Implementasi ECDSA untuk Verifikasi Berkas Berukuran Besar dengan Menggunakan Merkle Tree

Muhamad Visat Sutarno - 13513037
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
mvisat@gmail.com

***Abstrak* — Verifikasi berkas perlu dilakukan untuk mengecek integritasnya. Salah satu metode yang dapat digunakan untuk verifikasi berkas adalah perhitungan nilai *checksum* dengan fungsi *hash*, contohnya SHA-1. Akan tetapi, jika terjadi kesalahan saat transfer data, seluruh data pada berkas harus ditransfer ulang. Hal itu dapat diatasi dengan Merkle Tree sehingga partisi blok yang *corrupted* saja yang perlu ditransfer ulang. Masalah lainnya adalah dengan menggunakan *cheksum* kita tidak dapat menentukan siapakah pemilik asli berkas tersebut. Hal itu dapat diatasi dengan tanda tangan digital. Salah satu metode yang dapat digunakan untuk pembentukan tanda tangan digital adalah ECDSA. Makalah ini akan membahas implementasi ECDSA dan Merkle Tree untuk verifikasi berkas yang berukuran besar.**

***Kata Kunci* — ECDSA; Merkle Tree; SHA-1; Tanda Tangan Digital; Verifikasi**

I. PENDAHULUAN

Pada era informasi saat ini, pertukaran dan penyimpanan data dalam bentuk digital bukanlah hal yang asing lagi di kehidupan manusia. Hal ini dapat dibuktikan dengan informasi yang sangat mudah kita dapatkan. Selain kemudahan dalam mendapatkan informasi, kualitas dari konten informasi juga meningkat. Jika dulu video hanya berkualitas High Quality (480p), saat ini video dapat berkualitas 4K (2160p). Dengan semakin baiknya kualitas, semakin besar juga ukuran dari suatu berkas. Hal ini tidaklah menjadi masalah karena kapasitas penyimpanan data di berbagai media juga ikut semakin besar dan harganya lebih murah.

Akan tetapi, yang menjadi masalah adalah saat pemindahan data melalui suatu media. Sebagai contoh, jaringan komputer tidak dapat 100% diandalkan. Jika terjadi transfer data antar jaringan komputer, bisa jadi data hilang sehingga berkas menjadi *corrupted* dan tidak dapat dibuka. Selain itu, penyalinan data pada *flash drive* juga dapat menyebabkan suatu berkas menjadi *corrupted* karena berbagai faktor.

Untuk mengatasi hal ini, pada umumnya untuk menentukan apakah suatu berkas *corrupted* atau tidak dilakukan sebuah mekanisme pengecekan integritas atau verifikasi. Dalam melakukan verifikasi, sebuah berkas dilakukan perhitungan *message digest* yang disebut *checksum* dengan suatu fungsi *hash*. Pengunggah berkas di internet biasanya melakukan perhitungan *checksum* berkas yang ingin diunggah terlebih dahulu. Kemudian pengunggah berkas memberikan tautan menuju berkasnya dan mengumumkan hasil *checksum* tadi. Orang lain yang mengunduh berkas tersebut juga melakukan perhitungan *checksum* dengan fungsi *hash* yang sama. Jika *checksum*-nya tidak sama, maka telah terjadi kesalahan pada saat pertukaran data. Hal itu dapat terjadi baik pada sisi pengunggah maupun pengunduh.

Jika berkas tersebut berukuran kecil, mungkin hal tersebut tidak menjadi masalah. Jika kesalahan terjadi pada sisi pengunggah, berkas dapat diunggah ulang. Begitu juga jika kesalahan terjadi pada sisi pengunduh. Akan tetapi, berkas yang berukuran besar akan memakan waktu dan biaya yang lebih banyak.

Dibutuhkan suatu mekanisme agar tidak semua data pada berkas diunduh ulang. Dalam hal ini, Merkle Tree dapat digunakan untuk mengatasi hal tersebut. Merkle Tree dapat menyimpan nilai *checksum* untuk suatu partisi blok data, sehingga memungkinkan untuk mendapatkan partisi blok mana yang terjadi pengubahan data. Pengunggah atau pengunduh berkas tidak perlu melakukan transfer data ulang berkas tesebut, hanya perlu partisi blok yang *corrupted*.

Selain itu, jika hanya mengandalkan nilai *checksum*, identitas dari pemilik berkas asli tidak dapat diketahui. Hal ini dapat berbahaya jika ternyata berkas telah dimodifikasi pihak ketiga sedemikian rupa sehingga berkas menjadi virus/*malware*. Dalam hal ini, tanda tangan digital dapat digunakan. Tanda tangan digital dapat digunakan untuk memastikan apakah berkas benar dikirim oleh pemilik berkas asli dan tidak diubah oleh pihak ketiga. Salah satu tanda tangan digital yang dapat digunakan adalah ECDSA.

II. DASAR TEORI

A. Tanda Tangan Digital

Tanda tangan digital adalah suatu mekanisme matematis untuk menjaga otentikasi dari pesan atau dokumen. Tanda tangan digital merupakan nilai kriptografis yang bergantung

pada isi pesan dan kunci, sehingga tanda tangan digital selalu berbeda antara satu isi dokumen dengan dokumen lain. Tanda tangan digital memungkinkan penerima untuk memastikan bahwa pesan yang dikirimkan berasal dari pengirim yang bersangkutan, sehingga pengirim tidak bisa menyangkal telah mengirim pesan tersebut. Selain itu, tanda tangan digital juga memastikan pesan yang diterima tidak diubah sebelumnya. Tanda tangan digital dapat diimplementasikan dengan 2 cara, yaitu enkripsi pesan dan menggunakan kombinasi fungsi *hash*.

B. *Elliptic Curve Digital Signature Algorithm*

Elliptic Curve Digital Signature Algorithm (ECDSA) merupakan salah satu algoritma untuk membangkitkan tanda tangan digital yang termasuk dalam standar tanda tangan digital (*Digital Signature Standard* atau DSS). Setiap algoritma pada DSS terdiri dari dua komponen, yaitu algoritma tanda tangan digital (*Digital Signature Algorithm* atau DSA) dan fungsi *hash* standar (*Secure Hash Algorithm* atau SHA). Dalam ECDSA, DSA menggunakan algoritma *elliptic curve cryptography*, sementara fungsi *hash* menggunakan SHA-1.

Sebelum dilakukan penandatanganan, beberapa parameter kurva harus ditentukan, yaitu:

a. Persamaan kurva elips, $y^2 = x^3 + ax + b$, ditentukan a dan b
b. Bilangan prima p, domain pada medan berhingga F_p
c. Titik basis G
d. Orde prima n

Terdapat tiga proses dalam ECDSA:

a. Pembangkitan Kunci

Langkah-langkah untuk membangkitkan kunci privat dan publik adalah sebagai berikut:

1. Pilih sebuah bilangan bulat acak d, $1 \leq d \leq n-1$
2. Hitung $Q = d \,.\, G$

Kunci privatnya adalah d, dan kunci publiknya adalah Q. Setelah dibangkitkan, pengguna harus menyimpan baik-baik kunci privatnya. Sedangkan untuk kunci publik, siapapun boleh mengaksesnya.

b. Pembangkitan Tanda Tangan Digital

Untuk membangkitkan tanda tangan dari sebuah pesan M, langkah-langkahnya adalah sebagai berikut:

1. Pilih bilangan bulat acak k, $1 \leq k \leq n-1$
2. Hitung titik $(x, y) = k \,.\, G$
3. Hitung $r = x \bmod n$. Jika $r = 0$, kembali ke langkah 1.
4. Hitung $s = k^{-1}\,(hash(M) + d \,.\, r) \bmod n$. Jika $s = 0$, kembali ke langkah 1.

Tanda tangan digital dari pesan M adalah pasangan bilangan (r, s).

c. Verifikasi Tanda Tangan Digital

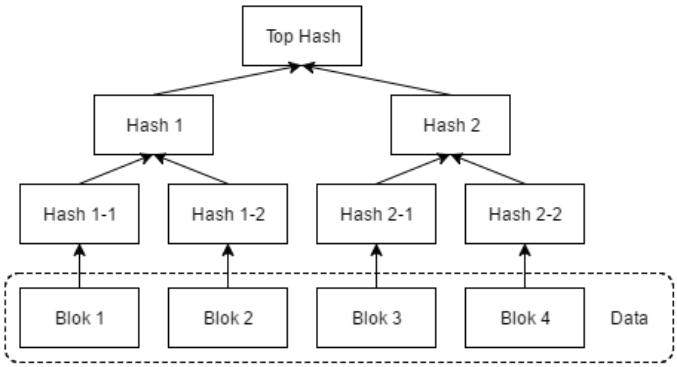
Untuk melakukan verifikasi tanda tangan digital dari pesan M, dibutuhkan kunci publik penanda tangan. Langkah-langkahnya adalah sebagai berikut:

1. Periksa apakah $1 \leq r, s \leq n-1$
2. Hitung $w = s^{-1} \bmod n$
3. Hitung $u = (w \,.\, hash(M)) \bmod n$
4. Hitung $v = (w \,.\, r) \bmod n$
5. Hitung titik $(x, y) = u \,.\, G + v \,.\, Q$
6. Hitung $x = x \bmod n$

Tanda tangan digital dan pesan tersebut dapat diverifikasi dengan valid jika nilai $x = r$.

C. Merkle Tree

Merkle Tree, atau biasa disebut juga *hash tree*, adalah sebuah struktur data pohon yang pada tiap simpulnya menyimpan hasil *hash* dari simpul-simpul anaknya. Misalkan terdapat data yang dipartisi menjadi beberapa blok. Setiap blok tersebut dihitung nilai *hash*-nya. Kemudian nilai *hash* tersebut dikonkatenasi dengan nilai *hash* sebelahnya dan membuat simpul dengan nilai *hash* dari hasil konkatenasi tadi di tingkatan berikutnya pada pohon. Hal tersebut dilakukan terus menerus hingga mencapai satu blok yang menjadi akar yang disebut *top hash*.



Gambar 1. Struktur Merkle Tree

Pada Merkle tree, data dapat diverifikasi secara parsial. Sebagai contoh, pada gambar 1 untuk melakukan verifikasi blok 1 dan blok 2, hanya diperlukan simpul hash 1, ataupun hash 1-1 dan hash 1-2.

Untuk melakukan verifikasi keseluruhan data, simpul pada pohon pun juga tidak perlu lengkap. Dengan hanya memiliki *top hash*, *hash* 1, dan *hash* 1-1, nilai dari setiap blok data dapat diverifikasi:

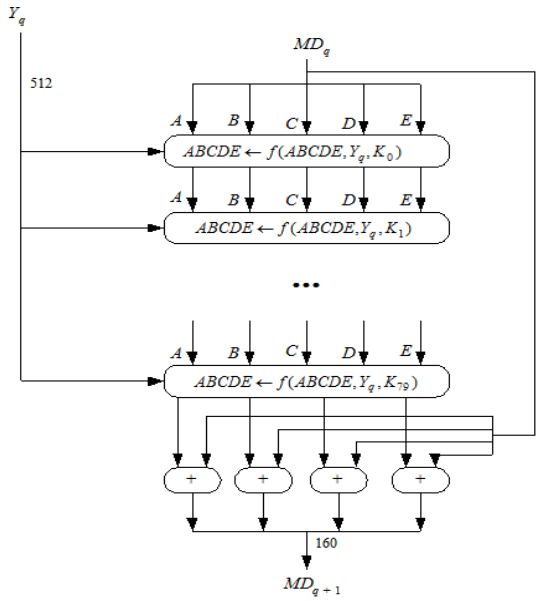
- Blok 2 dapat diverifikasi dengan membandingkan *hash* 1 dengan hasil konkatenasi *hash* 1-1 dan hasil *hash* blok 2.
- Blok 3 dan blok 4 dapat diverifikasi dengan membandingkan *top hash* dengan *hash* dari konkatenasi *hash* 1 dan konkatenasi hasil *hash* blok 3 dan blok 4.

D. *Secure Hash Algorithm*

SHA adalah fungsi *hash* satu arah yang dibuat oleh NIST dan digunakan bersama DSS (*Digital Signature Standard*). Oleh NSA, SHA dinyatakan sebagai standar fungsi *hash* satu arah. SHA didasarkan pada MD4 yang dibuat oleh Ronald L. Rivest dari MIT. Algoritma SHA menerima masukan berupa pesan dengan ukuran maksimum 2^{64} bit (2.147.483.648 gigabyte) dan menghasilkan message digest yang panjangnya 160 bit, lebih panjang dari *message digest* yang dihasilkan oleh MD5. Salah satu varian dari SHA adalah SHA-1.

Langkah-langkah pembuatan *message digest* dengan SHA-1 adalah sebagai berikut:

1. Penambahan bit-bit pengganjal (*padding bits*)
2. Penambahan nilai panjang pesan semula
3. Inisialisasi penyangga (*buffer*) MD
4. Pengolahan pesan dalam blok berukuran 512 bit



Gambar 2. Skema pembuatan *message digest* pada SHA-1

III. Desain Solusi dan Implementasi

A. EDCSA sebagai Pengganti Fungsi *Hash*

Pada Merkle Tree, simpul pada pohon biasanya bernilai *message digest* yang dihasilkan suatu fungsi *hash.* Pada makalah ini, digunakan ECDSA untuk menggantikan fungsi *hash* tadi, dan *message digest* digantikan dengan tanda tangan digital.

B. Parameter ECDSA

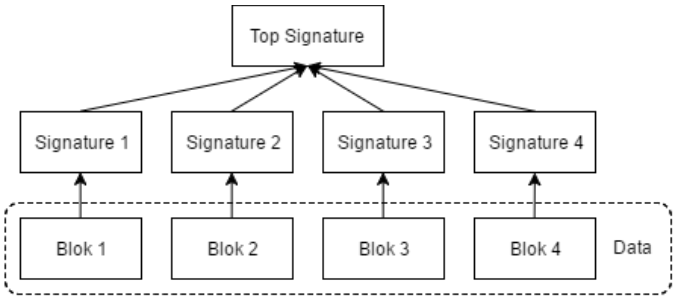
Masalah lain yang muncul saat menggunakan ECDSA adalah parameter-parameter yang digunakan. Perubahan sedikit saja pada parameter dapat menghasilkan tanda tangan digital yang jauh berbeda. Pada makalah ini, parameter untuk ECDSA mengikuti standar NIST-P256.

Parameter	Nilai
p	FFFFFFFF00000001000000000000000000000000FFFFFFFFFFFFFFFFFFFFFFFF
a	FFFFFFFF00000001000000000000000000000000FFFFFFFFFFFFFFFFFFFFFFFC
b	5AC635D8AA3A93E7B3EBBD55769886BC651D06B0CC53B0F63BCE3C3E27D2604B
Gx	6B17D1F2E12C4247F8BCE6E563A440F277037D812DEB33A0F4A13945D898C296
G_y	4FE342E2FE1A7F9B8EE7EB4A7C0F9E162BCE33576B315ECECBB6406837BF51F5
n	FFFFFFFF00000000FFFFFFFFFFFFFFFFBCE6FAADA7179E84F3B9CAC2FC632551

Tabel 1. Paramater kurva

C. Implementasi Merkle Tree

Variasi dari Merkle Tree yang penulis ajukan dalam makalah ini mempunyai dua tingkat, yaitu akar dan daun. Simpul akar mempunyai empat cabang yang merupakan daun. Akan tetapi, jumlah tingkat dan cabang yang dapat digunakan tidak tertutup pada angka tersebut, hanya saja tidak tercakup dalam makalah ini.



Gambar 3. Implementasi Merkle Tree

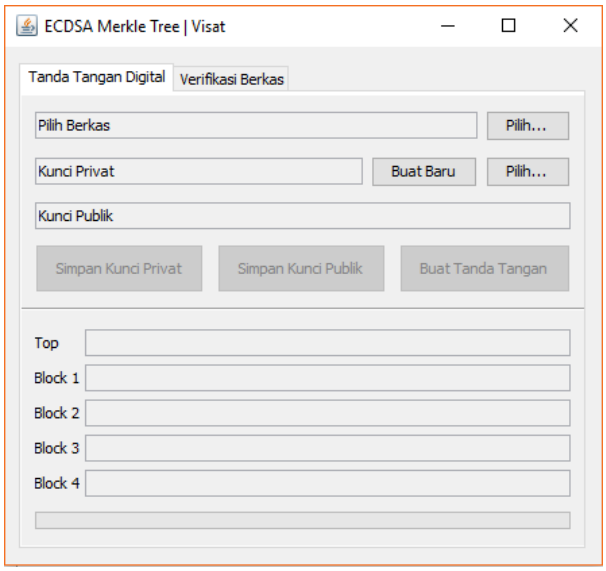
Pada gambar 3, pertama-tama data dipartisi menjadi empat blok. Kemudian setiap blok data dihitung nilai *hash*-nya dengan fungsi *hash* SHA-1. Dari nilai hash itu lalu dihitung nilai tanda tangan digital sebagai *signature* 1, *signature* 2, dan seterusnya. Tanda tangan digital tersebut menjadi daun pada Merkle Tree. Setelah itu, setiap nilai *hash* dari blok data yang telah dihitung dilakukan konkatenasi menjadi suatu *string* panjang. *String* tersebut lalu dihitung nilai tanda tangan digitalnya dan menjadi *top signature* yang merupakan akar pada Merkle Tree.

D. Implementasi Program

Pada makalah ini digunakan bahasa pemrograman Java untuk mengimplementasikan desain solusi yang telah diajukan. Selain itu digunakan juga Java Swing untuk antarmuka. Terdapat dua tab utama yang pengguna dapat pilih.

. Tab pertama digunakan untuk membuat tanda tangan digital. Langkah-langkah untuk membuat tanda tangan digital adalah sebagai berikut:

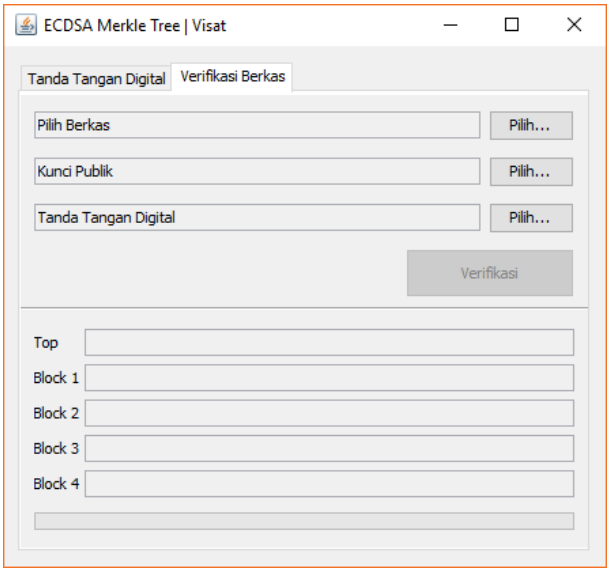
1. Pengguna memilih berkas yang ingin dibuat tanda tangan digitalnya
2. Pengguna dapat membuat kunci privat baru ataupun memilih berkas yang berisi kunci privat
3. Kunci publik akan secara otomatis dibangkitkan dari kunci privat tadi
4. Pengguna dapat menyimpan kunci privat dan kunci publik ke suatu berkas terpisah
5. Pengguna dapat membuat tanda tangan digital dari masukan tadi ke suatu berkas
6. *Progress* pembuatan tanda tangan digital juga ditampilkan ke pengguna
7. Setelah selesai, hasil perhitungan tanda tangan digital akan ditampilkan



Gambar 4. Tab tanda tangan digital

Tab kedua digunakan untuk melakukan verifikasi berkas. Langkah-langkah untuk melakukan verifikasi berkas adalah sebagai berikut:

1. Pengguna dapat memilih berkas yang ingin diverikasi
2. Kunci publik dapat dipilih dari suatu berkas oleh pengguna
3. Pengguna juga dapat memilih tanda tangan digital yang telah dibuat
4. Saat verifikasi dilakukan, pengguna dapat melihat *progress*-nya
5. Setelah selesai, hasil verifikasi setiap simpul pada pohon Merkle Tree akan ditampilkan



Gambar 5. Tab verifikasi berkas

IV. Pengujian dan Analisis

Untuk pengujian, digunakan suatu kunci privat dan kunci publik.

Kunci		Nilai
Kunci Privat (d)		900814425587998056530358464758114989994606956330879443854792690475951702103 65
Kunci Publik	Q_x	316958104813383691919072084829585776893710965918345070197067914807752646580 37
	Q_y	837349372442548334825450025095045390780769046089615489775650317317775379843 76

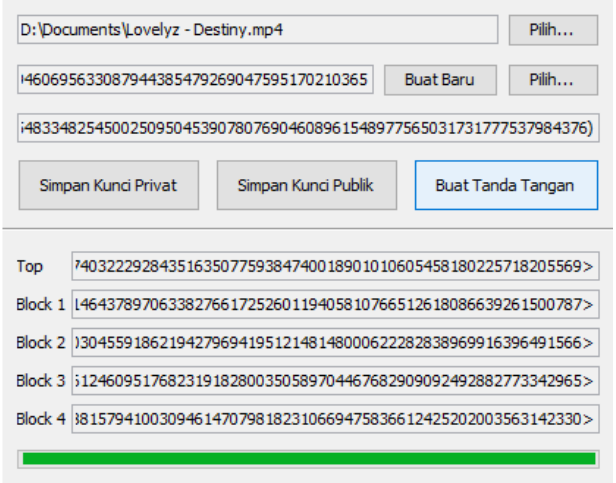
Tabel 2. Kunci privat dan publik

Berkas untuk pengujian adalah suatu berkas video.

Jenis	Nilai
Nama	Lovelyz – Destiny.mp4
Ukuran	450.478.142 bytes (429 MB)

Tabel 3. Berkas pengujian

Dengan kunci dan berkas tersebut dilakukan pembangkitan tanda tangan digital. Hasil tersebut disimpan ke dalam berkas bernama “digital.signature”.



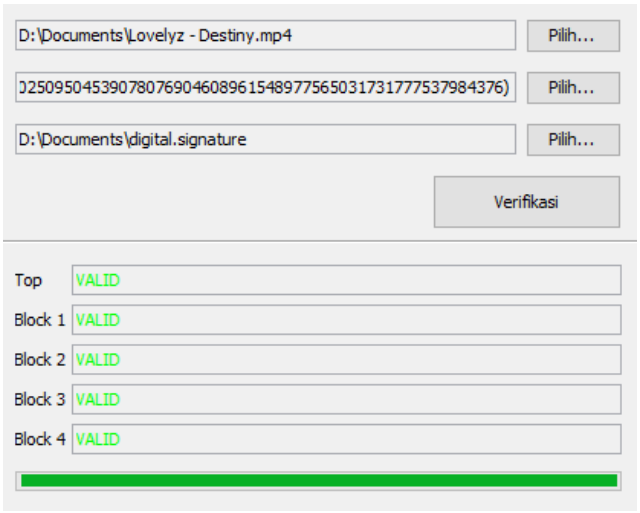
Gambar 6. Hasil tanda tangan digital berkas pengujian

Kasus-kasus yang akan diuji adalah sebagai berikut:

1. Berkas yang tidak diubah dan kunci publik yang valid
2. Berkas dengan blok 1 diubah dan kunci publik yang valid
3. Berkas dengan blok 1 dan 2 diubah dan kunci publik yang valid
4. Berkas yang tidak diubah dan kunci publik yang tidak valid
5. Berkas dengan blok 1 diubah dan kunci publik yang tidak valid

A. Berkas yang Tidak Diubah dan Kunci Publik yang Valid

Pengujian ini adalah *positive test case scenario*, dengan program seharusnya berjalan sebagaimana mustinya.

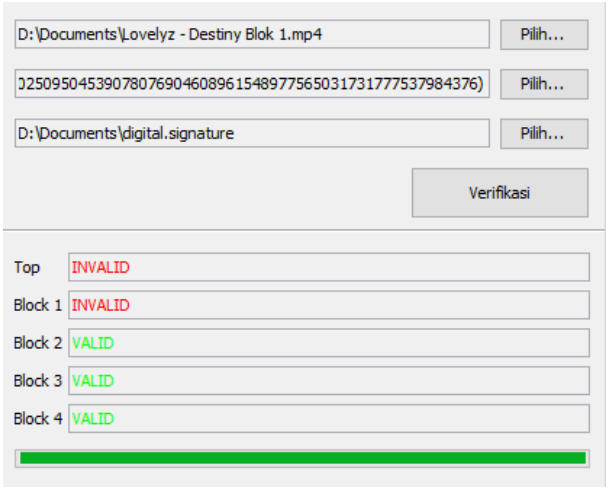


Gambar 7. Hasil pengujian A

Program berhasil memverifikasi berkas, dengan semua simpul pada Merkle Tree bernilai valid.

B. Berkas dengan Blok 1 Diubah dan Kunci Publik yang Valid

Pada kasus ini, penulis membuat program kecil untuk mengubah data pada blok 1, sehingga seharusnya berkas menjadi tidak valid pada blok tersebut.

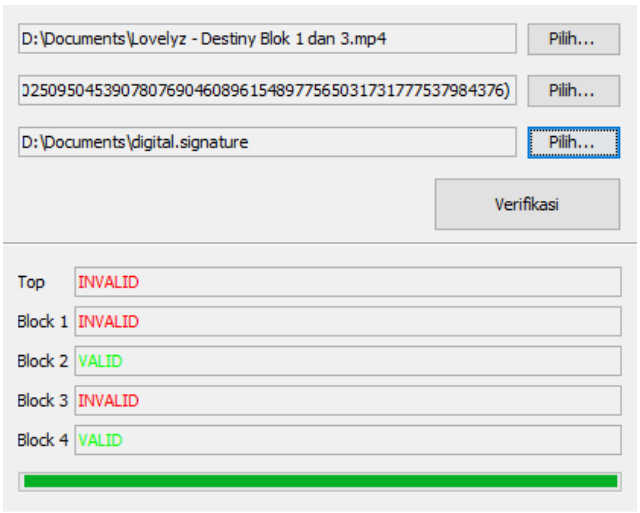


Gambar 8. Hasil pengujian B

Program tidak berhasil memverifikasi data blok 1 dan *top signature*. Hal ini sesuai dengan hasil yang diharapkan karena nilai *hash* dari blok 1 telah berubah sehingga *top signature* juga ikut berubah.

C. Berkas dengan Blok 1 dan 3 Diubah dan Kunci Publik yang Valid

Kasus ini mirip dengan kasus B. Akan tetapi, data pada blok 2 juga diubah.

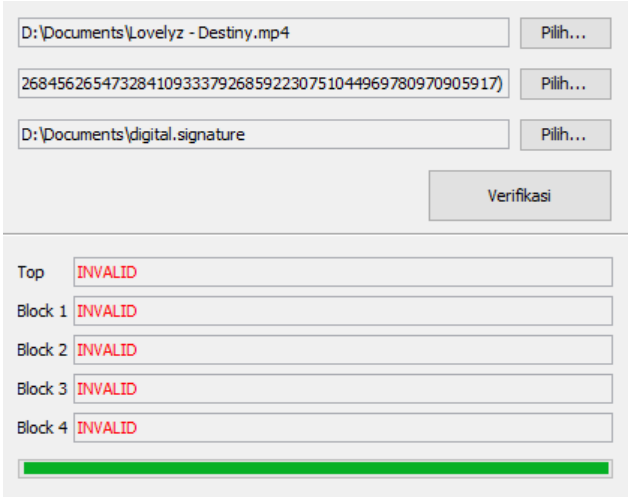


Gambar 9. Hasil pengujian C

Selain blok 1, blok 3 juga tidak berhasil diverifikasi. Dapat dilihat jika salah satu blok tidak berhasil diverifikasi, *top signature* juga akan tidak bernilai valid.

D. Berkas yang Tidak Diubah dan Kunci Publik yang Tidak Valid

Pengujian ini melakukan verifikasi dengan membuat kunci publik baru yang berbeda dengan aslinya. Data pada berkas tidak diubah.

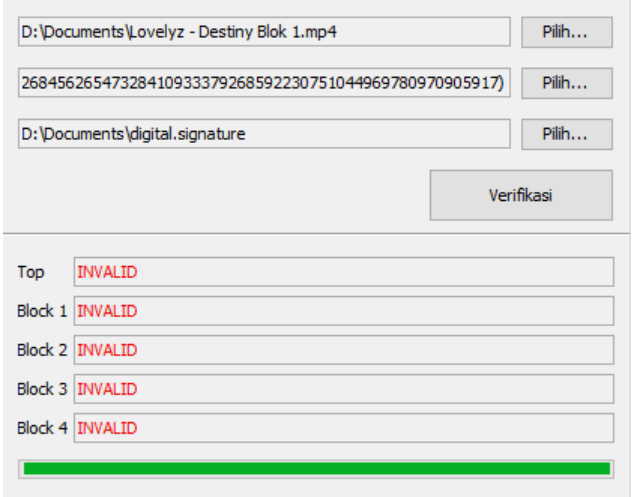


Gambar 10. Hasil pengujian D

Program tidak dapat melakukan verifikasi pada semua simpul. Hal ini sesuai dengan yang diharapkan karena digunakan kunci publik yang berbeda.

E. Berkas dengan Blok 1 Diubah dan Kunci Publik yang Tidak Valid

Pengujian ini selain memakai kunci publik yang berbeda, data pada blok 1 juga diubah.



Gambar 11. Hasil pengujian E

Program juga tidak dapat melakukan verifikasi pada semua simpul, karena kunci publik yang digunakan sudah berbeda.

V. KESIMPULAN DAN SARAN

Berdasarkan hasil implementasi dan pengujian, ECDSA dan Merkle Tree dapat digunakan untuk mengecek integritas berkas dan melakukan otentikasi pengirim. Dengan ECDSA, pihak ketiga tidak dapat mengubah isi file, dan pemilik berkas asli dapat diketahui. Dengan Merkle Tree, partisi blok yang corrupted dapat diketahui sehingga tidak semua data perlu dikirim ulang.

Hal lain yang dapat ditambahkan adalah dapat dilakukan pemrosesan secara paralel. Program pada makalah ini memroses berkas secara sekuensial, sehingga blok 2 akan diproses setelah blok 1, dan seterusnya. Jika tiap blok diproses secara bersamaan, proses verifikasi akan jauh lebih cepat. Selain itu, pada makalah ini jumlah tingkatan pada Merkle Tree adalah dua dan partisi blok adalah empat. Jumlah tersebut dapat dioptimasi menjadi *binary tree* sehingga kinerja dapat lebih baik lagi.

REFERENSI

[1] Rinaldi M., Bahan Kuliah IF4020 Kriptografi: Tanda Tangan Digital

[2] Rinaldi M., Bahan Kuliah IF4020 Kriptografi: Elliptic Curve Cryptography

[3] https://brilliant.org/wiki/merkle-tree/ (diakses pada 19 Desember 2016)

[4] http://developer.amd.com/community/blog/2015/05/29/merkle-tree-hashing-using-opencl-2-0/ (diakses pada 19 Desember 2016)

[5] Mehmet Adalier, "Efficient and Secure Elliptic Curve Cryptography Implementation of Curve P-256", 2015.

[6] https://dxr.mozilla.org/mozilla-central/source/security/nss/lib/freebl/ecl/ecl-curve.h (diakses pada 18 Desember 2016)

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Desember 2016

Muhamad Visat Sutarno
NIM 13513037