mengidentifikasikan entitas secara unik, sementara autentikasi tidak mengharuskan keunikan. Sebagai contoh, seseorang yang sedang *log into* rekening bersama tidak diidentifikasikan secara unik, tetapi dengan mengetahui *password* bersama, mereka diotentikasikan sebagai salah satu pemilik/pengguna rekening tersebut. Kemudian, identifikasi tidak perlu mengotentikasikan pengguna dalam maksud tertentu [3].

2.4 Dasar-Dasar Matematika Kriptografi

Beberapa operasi dasar matematika yang digunakan dalam kriptografi metode WAKE adalah operasi AND, OR, XOR, Penjumlahan Modulo dan *Shift Right*.

2.4.1 AND

Operasi *AND* dari dua *input* A dan B hanya akan bernilai bit "1" apabila kedua bit *input* A dan B bernilai bit "1". Atau dengan kata lain *output* dari operasi *AND* akan memiliki nilai bit "0" apabila salah satu *input*-nya bernilai bit "0". Operasi AND dilambangkan dengan tanda "$\wedge$".

Aturan operasi AND dapat dinyatakan seperti tabel berikut :

Tabel 2.1 Aturan operasi AND

A	**B**	**A $\wedge$ B**
0	0	0
0	1	0
1	0	0
1	1	1

Contoh :

```
  11000110
  10110011
------------ ∧
  10000010
```

2.4.2 OR

Operasi OR dari dua *input* A dan B hanya akan bernilai bit "0" apabila kedua bit *input* A dan B bernilai bit "0". Atau dengan kata lain *output* dari operasi OR akan memiliki nilai bit "1" apabila salah satu inputnya bernilai bit "1". Operasi OR dilambangkan dengan tanda "$\vee$".

Aturan operasi *OR* dapat dinyatakan seperti tabel berikut :

Tabel 2.2 Aturan operasi OR

A	**B**	**A $\vee$ B**
0	0	0
0	1	1
1	0	1
1	1	1

Contoh :

```
  11000110
  10110011
------------ ∨
  11110111
```

2.4.3 XOR

XOR adalah operasi *Exclusive*-OR yang dilambangkan dengan tanda "$\oplus$". Hasil dari operasi XOR akan bernilai bit "0" (nol) jika dua buah bit *input* memiliki nilai yang sama dan akan menghasilkan nilai bit "1" (satu) jika dua buah

bit *input* memiliki nilai bit yang berbeda. Aturan operasi XOR dapat dirumuskan seperti tabel berikut ini :

Tabel 2.3 Aturan Operasi XOR

A	**B**	**A ⊕ B**
0	0	0
0	1	1
1	0	1
1	1	0

Contoh :

```
  11000110
  10110011
------------ ⊕
  01110101
```

2.4.4 Penjumlahan Modulo

Penjumlahan bit modulo yang digunakan dalam metode WAKE adalah penjumlahan dua buah bit bilangan yang sama panjang dan menghasilkan bilangan dengan panjang bit yang sama pula. Jika panjang bit bilangan lebih besar, maka bit bilangan yang berlebihan tersebut akan dibuang.

Contoh : 10111101 + 10010101 = **1** 01010010

Bit 1 yang berlebihan di depan akan dibuang sehingga hasil yang didapatkan

dari proses penjumlahan di atas adalah 01010010.

2.4.5 Pergeseran Bit (Shift)

Pergeseran bit (*Shift*) adalah operasi pergeseran terhadap suatu barisan bit sebanyak yang diinginkan. Bit kosong yang telah tergeser akan diberikan nilai bit "0" (nol).

Operasi pergeseran terbagi menjadi dua macam yaitu :

1. Operasi Geser Kiri (*Shift Left*) yaitu operasi yang menggeser sejumlah bit ke kiri (*left*) dan mengisi tempat kosong dengan nilai bit "0" (nol). Operasi *shift left* dilambangkan dengan "**<<**". Contoh operasi *shift left* :

   ```
   11000110 << 1 : 10001100
   11000110 << 2 : 00011000
   ```

2. Operasi Geser Kanan (*Shift Right*) yaitu operasi yang menggeser sejumlah bit ke kanan (*right*) dan mengisi tempat kosong dengan nilai bit "0" (nol). Operasi *shift right* dilambangkan dengan "**>>**". Contoh operasi *shift right* :

   ```
   11000110 >> 1 : 01100011
   11000110 >> 2 : 00110001
   ```

2.4.6 Konversi Bilangan Berbasis

Bilangan – bilangan berbasis dapat diubah atau dikonversikan satu sama lain. Proses pengubahan bilangan berbasis yang akan dibahas antara lain :

1. Pengubahan bilangan biner ke bilangan heksadesimal
2. Pengubahan bilangan heksadesimal ke bilangan biner

2.4.6.1 Konversi dari Bilangan Biner ke Bilangan Heksadesimal

Proses konversi bilangan biner ke bilangan heksadesimal dapat dilakukan dengan 2 cara, yaitu secara langsung dan secara tidak langsung. Proses konversi bilangan biner ke bilangan heksadesimal secara langsung dapat dilakukan dengan menggunakan algoritma berikut :

1. Jika jumlah digit bilangan biner bukan kelipatan 4, maka tambahkan bilangan 0 di depan bilangan biner hingga jumlah digit merupakan kelipatan 4.
2. Pisahkan bilangan biner ke dalam bentuk kelompok empatan.
3. Konversi masing – masing kelompok empatan tersebut ke dalam bilangan heksadesimal dengan menggunakan tabel sistem bilangan di atas.

Sebagai contoh, diambil bilangan biner 1100101101, maka proses konversi bilangan biner tersebut ke dalam bentuk bilangan heksadesimal adalah sebagai berikut,

1. Jumlah digit bilangan biner 1100101101 ada sebanyak 10 buah dan bukan merupakan kelipatan 4, sehingga harus ditambahkan 2 buah bilangan 0 di depan bilangan biner tersebut agar jumlah digit merupakan kelipatan 4.
   1100101101 à **00**1100101101
2. Pisahkan bilangan biner tersebut ke dalam bentuk kelompok empatan.
   001100101101 à 0011 | 0010 | 1101
3. Konversi masing – masing kelompok empatan tersebut ke dalam bilangan heksadesimal dengan menggunakan tabel sistem bilangan di atas.

0011 | 0010 | 1101

↓ ↓ ↓

3 2 D

4. Sehingga bilangan heksadesimal yang didapat adalah 32D.

Proses pengubahan bilangan biner ke bilangan heksadesimal secara tidak langsung dapat dilakukan dengan langkah – langkah seperti berikut,

1. Konversikan bilangan biner ke dalam bentuk bilangan desimal.
2. Konversikan bilangan desimal hasil perhitungan tersebut ke dalam bentuk bilangan heksadesimal.

Untuk mengubah satu bilangan biner ke kesetaraan desimalnya, jumlahkan kesetaraan desimal masing – masing posisi 1-nya. Sebagai contoh, diambil bilangan biner 1100101101 di atas.

$1100101101 = 1 \text{ X } 2^9 + 1 \text{ X } 2^8 + 1 \text{ X } 2^5 + 1 \text{ X } 2^3 + 1 \text{ X } 2^2 + 1 \text{ X } 2^0$

$= 512 + 256 + 32 + 8 + 4 + 1 = 813$

Sedangkan, untuk mengubah satu bilangan bulat desimal ke kesetaraan heksadesimalnya, bagilah secara berurutan bilangan tersebut dengan 16 dan konversikan angka sisanya ke dalam bentuk heksadesimal dengan urutan terbalik.

813 / 16 = 50 sisa 13 à D ↑

50 / 16 = 3 sisa 2 à 2 |

3 / 16 = 0 sisa 3 à 3 |

Sehingga bilangan heksadesimal yang didapat adalah 32D.

2.4.6.2 Konversi dari Bilangan Heksadesimal ke Bilangan Biner

Proses konversi bilangan heksadesimal ke bilangan biner juga dapat dilakukan dengan 2 cara, yaitu secara langsung dan secara tidak langsung. Proses pengubahan bilangan heksadesimal ke bilangan biner secara langsung dapat dilakukan dengan menggunakan algoritma berikut :

1. Konversikan masing – masing digit bilangan heksadesimal ke dalam bentuk bilangan biner 4 digit.
2. Gabungkan semua bilangan biner hasil konversi tersebut sesuai dengan urutan posisinya.

Sebagai contoh, diambil bilangan heksadesimal 32D, maka proses konversi bilangan heksadesimal tersebut ke dalam bentuk bilangan biner adalah sebagai berikut,

1. Konversikan masing – masing digit bilangan heksadesimal tersebut ke dalam bentuk bilangan biner 4 digit.

   3 2 D
   ↓ ↓ ↓
   0011 0010 1101

2. Gabungkan semua bilangan biner hasil konversi tersebut sesuai dengan urutan posisinya.

   0011 0010 1101 à 001100101101

3. Sehingga bilangan biner yang didapat adalah 001100101101.

Proses pengubahan bilangan heksadesimal ke bilangan biner secara tidak langsung dapat dilakukan dengan langkah – langkah seperti berikut,

1. Konversikan bilangan heksadesimal ke dalam bentuk bilangan desimal.
2. Konversikan bilangan desimal hasil perhitungan tersebut ke dalam bentuk bilangan biner.

Untuk mengubah satu bilangan heksadesimal ke kesetaraan desimalnya, jumlahkan kesetaraan desimal masing – masing posisinya. Sebagai contoh, diambil bilangan heksadesimal 32D di atas.

32D $= 3 \text{ X } 16^2 + 2 \text{ X } 16^1 + \text{D X } 16^0$

$= 768 + 32 + 13 = 813$

Sedangkan, untuk mengubah satu bilangan bulat desimal ke kesetaraan binernya, bagilah secara berurutan bilangan tersebut dengan 2 dan catatlah angka sisanya dengan urutan terbalik.

813 / 2 = 406 sisa 1 à 1
406 / 2 = 203 sisa 0 à 0
203 / 2 = 101 sisa 1 à 1
101 / 2 = 50 sisa 1 à 1
50 / 2 = 25 sisa 0 à 0
25 / 2 = 12 sisa 1 à 1
12 / 2 = 6 sisa 0 à 0
6 / 2 = 3 sisa 0 à 0
3 / 2 = 1 sisa 1 à 1
1/2 = 0 sisa 1 à 1



Sehingga bilangan biner yang didapat adalah 1100101101.

2.5 WAKE (Word Auto Key Encryption)

Metode WAKE merupakan salah satu algoritma *stream cipher* yang telah digunakan secara komersial. WAKE merupakan singkatan dari *Word Auto Key Encryption*. Metode ini ditemukan oleh David Wheeler pada tahun 1993.

Metode WAKE menggunakan kunci 128 bit dan sebuah tabel 256 x 32 bit. Dalam algoritmanya, metode ini menggunakan operasi XOR, AND, OR dan *Shift Right*.

Proses utama WAKE terdiri dari :

1. Proses pembentukan tabel *S-Box* (*Substitution Box*).
2. Proses pembentukan kunci.
3. Proses enkripsi dan dekripsi.

Inti dari metode WAKE terletak pada proses pembentukan tabel *S-Box* dan proses pembentukan kunci. Tabel *S-Box* dari metode WAKE bersifat fleksibel dan berbeda-beda untuk setiap putaran [5].

2.5.1 Proses Pembentukan Tabel S-Box

Proses pembentukan tabel *S-Box* adalah sebagai berikut [5] :

1. Inisialisasi nilai TT[0] … TT[7] :

   TT[0] : 726a8f3b (dalam heksadesimal)

   TT[1] : e69a3b5c (dalam heksadesimal)

   TT[2] : d3c71fe5 (dalam heksadesimal)

   TT[3] : ab3c73d2 (dalam heksadesimal)

TT[4] : 4d3a8eb3 (dalam heksadesimal)

TT[5] : 0396d6e8 (dalam heksadesimal)

TT[6] : 3d4c2f7a (dalam heksadesimal)

TT[7] : 9ee27cf3 (dalam heksadesimal)

2. Inisialisasi nilai awal untuk T[0] … T[3] :

   T[0] = K[0] T[2] = K[2]

   T[1] = K[1] T[3] = K[3]

   K[0], K[1], K[2], K[3] dihasilkan dari kunci yang dipecah menjadi 4 bagian yang sama panjang.

3. Untuk T[4] sampai T[255], lakukan proses berikut :

   X = T[n-4] + T[n-1]

   T[n] = X >> 3 XOR TT(X AND 7)

4. Untuk T[0] sampai T[22], lakukan proses berikut :

   T[n] = T[n] + T[n+89]

5. Set nilai untuk beberapa variabel di bawah ini :

   X = T[33]

   Z = T[59] OR (01000001h)

   Z = Z AND (FF7FFFFFh)

   X = (X AND FF7FFFFFh) + Z

6. Untuk T[0] … T[255], lakukan proses berikut :

   X = (X AND FF7FFFFFh) + Z

   T[n] = T[n] AND 00FFFFFFh XOR X

7. Inisialisasi nilai untuk beberapa variabel berikut ini :

T[256] = T[0]

X = X AND 255

8. Untuk T[0] … T[255], lakukan proses berikut :

   Temp = (T[n XOR X] XOR X) AND 255

   T[n] = T[Temp]

   T[X] = T[n+1]

2.5.2 Proses Pembentukan Kunci

Proses pembentukan kunci dari metode WAKE dapat ditentukan sendiri yaitu sebanyak n putaran. Semakin banyak putaran dari proses pembentukan kunci, maka keamanan datanya akan semakin terjamin. Fungsi yang digunakan dalam proses pembentukan kunci adalah M(X, Y) = (X + Y) >> 8 XOR T[(X + Y) AND 255].

Pertama-tama, kunci yang di-*input* akan dipecah menjadi 4 bagian dan di-set sebagai nilai awal dari variabel A_0, B_0, C_0, dan D_0. Nilai dari variabel ini akan diproses dengan melalui langkah berikut [6] :

$A_{i+1} = M(A_i, D_i)$

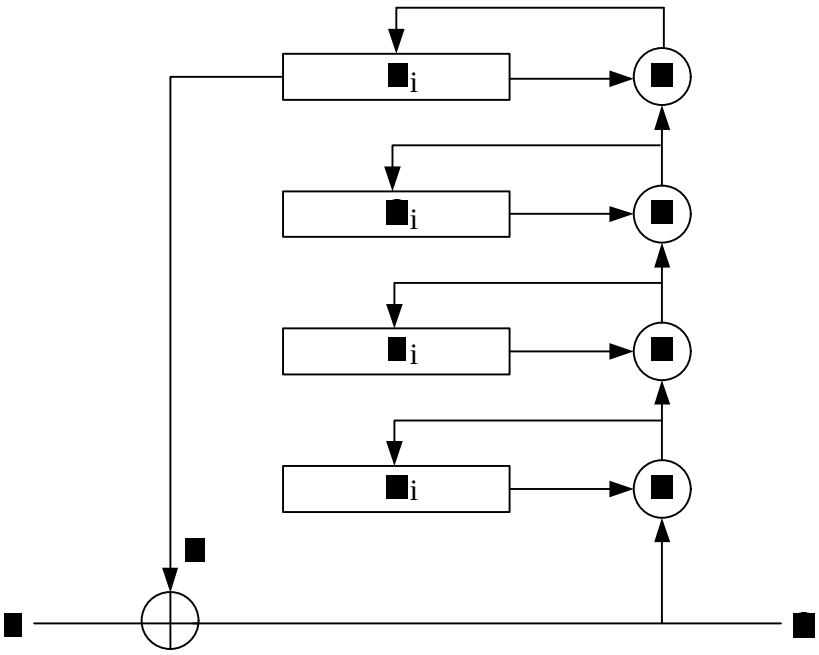
$B_{i+1} = M(B_i, A_{i+1})$

$C_{i+1} = M(C_i, B_{i+1})$

$D_{i+1} = M(D_i, C_{i+1})$

Nilai dari D_i merupakan nilai dari kunci K_i.

Agar lebih jelas, lihatlah bagan proses pembentukan kunci berikut :



Gambar 2.8 Bagan proses pembentukan kunci

Keterangan :

P	= *Plaintext*	Ai	= Bagian pertama dari pecahan kunci
K	= *Key*	Bi	= Bagian kedua dari pecahan kunci
C	= *Ciphertext*	Ci	= Bagian ketiga dari pecahan kunci
M	= Fungsi M	Di	= Bagian keempat dari pecahan kunci
i	= Dimulai dari 0 sampai n.		

2.5.3 Proses Enkripsi dan Dekripsi

Inti dari metode WAKE tidak terletak pada proses enkripsi dan dekripsinya, karena proses enkripsi dan dekripsinya hanya berupa operasi XOR dari *plaintext* dan kunci untuk menghasilkan *ciphertext* atau operasi XOR *ciphertext* dan kunci untuk menghasilkan *plaintext* [5].

$$P = C \oplus K$$

$$C = P \oplus K$$

dengan :

P = *Plaintext*

K = *Key*

C = *Ciphertext*