Performa Elliptic Curve Digital Signature Algorithm (ECDSA) dengan Variasi Fungsi Hash dan Panjang Pesan

Nixon Andhika 13517059[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13517059@std.stei.itb.ac.id

***Abstrak*—Tanda tangan digital digunakan untuk mengotentikasi dokumen digital. Salah satu algoritma tanda tangan digital yang umum digunakan adalah ECDSA. Akan tetapi, fungsi hash SHA-1 yang digunakan pada ECDSA tidak aman karena rentan terhadap collision attack. Fungsi hash yang digunakan dapat diganti dengan fungsi hash lain untuk meningkatkan performa dari ECDSA. Makalah ini membahas perbedaan performa ECDSA melalui penggunaan berbagai fungsi hash dan panjang pesan.**

***Kata Kunci*—ECDSA, fungsi hash, kecepatan, panjang pesan.**

I. PENDAHULUAN

Tanda tangan merupakan sebuah tanda atau bukti otentik dari suatu objek. Tanda tangan telah digunakan sejak zaman dahulu untuk mengotentikasi dokumen cetak. Pada zaman modern dengan meningkatnya dokumen digital, otentikasi dilakukan dengan tanda tangan digital. Tanda tangan digital diartikan sebagai teknik matematis yang digunakan untuk memvalidasi keaslian dan integritas dari sebuah pesan, perangkat lunak, atau dokumen digital. Nilai tanda tangan digital berasal dari isi dokumen yang ditandatangani sehingga segala bentuk perubahan pada isi dokumen akan menyebabkan verifikasi tanda tangan menjadi gagal [1].

Salah satu algoritma tanda tangan digital yang umum digunakan adalah *Elliptic Curve Digital Signature Algorithm* (ECDSA). ECDSA sering digunakan karena mampu memberi tingkat keamanan yang sama dengan algoritma tanda tangan digital lainnya dengan panjang kunci yang jauh lebih kecil. Untuk membentuk tanda tangan digital, ECDSA melakukan enkripsi terhadap nilai hasil *hash* yang disebut dengan *message digest*. *Message digest* merupakan nilai *hash* isi pesan atau dokumen yang didapatkan dari hasil operasi fungsi *hash*. Terdapat berbagai macam fungsi *hash* dengan tingkat keamanan dan kecepatannya masing-masing. Pada implementasi ECDSA, digunakan *Secure Hash Algorithm* 1 (SHA-1) sebagai fungsi *hash*. Akan tetapi, SHA-1 tidak aman digunakan lagi karena rentan terhadap *collision attack*. Oleh karena itu, ECDSA umumnya menggunakan fungsi *hash* lain untuk menghasilkan *message digest* yang dibutuhkan. Penggunaan fungsi *hash* yang berbeda akan memberikan efek yang berbeda terhadap ECDSA secara keseluruhan, baik efek terhadap keamanan dan kecepatan. Pada makalah ini akan dibahas perbedaan performa dari ECDSA melalui penggunaan berbagai fungsi *hash* yang berbeda serta panjang pesan dari dokumen yang ditandatangani.

II. DASAR TEORI

A. Tanda Tangan Digital

Tanda tangan digital menggunakan kriptografi asimetri yang akan membentuk kunci privat untuk mengenkripsi hasil *hash* data pesan atau *message digest* menjadi tanda tangan dan kunci publik untuk memverifikasi keaslian dari pesan. *Hashing* dilakukan karena *message digest* yang dihasilkan bersifat unik untuk setiap pesan masukan. Perubahan terhadap pesan, walaupun hanya satu karakter, akan menghasilkan *message digest* yang sangat berbeda. Sifat ini memungkinkan pengguna untuk memvalidasi integrasi dari data dengan cara membandingkan nilai *hash* pesan dengan nilai *hash* hasil dekripsi tanda tangan. Jika kedua nilai *hash* tidak sama, maka pesan dari dokumen kemungkinan telah dimodifikasi atau tanda tangan diciptakan dengan kunci privat yang tidak sesuai dengan kunci publiknya [1].

B. ECDSA

ECDSA merupakan algoritma tanda tangan digital yang menggunakan komputasi berbasis kurva eliptik. ECDSA tidak memiliki algoritma yang bersifat *subexponential-time* sehingga kekuatan per bit kunci lebih besar daripada algoritma tanda tangan digital yang tidak berbasis kurva eliptik. Karena kekuatan per bit kunci yang lebih tinggi, panjang kunci yang diperlukan lebih kecil untuk memberi tingkat keamanan yang sama dengan algoritma lainnya. Sebagai contoh, kunci 160-bit ECDSA memberikan tingkat keamanan yang sebanding dengan kunci 1024-bit algoritma *Rivest-Shamir-Adleman* (RSA) [2].

ECDSA, sebagai salah satu algoritma tanda tangan digital, melakukan enkripsi *message digest* yang dihasilkan fungsi *hash* dengan sebuah kunci privat. Implementasi awal ECDSA menggunakan fungsi *hash* SHA-1 yang akan membentuk *message digest* sepanjang 160-bit. Akan tetapi, penggunaan

SHA-1 di ECDSA kurang baik karena ditemukannya kelemahan terhadap *collision attack* [3].

C. Fungsi Hash

Fungsi *hash* merupakan fungsi yang melakukan kompresi terhadap pesan dengan panjang sembarang menjadi sebuah *string* dengan panjang tertentu. Luaran dari fungsi *hash* disebut dengan *message digest* atau nilai *hash* (*hash value*). Fungsi *hash* merupakan fungsi satu arah sehingga bersifat *irreversible*. Hasil akhir dari fungsi *hash* tidak dapat dikembalikan menjadi masukan semula [4].

Fungsi *hash* dapat dirumuskan menjadi bentuk berikut.

$$h = H(M)$$

dengan *h* merupakan *message digest*, *H* merupakan fungsi *hash*, *M* merupakan pesan semula dengan panjang sembarang.

Terdapat tiga properti yang diperlukan oleh sebuah fungsi *hash*, yaitu *collision resistance*, *preimage resistance*, *second preimage resistance*. *Collision resistance* mengacu kepada sulitnya ditemukan dua masukan berbeda yang menghasilkan keluaran yang sama. *Preimage resistance* mengacu kepada sulitnya ditemukan masukan dari suatu keluaran. *Second preimage resistance* mengacu kepada sulitnya ditemukan masukan kedua yang menghasilkan keluaran yang sama dengan masukan pertama [5]. Sebuah fungsi *hash* dinyatakan aman apabila ketiga properti tersebut terpenuhi.

Sejak tahun 1989, telah dikembangkan berbagai fungsi *hash*. Beberapa dari fungsi *hash* tersebut adalah sebagai berikut.

1. *Message Digest* (MD)
   MD merupakan sebuah seri fungsi *hash* yang dikembangkan oleh Ronald Rivest. Terdapat empat fungsi *hash* pada seri ini, yaitu MD2, MD4 [6], MD5 [7], dan MD6 [8].
   a. MD2
      MD2 merupakan fungsi *hash* pertama dari seri MD yang dikembangkan oleh Ronald Rivest pada tahun 1989. Fungsi ini dioptimisasi untuk komputer 8-bit. MD2 menghasilkan *message digest* sepanjang 128-bit. MD2 rentan terhadap *preimage attack* dan *collision attack* sehingga digantikan oleh MD4 [5].

   b. MD4
      MD4 dikembangkan oleh Ronald Rivest pada tahun 1990 untuk menggantikan MD2. MD4 menghasilkan *message digest* sepanjang 128-bit. Jika diinginkan *message digest* sepanjang 256-bit, dijalankan dua salinan MD4 secara paralel, dilakukan beberapa substitusi tambahan, dan dilakukan penyambungan kedua *digest* 128-bit menjadi 256-bit. MD4 didesain agar dapat berjalan cepat pada mesin 32-bit, simpel, dan tidak memerlukan tabel substitusi yang besar [6]. MD4 rentan terhadap *preimage attack* dan *collision attack* sehingga digantikan oleh MD5.

   c. MD5
      MD5 dikembangkan oleh Ronald Rivest pada tahun 1992 untuk menggantikan MD4. MD5 menggunakan struktur Merkle-Damgard dan menghasilkan *message digest* sepanjang 128-bit. MD5 merupakan ekstensi dari MD4 dengan fokus kepada peningkatan keamanan dengan *tradeoff* kecepatan. Ekstensi tersebut berupa penambahan ronde dan operasi, serta perubahan isi fungsi [7]. Pada tahun 2013, ditemukan bahwa MD5 rentan terhadap kolisi.

   d. MD6
      MD6 merupakan fungsi *hash* terakhir dari seri MD yang dikembangkan pada tahun 2008. Proposal MD6 diajukan untuk menjadi algoritma SHA-3 baru. MD6 dapat menghasilkan *message digest* sepanjang 224, 256, 384, dan 512 *bits*. MD6 menggunakan struktur *Merkle Tree* untuk memungkinkan paralelisasi masukan yang sangat panjang dan waktu pemrosesan yang lebih cepat [8].

2. *Secure Hash Algorithm* (SHA)
   SHA merupakan sebuah grup fungsi *hash* yang dipublikasikan oleh *National Institute of Standards and Technology* (NIST). Terdapat empat fungsi *hash*, yaitu SHA-0, SHA-1, SHA-2, SHA-3.
   a. SHA-0
      SHA-0 merupakan fungsi *hash* pertama dari grup SHA yang dikembangkan oleh NIST pada tahun 1993. SHA-0 menghasilkan *message digest* sepanjang 160-bit. Akibat kelemahan signifikan yang ditemukan, publikasi SHA-0 ditarik kembali dan digantikan oleh SHA-1.

   b. SHA-1
      SHA-1 dikembangkan oleh NIST pada tahun 1995 untuk menggantikan SHA-0. Sama seperti MD5, SHA-1 menggunakan struktur Merkle-Damgard. SHA-1 menghasilkan *message digest* sepanjang 160-bit. SHA-1 rentan terhadap *collision attack* dan *chosen-prefix attack* sehingga tidak digunakan lagi untuk tanda tangan digital [3].

   c. SHA-2
      SHA-2 dikembangkan pada tahun 1995 untuk menggantikan SHA-1. SHA-2 menghasilkan keluaran sepanjang 224, 256, 384, dan 512 *bits*. SHA-2 menggunakan struktur yang sama dengan MD5 dan SHA-1, tetapi lebih kompleks dengan penambahan

fungsi non linear ke fungsi kompresi. Hal ini membuat SHA-2 lebih pelan, tetapi lebih aman, daripada SHA-1. Belum ditemukan cara untuk sepenuhnya merusak *resistance* SHA-2 sehingga masih aman digunakan [3].

d. SHA-3

SHA-3 merupakan fungsi *hash* terakhir dari grup SHA. Berbeda dari SHA sebelumnya, SHA-3 tidak dikembangkan oleh NIST dan merupakan hasil kompetisi yang diciptakan oleh NIST. SHA-3 merupakan bagian dari grup fungsi *hash* Keccak. Struktur SHA-3 berbeda dari SHA sebelumnya, dengan SHA-3 menggunakan struktur *sponge* dibandingkan struktur Merkle-Damgard yang sebelumnya digunakan. SHA-3 menghasilkan *message digest* sepanjang 224, 256, 384, dan 512 *bits* [9]. Sama seperti SHA-2, SHA-3 juga masih aman digunakan.

3. *Race Integrity Primitives Evaluation Message Digest* (RIPEMD)

RIPEMD merupakan sebuah grup fungsi *hash* yang dikembangkan pada tahun 1992 oleh Hans Dobbertin, Antoon Bosselaers, dan Bart Preneel. RIPEMD diciptakan berdasarkan fungsi *hash* MD4. Terdapat lima varian dari RIPEMD, yaitu RIPEMD, RIPEMD-128, RIPEMD-160, RIPEMD-256, dan RIPEMD-320. Fungsi *hash* yang umum digunakan adalah RIPEMD-160, sedangkan varian lain jarang digunakan. Hal ini dikarenakan ditemukan kelemahan pada RIPEMD dan RIPEMD-128, sedangkan RIPEMD-256 dan RIPEMD-320 memberikan tingkat keamanan yang masing-masing sama dengan RIPEMD-128 dan RIPEMD-160. RIPEMD-160, yang merupakan perbaikan dari RIPEMD, melakukan perubahan terhadap jumlah ronde dan operasi dalam ronde, mengubah urutan blok pesan, dan mengubah urutan fungsi *boolean* untuk meningkatkan keamanan dengan *tradeoff* kecepatan [10].

4. GOST

GOST merupakan fungsi-fungsi *hash* yang terdefinisi di standar nasional Russia. Terdapat dua fungsi *hash*, yaitu GOST R 34.11-94 dan GOST R 34.11-2012 / Streebog.

a. GOST R 34.11-94

GOST R 34.11-94 diciptakan pada tahun 1994. Fungsi *hash* ini menghasilkan *message digest* sepanjang 256-bit. GOST R 34.11-94 rentan terhadap *collision attack* dan *preimage attack* sehingga digantikan oleh Streebog.

b. GOST R 34.11-2012 / Streebog

Streebog diciptakan untuk menggantikan GOST R 34.11-94 pada tahun 2012. Streebog menghasilkan *message digest* sepanjang 256 dan 512 *bits*. Streebog menggunakan struktur Merkle-Damgard seperti pada SHA-2 dengan 12 ronde fungsi kompresi seperti pada *Advanced Encryption Standard* (AES).

5. Whirlpool

Whirlpool diciptakan oleh Vincent Rijmen dan Paulo S. L. M. Barreto pada tahun 2000. Whirlpool didesain berdasarkan *square block cipher* dan menggunakan struktur Miyaguchi-Preneel. Whirlpool menerima masukan pesan dengan panjang kurang dari 2^{256} serta kunci 512-bit dan menghasilkan *message digest* sepanjang 256-bit [11]. Tidak terdapat kelemahan yang diketahui pada Whirlpool.

6. BLAKE

BLAKE merupakan fungsi *hash* yang diciptakan berdasarkan ChaCha *stream cipher*. Terdapat tiga fungsi *hash* BLAKE, yaitu BLAKE, BLAKE2, dan BLAKE3 [12].

a. BLAKE

BLAKE merupakan salah satu dari fungsi *hash* yang diikutsertakan pada kompetisi SHA-3 dan mencapai ronde terakhir, tetapi kalah kepada Keccak. Terdapat empat varian dari BLAKE, yaitu BLAKE-224 dan BLAKE-256 yang menggunakan blok 32-bit, serta BLAKE-384 dan BLAKE-512 yang menggunakan blok 64-bit.

b. BLAKE2

BLAKE2 merupakan perkembangan lebih lanjut dari BLAKE. BLAKE2 diklaim lebih cepat dibandingkan MD5, SHA-1, SHA-2, dan SHA-3 dengan tingkat keamanan sebanding dengan SHA-3. Terdapat dua varian BLAKE2, yaitu BLAKE2b dan BLAKE2s. BLAKE2b dioptimisasi untuk mesin 64-bit dan menghasilkan *message digest* dengan panjang antara 1-64 *bytes* (8-512 *bits*) sedangkan BLAKE2s dioptimisasi untuk mesin 8-32 bit dan menghasilkan *message digest* dengan panjang antara 1-32 *bytes* (8-256 *bits*).

c. BLAKE3

BLAKE3 merupakan perkembangan lebih lanjut lagi dari BLAKE2. BLAKE3 diciptakan menggunakan struktur pohon Merkle untuk memungkinkan tingkat paralelisasi yang tinggi dan mempercepat proses *hashing*. BLAKE menghasilkan *message digest* sepanjang 256-bit yang bisa di ekstensi.

III. PENGUJIAN

Pengujian dilakukan dengan menjalankan algoritma tanda tangan digital ECDSA dengan berbagai fungsi *hash* dan tiga panjang pesan yang berbeda. Algoritma ECDSA yang digunakan dapat ditemukan pada *repository* berikut https://github.com/starkbank/ecdsa-python

A. Spesifikasi Sistem

Pengujian dilakukan dengan menggunakan algoritma ECDSA dalam bahasa Python. Berikut spesifikasi sistem yang digunakan untuk menjalankan pengujian.

Tabel 1. Spesifikasi sistem pengujian

Sistem Operasi	Windows 10 Home Version 20H2 OS Build 19042.685
Processor	Intel® Core™ i7-7700HQ CPU @ 2.80 GHz
Graphic Card	Nvidia GTX 1050 4GB
Memori	8192MB
Arsitektur	64-bit
Versi Python	3.8.6

B. Fungsi Hash

Terdapat 20 fungsi *hash* yang diuji. Dari 20 fungsi tersebut, terdapat beberapa fungsi *hash* lama yang sudah tidak digunakan karena tidak aman. Fungsi-fungsi tersebut digunakan sebagai pembanding kecepatan. Berikut tabel fungsi *hash* beserta *library* dari fungsi yang digunakan.

Tabel 2. Daftar fungsi *hash* beserta *library* dari fungsi yang digunakan.

Fungsi *hash*	*Library*
BLAKE2b	hashlib
BLAKE2s	hashlib
BLAKE3	hashlib
GOST	pygost
Streebog-256	pygost
Streebog-512	pygost
RIPEMD-160	pycryptodome
MD2	pycryptodome
MD4	pycryptodome
MD5	hashlib
SHA-1	hashlib
SHA-224	hashlib
SHA-256	hashlib
SHA-384	hashlib
SHA-512	hashlib
SHA3-224	hashlib
SHA3-256	hashlib
SHA3-384	hashlib
SHA3-512	hashlib
Whirlpool	whirlpool

C. Panjang Pesan

Isi pesan merupakan sebuah *string* berisi huruf-huruf acak yang dibentuk hingga panjangnya mencapai nilai tertentu. Digunakan tiga variasi panjang pesan, yaitu pesan sepanjang 1.000.000 (satu juta) huruf, 10.000.000 (sepuluh juta) huruf, dan 100.000.000 (seratus juta) huruf. Ukuran panjang pesan yang digunakan cukup besar agar terlihat perbedaan signifikan kecepatan fungsi *hash* dalam membentuk *message digest* seiring dengan peningkatan ukuran pesan. Berikut algoritma pembangkitan pesan acak.

```
1. import string
2. import random
3.
4. N = 1000000
5. message =
   ''.join(random.choices(string.ascii_upperc
   ase + string.digits, k=N))
```

D. Langkah Pengujian

Berikut langkah-langkah yang digunakan dalam pengujian.

1. Pembentukan *string* pesan acak dengan panjang N ($N \in \{1000000, 10000000, 100000000\}$).
2. Pembentukan kunci privat dan kunci publik.
3. Inisialisasi *list* waktu tanda tangan dan *list* waktu verifikasi tanda tangan.
4. Dilakukan operasi tanda tangan dan verifikasi tanda tangan sebanyak 10 kali untuk setiap fungsi *hash* yang diuji. Khusus untuk fungsi *hash* GOST, Streebog-256, dan Streebog-512 hanya dilakukan 1 kali karena waktunya yang lama.
5. Dilakukan pencatatan waktu tanda tangan dan waktu verifikasi tanda tangan pada kedua *list* untuk setiap operasi tanda tangan dan verifikasi.
6. Dilakukan perhitungan rata-rata waktu tanda tangan dan rata-rata waktu verifikasi tanda tangan.

IV. HASIL PENGUJIAN DAN ANALISIS

A. Hasil Pengujian

Berikut hasil pengujian kecepatan ECDSA dengan fungsi-fungsi *hash* berbeda dan variasi panjang pesan.

1. Panjang pesan = 1 juta huruf.
   Berikut kecepatan ECDSA dengan fungsi *hash* berbeda untuk panjang pesan 1 juta huruf.

Tabel 3. Perbandingan kecepatan ECDSA dengan variasi fungsi *hash* untuk panjang pesan 1 juta huruf

Fungsi *hash*	*Sign Time* (s)	*Verify Time* (s)
BLAKE2b	0,0053329	0,00836926
BLAKE2s	0,00658971	0,00961143
BLAKE3	0,00463274	0,00719632
GOST	18,5474383	18,4699646
Streebog-256	36,0595446	35,2456898
Streebog-512	35,8441486	40,2053664
RIPEMD-160	0,00889304	0,01165833
MD2	0,16777821	0,1710474

MD4	0,00509567	0,00816377
MD5	0,00576936	0,00904521
SHA-1	0,00542075	0,00887724
SHA-224	0,00635222	0,00887326
SHA-256	0,00575686	0,00892595
SHA-384	0,00520024	0,00839653
SHA-512	0,00504948	0,00822952
SHA3-224	0,00632002	0,00910617
SHA3-256	0,00690568	0,01009145
SHA3-384	0,00837596	0,01148703
SHA3-512	0,00920337	0,01252532
Whirlpool	0,01485886	0,0176474

2. Panjang pesan = 10 juta huruf.
Berikut kecepatan ECDSA dengan fungsi *hash* berbeda untuk panjang pesan 10 juta huruf.

Tabel 4. Perbandingan kecepatan ECDSA dengan variasi fungsi *hash* untuk panjang pesan 10 juta huruf

Fungsi *hash*	S*ign Time* (s)	*Verify Time* (s)
BLAKE2b	0,02913748	0,03296978
BLAKE2s	0,03887057	0,04176037
BLAKE3	0,01032141	0,01332075
GOST	187,4801133	189,3502201
Streebog-256	352,5972154	327,9291672
Streebog-512	327,6165079	331,3550041
RIPEMD-160	0,060926100	0,06459049
MD2	1,63371819	1,64340853
MD4	0,02747357	0,03039645
MD5	0,02509104	0,02869982
SHA-1	0,01921604	0,02235979
SHA-224	0,02967178	0,0330969
SHA-256	0,03156621	0,03509256
SHA-384	0,0235117	0,02589208
SHA-512	0,02188918	0,02492324
SHA3-224	0,04186938	0,04580606
SHA3-256	0,03873666	0,04166005
SHA3-384	0,05427918	0,05594311
SHA3-512	0,06842647	0,07242418
Whirlpool	0,12854959	0,12983034

3. Panjang pesan = 100 juta huruf.
Berikut kecepatan ECDSA dengan fungsi *hash* berbeda untuk panjang pesan 100 juta huruf.

Tabel 5. Perbandingan kecepatan ECDSA dengan variasi fungsi *hash* untuk panjang pesan 100 juta huruf

Fungsi *hash*	S*ign Time* (s)	*Verify Time* (s)
BLAKE2b	0,22770009	0,22893663
BLAKE2s	0,33689702	0,34802833
BLAKE3	0,08114517	0,0814334
GOST	1767,667754	1746,261539
Streebog-256	3264,650731	3239,204847
Streebog-512	3215,61979	3202,184071
RIPEMD-160	0,56014456	0,55956884
MD2	16,10231199	16,11847616
MD4	0,21378602	0,21865664
MD5	0,18971417	0,1926816
SHA-1	0,15396192	0,15670608
SHA-224	0,2660486	0,26668827
SHA-256	0,26901886	0,27465761
SHA-384	0,19386704	0,19567696
SHA-512	0,19219161	0,19574255
SHA3-224	0,34170315	0,34079129
SHA3-256	0,36708112	0,3714835
SHA3-384	0,44634062	0,45192384
SHA3-512	0,6391657	0,64932706
Whirlpool	1,1603711	1,15316989

B. Analisis

Berdasarkan hasil ketiga eksperimen kecepatan tersebut, terlihat bahwa BLAKE3 merupakan fungsi *hash* yang memberikan kecepatan tertinggi dan paling *scalable.* BLAKE3 lebih cepat 1,09 kali dibandingkan SHA-512 untuk ukuran pesan 1 juta dan sekitar 1,9 kali lebih cepat dibandingkan SHA-1 untuk ukuran pesan 10 dan 100 juta. Dengan peningkatan 10 kali pada ukuran pesan, waktu yang dibutuhkan BLAKE3 untuk melakukan tanda tangan hanya meningkat 2,2279 kali untuk peningkatan ukuran pesan dari 1 juta ke 10 juta dan 7,8618 kali untuk peningkatan ukuran pesan dari 10 juta ke 100 juta.

Dari ketiga eksperimen, terdapat 8 fungsi *hash* yang memberikan kecepatan terlambat yang memiliki urutan yang sama. Urutan 8 fungsi *hash* tersebut adalah SHA3-384 > RIPEMD-160 > SHA3-512 > Whirlpool > MD2 > GOST > Streebog-512 > Streebog-256. Tiga fungsi *hash* terpelan, yaitu GOST, Streebog-512, dan Streebog-256 tidak cocok digunakan di ECDSA karena sangat lambat. Penandatanganan pesan berukuran 1 juta dengan fungsi *hash* GOST membutuhkan waktu 18,547 detik, sedangkan dengan fungsi *hash* Streebog membutuhkan waktu sekitar 36 detik. GOST dan Streebog juga kurang *scalable* dengan waktu tanda tangan yang dibutuhkan meningkat sekitar 9-10 kali untuk peningkatan ukuran pesan 10 kali. Jika dibandingkan dengan waktu BLAKE3 untuk penandatanganan panjang pesan 1 juta, GOST sekitar 4000 kali lebih lambat dan Streebog sekitar 7700 kali lebih lambat.

Dari 20 fungsi *hash* yang digunakan, 12 fungsi *hash* lainnya (selain 8 fungsi *hash* terpelan yang disebutkan di atas) memberikan kecepatan yang cukup baik. ECDSA membutuhkan kecepatan dan keamanan dari fungsi *hash* yang baik. Tidak semua fungsi *hash* dari 12 fungsi *hash* tersebut tergolong dalam fungsi *hash* yang aman. Oleh karena itu, 12 fungsi *hash* tersebut perlu disaring lebih lanjut untuk mendapatkan fungsi *hash* yang cepat dan aman. Fungsi *hash* aman apabila fungsi belum berhasil dirusak sepenuhnya, kuat terhadap kolisi, dan menghasilkan *message digest* minimal 256 *bits*. Dari 12 fungsi *hash* tersebut, 8 fungsi *hash*; yaitu BLAKE2b, BLAKE2s, BLAKE3, SHA-256, SHA-384, SHA-512, SHA3-256; memberikan kecepatan dan keamanan yang cukup baik untuk digunakan di ECDSA. Karena berdasarkan hasil eksperimen ditemukan bahwa BLAKE3 memberikan kecepatan dan *scalability* paling tinggi serta belum ditemukan

kelemahan pada BLAKE3, BLAKE3 merupakan fungsi *hash* yang paling cocok digunakan untuk algoritma tanda tangan ECDSA.

V. KESIMPULAN

Salah satu algoritma tanda tangan digital yang umum digunakan adalah ECDSA. Implementasi awal ECDSA menggunakan fungsi *hash* SHA-1 yang walaupun cukup cepat, tidak aman digunakan karena rentan terhadap *collision attack.* Untuk meningkatkan performa ECDSA, baik dari sisi kecepatan maupun keamanan, fungsi *hash* yang digunakan dapat diubah dengan fungsi *hash* lainnya. Berdasarkan hasil eksperimen, ditemukan bahwa BLAKE3 memberikan kecepatan, skalabilitas, dan keamanan yang tinggi sehingga cocok digunakan untuk algoritma tanda tangan digital ECDSA.

VI. UCAPAN TERIMA KASIH

Penulis ingin mengucapkan terima kasih kepada Bapak Dr. Ir. Rinaldi Munir, MT. selaku dosen pengajar mata kuliah Kriptografi 2020/2021 yang telah memberikan ilmu yang digunakan dalam penulisan makalah ini. Penulis juga ingin mengucapkan terima kepada teman-teman penulis yang membantu memberikan inspirasi topik makalah ini. Selain itu, penulis juga ingin berterima kasih kepada semua penulis dari berbagai referensi yang digunakan dalam penulisan makalah ini.

REFERENSI

[1] Rouse, Margaret. 2020. *Digital Signature*. Diakses pada 15 Desember 2020 dari https://searchsecurity.techtarget.com/definition/digital-signature#:~:text=A%20digital%20signature%20is%20a,message%2C%20software%20or%20digital%20document.

[2] Munir, Rinaldi. *Elliptic Curve Cryptography (ECC) Bagian 1*. Diakses pada 15 Desember 2020 dari http://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2020-2021/ECC-2020-Bagian1.pdf

[3] Maetouq, A., Daud, S. M., Ahmad, N. A., Maarop, N., Sjarif, N. N. A., & Abas, H. (2018). Comparison of hash function algorithms against attacks: A review. *International Journal of Advanced Computer Science and Applications*, *9*(8), 98–103. https://doi.org/10.14569/ijacsa.2018.090813

[4] Munir, Rinaldi. *Fungsi Hash*. Diakses pada 19 Desember 2020 dari http://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2020-2021/Fungsi-hash-2020.pdf

[5] Muller, F. (2004). The MD2 hash function is not one-way. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *3329*, 214–229. https://doi.org/10.1007/978-3-540-30539-2_16

[6] Rivest, R. L. (1991). The MD4 message digest algorithm. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *473 LNCS*, 492. https://doi.org/10.1007/3-540-46877-3_46

[7] Rivest, R. L., & Dusse, S. (1992). The MD5 message-digest algorithm.

[8] Rivest, R. L., Agre, B., Bailey, D. V., Crutchfield, C., Dodis, Y., Fleming, K. E., Khan, A., Krishnamurthy, J., Lin, Y., Reyzin, L., Shen, E., Sukha, J., Sutherland, D., Tromer, E., & Yin, Y. L. (2008). The MD6 hash function: A proposal to NIST for SHA-3. *Preprint*, 3. http://people.csail.mit.edu/rivest/pubs/RABCx08.pdf

[9] Team Keccak. *Keccak Specifications Summary*. Diakses dari https://keccak.team/keccak_specs_summary.html pada 19 Desember 2020.

[10] Dobbertin, H., Bosselaers, A., & Preneel, B. (1996). RIPEMD-160: A strengthened version of RIPEMD. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. https://doi.org/10.1007/3-540-60865-6_44

[11] Barreto, P. S. L. M., & Rijmen, V. (2011). The WHIRLPOOL Hashing Function. *Encyclopedia of Cryptography and Security*, 1384--1385. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.529.3184&rep=rep1&type=pdf

[12] Anonim. *BLAKE2 – fast secure hashing.* Diakses dari https://www.blake2.net/ pada 20 Desember 2020.

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, 20 Desember 2020

Nixon Andhika
13517059