BAB II

LANDASAN TEORI

2.1 Digital Signature

Pada penelitian ini, aplikasi dibuat menggunakan *Digital Signature Standard* (DSS) yang dipublikasikan oleh *Federal Infromation Processing Standard* pada FIPS PUB 186-3 sebagai sistem kriptografi. Standar ini mendefinisikan metode-metode yang dapat digunakan untuk menghasilkan *digital signature*, di mana sistem dari *digital signature* ini dapat digunakan untuk membuat tanda tangan digital untuk sebuah file dan untuk memverifikasi keaslian dari tanda tangan digital file yang dihasilkan (NIST, 2009). Tanda tangan digital yang dihasilkan untuk setiap berkas digital hanya satu masing-masing berkas, di mana nanti setiap tanda tangan digital ini ditaruh ke dalam berkas digital. Metode *Digital Signature* menggunakan standarisasi *hash function* untuk proses verifikasi file yang telah dienkripsi dan dikirimkan oleh pihak pengirim. *Rivest-Shamir-Adleman* (RSA) merupakan salah satu algoritma kunci publik yang dianjurkan di dalam *Digital Signature Standard* sedangkan untuk algoritma hash menggunakan metode SHA-2.

Digital signature adalah sebuah metode pembanding digital dari sebuah tanda yang tertulis digital, sebuah tanda digital dapat digunakan untuk menyediakan sebuah jaminan untuk mengakui tanda dari berkas yang telah ditandatangani (NIST, 2009). Tanda tangan digital yang digunakan pada metode

ini bukan merupakan tanda tangan yang dipindai menggunakan alat *scanner*, tetapi tanda tangan digital dengan metode ini adalah tanda tangan digital yang merupakan hasil dari komputasi kritptografi dari berkas file digital dan pengirim berkas (Munir, 2005).

Secara garis besar proses pembentukan tanda tangan digital pada *digital signature* dapat dibagi menjadi dua proses yaitu pembentukan tanda tangan digital pada file dan proses verifikasi file (Subramanya, 2006).

Proses pembentukan tanda tangan digital dimulai dengan menjalankan fungsi *hash* (SHA256) untuk menghasilkan sebuah *digest file* dengan ukuran yang sudah pasti. Kemudian *digest file* tersebut dienkripsi dengan kunci privat yang telah dibentuk oleh algoritma RSA. Hasil dari enkripsi tersebut digabungkan dengan kunci publik dari algoritma RSA untuk membentuk sebuah tanda tangan digital. Tanda tangan digital ini dienkripsi lagi oleh algoritma *twofish* dengan menggunakan kunci yang dimasukan secara manual oleh pengguna.

Pada proses verifikasi tanda tangan yang ada di dalam berkas file yang telah dienkripsi oleh algoritma *twofish*. Sebelum proses verifikasi dimulai, pengguna terlebih dahulu mendekripsikan file hasil enkripsi algoritma *twofish*. Proses dekripsi file terdapat dua bagian yaitu dekripsi file dengan kunci simetrik dan dekripsi digest file dengan kunci publik.

Sesudah file tersebut didekripsikan, proses verifikasi dapat dijalankan. Proses verifikasi terdiri dari evaluasi digest dan perbandingan nilai digest. Evaluasi *digest* adalah file awal kembali diproses oleh algoritma hash pada saat diterima oleh penerima, karena algoritma hash merupakan algoritma yang

melakukan proses hashing one-way atau tidak dapat dibalikan ke kondisi awal. Oleh karena itu pengguna harus melakukan evaluasi digest terhadap file tersebut menggunakan algoritma hash yang digunakan oleh pengirim.

Pada perbandingan nilai *digest, digest* yang telah dienkripsi dibandingkan dengan digest yang file telah dievaluasi *digest* untuk memverifikasi tanda tangan digital yang ada pada file yang dikirim. Bila saat proses verifikasi ditemukan ketidakcocokan dari perbandingan itu, maka dapat disimpulkan bahwa file tersebut bukanlah file yang ditandatangani oleh pengirim dan file tersebut telah berubah akibat adanya campur tangan pihak ketiga.

2.2 *Rivest-Shamir-Adleman (RSA)*

Metode *Rivest-Shamir-Adleman* (RSA) merupakan salah satu metode dari kriptografi kunci publik, di mana pada kriptografi ini enkripsi dan dekripsi menggunakan sepasang kunci yaitu kunci publik dan kunci privat. File dienkripsi dengan menggunakan kunci privat dan dapat didekripsi dengan kunci publik dari pengirim. Dan sebaliknya file yang telah dienkripsi dengan kunci publik dari penerima hanya dapat didekripsi dengan menggunakan kunci privat penerima (Kromodimoeljo, 2010). Sebelum proses enkripsi algoritma *Rivest-Shamir-Adleman* (RSA) dimulai, proses enkripsi memerlukan parameter – parameter pembentuk sepasang kunci, maka dari itu proses manajemen kunci pada aplikasi berkas pengaman digital dimulai terlebih dahulu sebelum proses enkripsi berlangsung, proses manajemen kunci menghasilkan kunci yang unik dan memerlukan komputasi yang kompleks.

Pada RSA, proses yang dilakukan adalah membuat nilai acak pada parameter-parameter untuk dua bilangan prima (p dan q) untuk menghasilkan nilai n, sepasang kunci publik (e, n), dan sepasangan kunci privat (d, n) menggunakan metode *random number generator* (Kromodimoeljo, 2010). Nilai dari kedua bilangan prima (p dan q) harus bersifat rahasia atau dihancurkan setelah nilai n terbentuk, agar nilai n yang digunakan untuk membuat kunci publik dan kunci privat tidak dapat diketahui oleh pihak yang tidak bersangkutan. Proses pembuatan sepasang kunci berserta parameter-parameternya algoritma RSA secara garis besar ditunjukkan pada gambar 2.1.

1. Generate two large random primes, p and q, of approximately equal size such that their product n = p*q is of the required bit length, e.g. 1024 bits.
2. Compute n = p*q and (phi) φ = (p-1)(q-1).
3. Choose an integer e, 1 < e < phi, such that gcd(e, phi) = 1.
4. Compute the secret exponent d, 1 < d < phi, such that ed ≡ 1 (mod phi).
5. The public key is (n, e) and the private key (d, p, q). Keep all the values d, p, q and phi secret. [We prefer sometimes to write the private key as (n, d) because you need the value of n when using d. Other times we might write the key pair as ((N, e), d).]

n is known as the modulus.
e is known as the public key exponent.

Gambar 2.1 *Pseudocode* RSA (Menezes, 1996)

Setelah parameter-parameter yang dibutuhkan dihasilkan, maka selanjutnya adalah proses pembuatan kedua pasang kunci dengan menggunakan parameter-parameter tersebut. Kedua pasang kunci ini yaitu kunci publik dan kunci privat memiliki rumus untuk mengenkripsi dan mendekripsi suatu file asli atau file acak (*ciphertext*). Pada penelitian ini untuk proses enkripsi file asli menggunakan kunci privat dan untuk proses dekripsi menggunakan kunci publik user.

Kedua pasang kunci ini digunakan untuk mengenkripsi dan mendekripsi file digest yang terlebih dahulu diproses oleh metode SHA-2 menjadi file acak (*ciphertext*). Selanjutnya file acak tersebut diproses untuk verifikasi keaslian tanda tangan digital oleh pihak penerima. Proses verifikasi diawali dengan proses dekripsi pada file acak dengan menggunakan kunci privat. Di dalam file acak yang telah didekripsi terdapat file asli dan file *digest* , file asli ini di*hashing* seperti proses *hashing* awal, lalu hasil dari proses hash file asli dibandingkan dengan file *digest*, bila sama maka proses verifikasi berhasil dan sebaliknya.

2.3 *Secure Hashing Algorithm* – 2 (SHA-2)

SHA-2 merupakan algoritma hash yang dikembangkan dari SHA-1. SHA merupakan fungsi hash yang tidak dapat dikembalikan ke bentuk data awal atau *one-way*. *Hash value* tidak akan menghasilkan nilai yang sama ketika ada perubahan pada data. SHA256 mampu menghasilkan *digest file* atau *hash value* dengan panjang 256 bit (NIST, 2002). *Digest file* digunakan oleh RSA untuk menghasilkan sebuah tanda tangan digital untuk berkas file yang dienkripsi.

Proses pada SHA-2 terdiri dari dua proses utama untuk membuat digest file yaitu *preprocessing* dan *hashing*. Pada proses *preprocessing* ada tiga tahap yaitu menentukan nilai-nilai inisialisasi proses *hash*, melakukan *padding* pada file, dan menguraikan file menjadi sejumlah ukuran bit blok file (Kromodimoeljo, 2010). Tujuan dari *padding* file adalah untuk memastikan ukuran dari file yang merupakan kelipatan dari 512 atau 1024 bits (NIST, 2002). Bila file bukan merupakan kelipatan dari ukuran tersebut, maka file tersebut disisipkan atau ditambahkan dengan beberapa kumpulan karakter *padding* agar ukuran file

tersebut sesuai dengan ukuran kelipatan di atas. Penentuan banyaknya nilai inisialisasi proses *hash* bergantung pada ukuran file *digest*. Untuk SHA256 yang digunakan pada penelitian ini nilai inisialisasi proses *hash* menggunakan sebanyak delapan huruf yang masing-masing berukuran 32 bit dalam bentuk *hex*.

Proses kedua dari SHA-2 yaitu *hashing* ke setiap blok file yang ada. Setelah nilai inisialisasi untuk proses hash dibuat, maka *hashing* dilakukan terhadap setiap blok file yang telah dibuat. *Hashing* diawali dengan membagi data dari file menjadi 64 bagian yang diberi tanda W_0, W_1,…, W_{63} . Nilai inisialisasi yang telah ditetapkan diberikan 8 label yaitu a, b, c, d, e, f, g, dan h untuk menampung nilai-nilai tersebut, di mana masing-masing nilai ini menggunakan fungsi logika f_0, f_1…, f_{63} yang beroperasi pada 32-bit word yaitu x, y, z seperti pada fungsi 2.1.

$$Ch(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z)$$
$$Maj(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$$
$$\sum_0^{\{256\}}(x) = ROTR^{2}(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x)$$
$$\sum_1^{\{256\}}(x) = ROTR^{6}(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x)$$
$$\sigma_0^{\{256\}}(x) = ROTR^{7}(x) \oplus ROTR^{18}(x) \oplus SHR^{3}(x)$$
$$\sigma_1^{\{256\}}(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x)$$

Gambar 2.2. *Six logical function 32-bit word* (NIST, 2002).

Setelah kedelapan nilai inisialisasi diproses dengan menggunakan fungsi di atas, nilai dari hasil proses fungsi itu ditambahkan dengan masing-masing nilai awal sebelum diproses dengan fungsi tersebut. Proses-proses diatas diulang sebanyak N kali yang menghasilkan 256 bit digest file dari file yang diproses dengan metode SHA256.

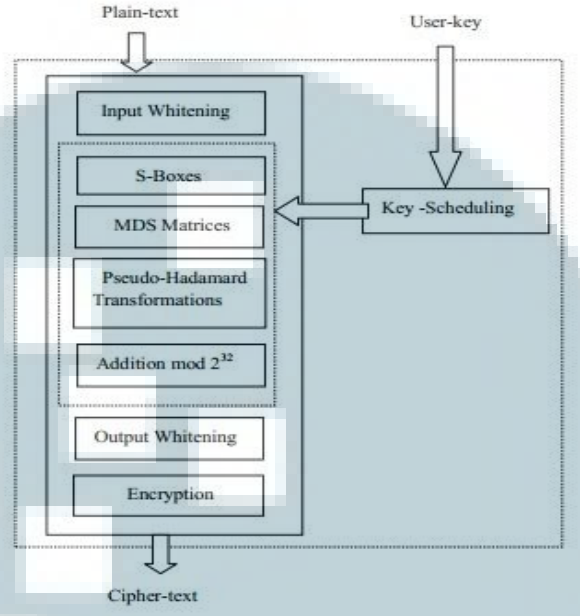
Seperti yang telah dijelaskan pada proses kedua *digital signature*, proses digest dari sebuah file merupakan proses yang tidak dapat dibalikan ke kondisi awal, oleh karena itu penerima dalam hal ini harus memasukkan file dari pengirim ke dalam fungsi SHA-2 kembali, untuk mendapatkan MD dan kemudian dibandingkan dengan MD yang ada di dalam berkas file yang diterima untuk memverifikasi keaslian file tersebut. Sehingga tidak mungkin menemukan dua file yang berbeda menghasilkan nilai *digest* atau MD yang sama. Setelah file *digest* terbentuk, file tersebut akan dienkripsi menggunakan kunci privat *Rivest-Shamir-Adleman* (RSA).

2.4 *Symmetric Cryptography Twofish*

Twofish merupakan merupakan 128-bit *block cipher* yang menerima variasi panjang kunci. *Block cipher* merupakan sebuah metode enkripsi yang memproses sebuah *block file* dari *plaintext* sebagai satu kesatuan dan digunakan untuk membuat *block ciphertext* dengan ukuran yang sama. *Block cipher* dapat beroperasi pada *block plaintext* yang berukuran n bits untuk menghasilkan *block ciphertext* yang berukuran n bits. Sebuah proses enkripsi harus bersifat reversibel oleh karena itu pada *block plaintext* terdapat 2n blok yang unik dan tiap blok tersebut mampu membuat sebuah *block ciphertext* yang unik (Stallings, 2010).

Algoritma kriptografi simetrik *Twofish* merupakan perkembangan dari algoritma *blowfish*. *Twofish* merupakan 128-bit *block cipher* yang menerima macam-macam ukuran kunci. Secara garis besar *twofish* dapat dijelaskan seperti pada gambar 2.2, 128-bit *plain text* atau file asli diberikan untuk menjadi input untuk proses *whitening*. Di mana file asli yang telah diproses *whitening*, akan di-

XOR-kan dengan empat kunci. Hasil dari XOR yaitu *output whitening* dienkripsi menggunakan fungsi g (Gehlot, 2013).



Gambar 2.3 Algoritma *Twofish* (Gehlot, 2013)

Block cipher yang digunakan pada *twofish* memiliki banyak mode operasi, tetapi yang digunakan pada penelitian ini adalah mode operasi *Cipher Block Chaining* (CBC). Mode operasi CBC pada penelitian digunakan untuk menghasilkan *Initialization Vector* (IV) yang unik untuk tiap proses enkripsi untuk menghasilkan *ciphertext* yang berbeda (Mao, 2003). IV yang unik pada setiap proses dibuat oleh *Pseudo-Random Number Generator* (PRNG) dan dapat disimpan di dalam tiap berkas file yang dienkripsi, agar dapat didekripsi.

2.5 *Computer-Assisted Instruction (CAI)*

Menurut Herman D Surjono (Surjono, 1999), istilah *Computer-Assisted Instruction* (CAI) umumnya menunjuk pada semua *software* pendidikan yang diakses melalui komputer di mana penggunanya dapat berinteraksi dengannya. Sistem komputer menyajikan serangkaian program pengajaran kepada pengguna baik berupa informasi maupun latihan soal-soal untuk mencapai tujuan pengajaran tertentu dan pengguna melakukan aktivitas belajar dengan cara berinteraksi dengan sistem komputer. Sehingga *Computer Assisted Instruction* (CAI) dapat dijelaskan sebagai sebuah konsep pengajaran suatu topik menggunakan perangkat aplikasi yang dirancang. Aplikasi tersebut dirancang untuk membuat materi dan metode pengajaran yang dapat dipelajari dan dipahami lebih baik. Pada penelitian ini penulis menggunakan konsep *Computer Assisted Instruction* (CAI) dalam perancangan aplikasi alat bantu pembelajaran tentang proses – proses dari metode – metode kriptografi SHA256, RSA, dan *twofish*.