Implementasi Fungsi Hash xxhash pada Tanda Tangan Digital

Faisal Helmi Wicaksono (18219025)
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail : faisal.helmi.w@gmail.com

***Abstract*—Meningkatnya penyebaran dokumen digital saat ini memunculkan kebutuhan akan mekanisme untuk menjamin keaslian dokumen dan autentikasi identitas pengirim dokumen. Tanda tangan digital dapat digunakan untuk memenuhi kebutuhan tersebut. Melalui tanda tangan digital yang ada pada sebuah dokumen, penerima dapat memverifikasi identitas pengirim dan keaslian dokumen yang diterimanya. Langkah awal pembentukan tanda tangan digital menggunakan hashing terhadap dokumen, yang dilanjutkan enkripsi terhadap message digest tersebut. Fungsi hash yang umum digunakan dalam tanda tangan digital adalah SHA-1, SHA-256, dan MD5 [1]. Pada makalah kali ini, penulis ingin mempelajari dan menganalisis implementasi fungsi hash lainnya yaitu xxhash yang memiliki klaim kecepatan hash yang tinggi dengan collision yang rendah [2] bersamaan dengan algoritma kriptografi RSA dalam tanda tangan digital. Dari implementasi tersebut, dapat dibandingkan performa bebeberapa hashing populer dengan xxhash.**

Keywords—tanda tangan digital; xxhash; performa;

I. PENDAHULUAN

Perkembangan internet sudah semakin pesat. Hal tersebut terlihat dari semakin meningkatnya jumlah pengguna internet di dunia. Di tahun 2020, jumlah pengguna internet mencapai 4.54 milyar orang atau mencapai 59% penduduk di dunia [3]. Menurut Badan Siber dan Sandi Negara [4], dari Januari – April 2020, tercatat 88.414.296 serangan siber. Hal ini membuktikan bahwa pengguna internet memiliki urgensi tinggi untuk menjamin *information sharing* yang dilakukan melalui platform digital terjamin keamanannya. Pada kondisi tersebut, tentunya kegiatan *document sharing* secara daring melalui internet juga akan meningkat, tak terlepas *document sharing* untuk dokumen penting yang memiliki nilai tinggi.

Hal yang dapat diterapkan untuk menangani permasalahan otentikasi dan otorisasi dokumen adalah dengan menggunakan tanda tangan digital. Tanda tangan digital merupakan salah satu solusi yang bisa menjamin *document sharing* lebih tepercaya dan aman. Ada banyak aspek yang bisa ditinjau dari tanda tangan digital; salah satunya adalah fungsi *hash* yang digunakan. Pada penelitian ini, fungsi *hash* yang tidak umum digunakan untuk tanda tangan digital yaitu xxhash akan diimplementasikan dan dibandingkan performanya dengan fungsi *hash* lainnya yang lebih umum digunakan. Pengujian performa yang dilakukan yaitu kecepatan eksekusi *signing* dan verifikasi dari tanda tangan digital.

II. DASAR TEORI

A. RSA

RSA merupakan algoritma kunci-publik yang paling terkenal dan paling banyak aplikasinya. Ditemukan oleh tiga peneliti dari MIT (*Massachusetts Institute of Technology*), yaitu Ronald Rivest, Adi Shamir, dan Len Adleman, pada tahun 1976. RSA merupakan singkatan Rivest-Shamir-Adleman [9]. RSA mempunyai dua kunci, yaitu kunci publik dan kunci pribadi. Kunci publik boleh diketahui oleh siapa saja, dan digunakan untuk proses enkripsi. Sedangkan kunci pribadi hanya pihak - pihak tertentu saja yang boleh mengetahuinya, dan digunakan untuk proses dekripsi.

Algoritma pembentukan kunci dari RSA yaitu:

1. Tentukan p dan q bernilai dua bilangan Prima besar, acak dan dirahasiakan. $p \neq q$, p dan q memiliki ukuran sama.
2. Hitung $n = pq$ dan hitung $(n) = (p-1)(q-1)$. Bilangan integer n disebut (RSA) modulus.
3. Tentukan e bilangan Prima acak, yang memiliki syarat:

   $1 < e < (n)$

   GCD(e, (n)) = 1, disebut e relatif prima terhadap (n),

   Bilangan integer e disebut (RSA) enciphering exponent.
4. Menghitung bilangan khusus d, yang memiliki syarat:

   $1 < d < (n)$

   $d \equiv e \bmod (n)^{-1}$

   $ed \equiv 1 \pmod{(n)}$

   $ed \equiv 1 + k.(n)$ untuk nilai k integer.

   Bilangan integer d disebut (RSA) deciphering exponent.
5. Nilai (n, e) adalah nilai yang boleh dipublikasi.

6. Nilai d, p, q, (n) adalah nilai yang harus dirahasiakan. Pasangan (n, e) merupakan kunci publik. Pasangan (n, d) merupakan kunci rahasia.

B. MD5

MD5 atau *Message-Digest algorithm* 5 adalah fungsi hash kriptografik. Algoritma ini terutama digunakan untuk melakukan pemeriksaan integritas file dalam berbagai situasi. Dalam ilmu kriptografi, MD5 adalah salah satu algoritma hash yang paling populer. Hash atau hashing sendiri adalah proses perubahan suatu data menjadi data lain dengan panjang tertentu, sedemikian sehingga data itu tidak dapat dipulihkan kembali. Teknik ini biasa digunakan dalam enkripsi data, misalnya untuk menyimpan password agar tidak ada yang dapat mengetahuinya meskipun dia dapat melihat hash dari password itu. MD5 adalah fungsi hash satu-arah yang dibuat oleh Ron Rivest [10]. MD5 merupakan perbaikan dari MD4 setelah MD4 ditemukan kolisinya. Algoritma MD5 menerima masukan berupa pesan dengan ukuran sembarang dan menghasilkan message digest yang panjangnya 128 bit.

Langkah-langkah untuk menghasilkan *message digest* adalah sebagai berikut.

- Penambahan bit-bit pengganjal

  Pesan mula-mula ditambahkan dengan bit-bit pengganjal sehingga panjang pesan (dalam satuan bit) paralel dengan modulo 512. Setelah panjang data menjadi kelipatan 512, selanjutnya akan dikurangi 64 bit.

- Penambahan nilai panjang pesan semula

  Representasi sebesar 64 bit tersebut akan ditambahkan ke pesan.

- Inisialisasi penyangga (*buffer*) MD5

  MD5 membutuhkan 4 buah register sebagai buf er dengan panjang masing-masing 32 bit, sehingga total panjang *buffer* sebesar 128 bit. Keempat register ini diberi nama A, B, C dan D yang diinisialisasi dalam nilai heksa sebagai berikut:

  A = 01 23 45 67

  B = 89 ab cd ef

  C = fe dc ba 98

  D = 76 54 32 10.

- Pengolahan pesan dalam blok bernilai 512 bit

  Dalam langkah ini, awalnya didefinisikan 4 buah fungsi dengan inputan 3 buah word 32 bit dan menghasilkan 1 buah word 32 bit. Inisialisasi fungsi dinyatakan dalam nama F, G, H dan I adalah sebagai berikut:

  F (X, Y, Z) = XY or (~X)Z

  G (X,Y,Z) = XZ or Y(~Z)

  H (X, Y, Z) = X xor Y xor Z

  I (X, Y, Z) = Y xor (X or (~Z))

Fungsi-fungsi tersebut masing-masing berisikan 16 kali operasi dasar terhadap inputan yang memanfaatkan suatu nilai elemen T. nilai T adalah sebuah array berisi 64 elemen yang didapat dengan perhitungan tertentu menggunakan fungsi sinus. Pesan yang di input dibagi menjadi n buah blok berukuran 512 bit. Setiap blok tersebut diproses bersama dengan *buffer* sehingga menghasilkan output sebesar 128 bit. Dalam proses ini terdiri dari 4 kali proses, masing-masing proses melakukan operasi tiap-tiap fungsi 16 kali banyaknya dan memakai elemen operasi dasar T. Hasil akhir dari algoritma MD5 ini akan menghasilkan output (message digest) dengan panjang yang tetap sebesar 32 karakter.

C. SHA1

SHA dikembangkan oleh *National Institute of Standards and Technology* (NIST) dan dipublikasikan sebagai *Federal Information Processing Standards* (FIPS 180) pada tahun 1993. *Secure Hash Standard* (SHS) menspesifikasikan SHA-1 untuk menghitung nilai hash dari sebuah message atau file. SHA-1 memiliki panjang pesan maksimal 264 bits dan memiliki keluaran sebesar 160 bits yang dinamakan *message digest* atau *hash code*. Message digest tersebut dapat dimanfaatkan sebagai input untuk *Digital Signature Algorithm* (DSA), yang digunakan untuk menghasilkan signature untuk memvalidasi pesan tersebut [11].

Pada saat ditemukan kelemahan pada algoritma SHA-0, berbagai revisi dan perbaikan dilakukan untuk membuat suatu algoritma yang lebih baik. Hasil dari perbaikan tersebut diterbitkan pada tahun 1995 dan dijadikan acuan untuk pembuatan algoritma SHA-1.

Algoritma SHA-1 merupakan revisi teknis dari algoritma SHA. Algoritma SHA-1 dapat dikatakan aman karena proses perhitungannya tidak memungkinkan untuk menemukan pesan yang sebenarnya dari message digest yang dihasilkan. Setiap perubahan yang terjadi pada pesan saat perjalanan akan menghasilkan message digest yang berbeda. Algoritma SHA-1 berbasis pada algoritma MD4 dan rancangannya sangatlah mirip terhadap algoritma tersebut.

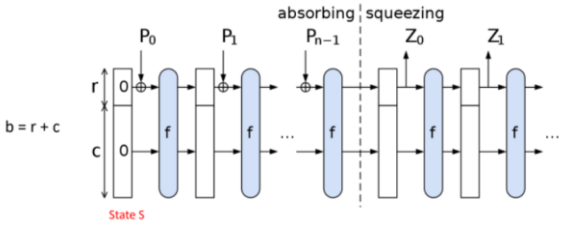
D. SHA3-256

Seperti sejarah AES, *National Institute of Standards and Technology* (NIST) menyelenggarakan kompetisi terbuka untuk mengembangkan fungsi *hash* yang baru, bernama SHA-3. SHA-3 menjadi komplementer SHA-1 dan SHA-2. Kompetisi diumumkan pada tahun 2007 dan berakhir pada Oktober 2012 dengan memilih pemenang [12].

Pemenang dari kompetisi tersebut Keccak. Nama 'Keccak' berasal dari kata 'Kecak', sebuah tari dari Bali. Keccak berbeda dari finalis SHA3 lainnya dalam hal menggunakan konstruksi 'spons' (*sponge construction*). Jika desain lainnya bergantung pada 'fungsi kompresi, Keccak menggunakan fungsi non-kompresi untuk menyerap dan kemudian 'memeras' *digest*. Desain Keccak berbeda dari pendekatan yang ada. NIST merasa bahwa dalam kasus ini, yang berbeda adalah lebih baik.

Rancangan SHA-3 adalah menggunakan konstruksi spons yaitu data diserap ke dalam spons dimana dalam fase ini blok

pesan XOR menjadi bagian dari status yang kemudian diubah keseluruhan menggunakan fungsi permutasi f kemudian hasilnya diperas dimana dalam fase ini blok keluaran dibaca dari subset yang sama dari keadaan dengan fungsi transformasi keadaan f.



Gambar 1
(sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi-dan-Koding/2021-2022/20%20-%20SHA-3-2020)

Fase Penyerapan (*Absorbing*):

- Untuk setiap blok masukan P_i berukuran r-bit, XOR-kan dengan r-bit pertama dari state S, lalu hasilnya dimasukkan ke dalam fungsi permutasi f untuk menghasilkan state baru S.
- Bila semua blok masukan selesai diproses, konstruksi spons beralih ke fase pemerasan (*squeezing*).

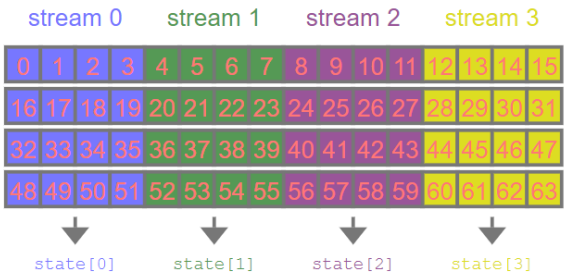
Fase Pemerasan (*Squeezing*):

- *Message digest* akan disimpan di dalam Z.
- Inisialisasi Z dengan string kosong (*null string*).
- Selagi panjang Z belum sama dengan d, r-bit pertama dari *state* S disambungkan (*append*) ke Z.
- Jika panjang Z masih belum sama dengan d, masukkan ke dalam fungsi permutasi f menghasilkan state baru S.
- Perhatikan bahwa c bit terakhir dari state tidak pernah terpengaruh secara langsung oleh blok masukan dan tidak pernah mengeluarkan luaran selama fase pemerasan. Hal ini untuk mencegah terjadinya kolisi.

E. xxHash

Xxhash dirancang dari awal untuk secepat mungkin pada CPU modern. Fungsi ini bukan *hash* kriptografis yang kuat, seperti keluarga SHA, tetapi masih melewati set tes *SMHasher* dengan skor 10 poin.

xxHash membagi data input menjadi empat aliran independen. Setiap aliran memproses blok 4 *byte* per langkah dan menyimpan status sementara. Hanya langkah terakhir yang menggabungkan keempat status ini menjadi satu.



Gambar 2
(sumber: https://create.stephan-brumme.com/xxhash/)

Berikut ini tabel yang berisi perbandingan performa fungsi hash yang digunakan pada percobaan kali ini [2].

Tabel 1 Perbandingan Performa Beberapa Fungsi *Hash*

Hash Name	***Width***	***Bandwidth* (GB/s)**	***Small Data Velocity***	***Quality***
XXH3 (SSE2)	64	31.5 GB/s	133.1	10
XXH128 (SSE2)	128	29.6 GB/s	118.1	10
SHA1	160	0.8 GB/s	5.6	10
MD5	128	0.6 GB/s	7.8	10

III. DESAIN PROGRAM

Untuk menguji berbagai fungsi hash yang digabungkan dengan algoritma RSA, penulis membuat aplikasi desktop menggunakan bahasa pemrograman Python dengan GUI PyQt. Bahasa Python dipilih karena memiliki *library* kriptografi yang cukup lengkap.

Library dan *source* yang digunakan dalam implementasi program tanda tangan digital adalah sebagai berikut:

1. RSA menggunakan modifikasi program m implementasi tugas mata kuliah II4031 Kriptografi & Koding milik kelompok penulis [5].
2. xxHash menggunakan *library* untuk Python yang diimplementasikan oleh Yue Du [6].
3. Fungsi hash SHA1, SHA3-256, dan MD5 menggunakan library hashlib [7].

Berikut adalah spesifikasi program tanda tangan digital yang dibuat:

1. Dokumen yang dapat diuji memiliki format (.txt), (.jpg), (.mp3), dan (.mp4)
2. Pasangan kunci publik dan kunci privat yang digunakan selama pengujian memiliki nilai tetap

3. Program menggunakan GUI dan dilengkapi *timer* untuk melihat kecepatan proses *signing* dan verifikasi.

IV. HASIL EKSPERIMEN DAN PEMBAHASAN

Setelah kode selesai ditulis [8], penulis melakukan eksperimen secara individu yang dimulai pada 13 Mei 2022. Eksperimen dilakukan menggunakan perangkat keras dan lunak dengan spesifikasi sebagai berikut:

1. *Operating system*: Windows 11
2. *Processor*: Intel® Core™ i5-8250U CPU @ 1.60GHz 1.8 GHz
3. RAM: 16.00 GB
4. *System type*: 64-bit *operating system*, x64-*based processor*

Eksperimen untuk setiap fungsi *hash* dan setiap format file dilakukan sebanyak lima kali, sehingga untuk setiap *hashing* akan didapatkan 20 pasang nilai durasi *signing* dan verifikasi. Dari 20 pasang tersebut, terdapat empat jenis nilai durasi *signing* dan verifikasi terhadap format file yang berbeda. Berikut durasi *signing* dan verifikasi dari empat fungsi *hash* tanda tangan digital. Durasi yang ditulis memiliki satuan sekon.

Tabel 2 Hasil Eksperimen MD5 untuk Format (.txt)

Percobaan ke-	***Signing***	**Verifikasi**
1	0.0070433617	0.0040345192
2	0.0049760342	0.0039916039
3	0.0049984455	0.0040321355
4	0.0040347576	0.0040328503
5	0.0049941544	0.0049943924
Rata-Rata	0.0052093507	0.0042171003

Tabel 3 Hasil Eksperimen MD5 untuk Format (.jpg)

Percobaan ke-	***Signing***	**Verifikasi**
1	0.2809505463	0.2190389633
2	0.2180008888	0.2215061188
3	0.2210276127	0.2179899216
4	0.2219955921	0.2189953327
5	0.2169976234	0.2200362682
Rata-Rata	0.2317944527	0.2195133209

Tabel 4 Hasil Eksperimen MD5 untuk Format (.mp3)

Percobaan ke-	***Signing***	**Verifikasi**
1	0.7560107708	0.7613680363
2	0.7652671337	0.7540416718
3	0.7714996338	0.7570452691
4	0.7524240017	0.7619805336
5	0.7620530128	0.7560388658
Rata-Rata	0.7614509106	0.7580948753

Tabel 5 Hasil Eksperimen MD5 untuk Format (.mp4)

Percobaan ke-	***Signing***	**Verifikasi**
1	1.5930113792	1.6680390835
2	1.6089994907	1.6679944921
3	1.6390366554	1.6570186615
4	1.5960264206	1.7151994705
5	1.5880575181	1.6660399437
Rata-Rata	1.605026293	1.67485833

Tabel 6 Hasil Eksperimen SHA1 untuk Format (.txt)

Percobaan ke-	***Signing***	**Verifikasi**
1	0.0089979172	0.0049948692
2	0.0049967766	0.0039935112
3	0.0060653687	0.0039999485
4	0.0050415993	0.0049958229
5	0.0050384998	0.0039961338
Rata-Rata	0.00602803232	0.00439605712

Tabel 7 Hasil Eksperimen SHA1 untuk Format (.jpg)

Percobaan ke-	***Signing***	**Verifikasi**
1	0.2159976959	0.2240335941
2	0.2140412331	0.2139976025
3	0.2220358849	0.2189908028

4	0.2154524326	0.2130320072
5	0.2200558186	0.2129902841
Rata-Rata	0.217516613	0.2166088581

Tabel 8 Hasil Eksperimen SHA1 untuk Format (.mp3)

Percobaan ke-	***Signing***	**Verifikasi**
1	0.7560467721	0.7440412045
2	0.7350375652	0.7650008202
3	0.7531170845	0.7455084324
4	0.7359900475	0.7347111702
5	0.7310452461	0.7369914055
Rata-Rata	0.7422473431	0.7452506066

Tabel 9 Hasil Eksperimen SHA1 untuk Format (.mp4)

Percobaan ke-	***Signing***	**Verifikasi**
1	1.5940427781	1.6438288689
2	1.5979893208	1.6090304852
3	1.5753204823	1.5967857838
4	1.5539090633	1.6229898931
5	1.5909883976	1.6400456429
Rata-Rata	1.582450008	1.622536135

Tabel 10 Hasil Eksperimen SHA3-256 untuk Format (.txt)

Percobaan ke-	***Signing***	**Verifikasi**
1	0.0050098896	0.0040426254
2	0.0050373077	0.0039966106
3	0.0070312023	0.0050325394
4	0.0050141811	0.0039947033
5	0.0039975643	0.0040428638
Rata-Rata	0.005218029	0.0042218685

Tabel 11 Hasil Eksperimen SHA3-256 untuk Format (.jpg)

Percobaan ke-	***Signing***	**Verifikasi**
1	0.2279460431	0.2190098763
2	0.2190463543	0.2220020294
3	0.2189960481	0.2166311741
4	0.2190136909	0.2229943275
5	0.2149994373	0.2139949799
Rata-Rata	0.2200003147	0.2189264774

Tabel 12 Hasil Eksperimen SHA3-256 untuk Format (.mp3)

Percobaan ke-	***Signing***	**Verifikasi**
1	0.7688806057	0.7788102627
2	0.7790079117	0.7579915524
3	0.7800419331	0.7640371323
4	0.7720170021	0.7669906616
5	0.7749094963	0.7630550861
Rata-Rata	0.7749713898	0.766176939

Tabel 13 Hasil Eksperimen SHA3-256 untuk Format (.mp4)

Percobaan ke-	***Signing***	**Verifikasi**
1	1.6108996868	1.6090416908
2	1.6170611382	1.6340327263
3	1.6220118999	1.6319935322
4	1.6018047333	1.6370007992
5	1.6282477379	1.7119555473
Rata-Rata	1.616005039	1.644804859

Tabel 14 Hasil Eksperimen xxHash untuk Format (.txt)

Percobaan ke-	***Signing***	**Verifikasi**
1	0.0060362816	0.0050535202
2	0.0040006638	0.0039956571
3	0.0039956571	0.0040540695

4	0.0050320625	0.0039997101
5	0.0039982796	0.0040619373
Rata-Rata	0.00461258892	0.00423297884

Tabel 15 Hasil Eksperimen xxHash untuk Format (.jpg)

Percobaan ke-	***Signing***	**Verifikasi**
1	0.2404189111	0.2210121155
2	0.2173511982	0.2126328945
3	0.22570467	0.2160432339
4	0.2199923992	0.2149806023
5	0.2186141014	0.2161478996
Rata-Rata	0.224416256	0.2161633492

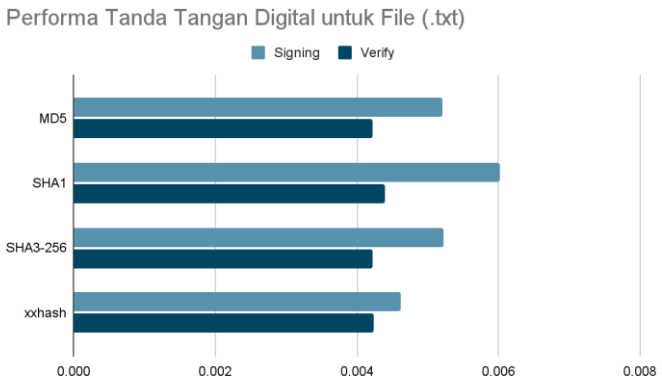
Tabel 16 Hasil Eksperimen xxHash untuk Format (.mp3)

Percobaan ke-	***Signing***	**Verifikasi**
1	0.7410519123	0.767996788
2	0.7536795139	0.7790312767
3	0.762475729	0.7515046597
4	0.7540531158	0.7520418167
5	0.747574091	0.7532496452
Rata-Rata	0.7517668724	0.7607648373

Tabel 17 Hasil Eksperimen xxHash untuk Format (.mp4)

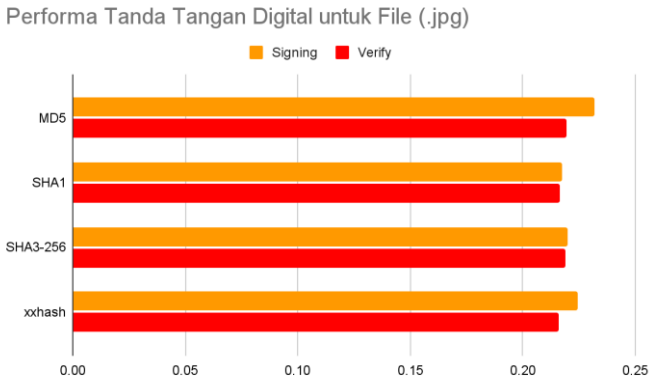
Percobaan ke-	***Signing***	**Verifikasi**
1	1.6120569706	1.6190414429
2	1.7325150967	1.6326019764
3	1.6071426868	1.6370298862
4	1.5720341206	1.6405003071
5	1.5781166553	1.6170327663
Rata-Rata	1.620373106	1.629241276

Rata-rata durasi *signing* dan verifikasi dari keempat jenis tanda tangan digital akan di-*plot* menjadi grafik *bar chart* untuk setiap format file agar dapat dibandingkan dengan lebih mudah.



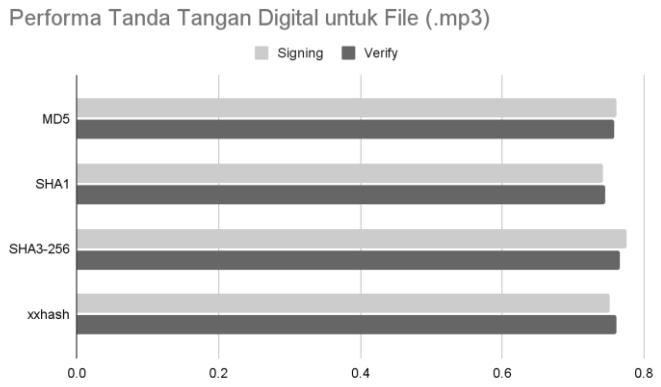
Gambar 3 Performa Tanda Tangan Digital untuk File (.txt)
(sumber: dokumentasi penulis)

Dapat dilihat bahwa performa tanda tangan digital untuk format file (.txt) memiliki performa tercepat dengan menggunakan fungsi hashing xxhash. Namun demikian, perbedaan kecepatan tersebut tidak sesuai pada sumber kajian pustaka yang terkait kecepatan hashing xxhash. Kemungkinan besar karena ukuran file yang sangat kecil.



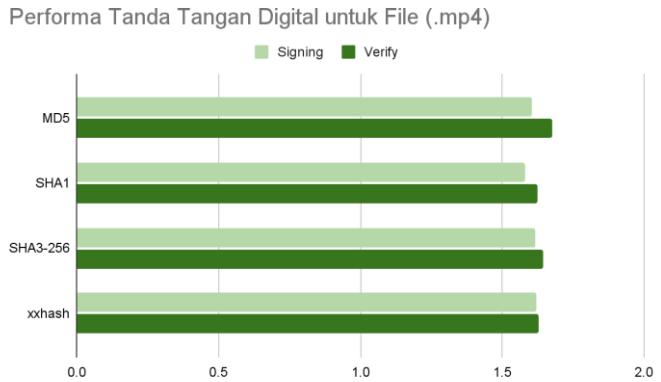
Gambar 4 Performa Tanda Tangan Digital untuk File (.jpg)
(sumber: dokumentasi penulis)

Dapat dilihat pada grafik, performa tanda tangan digital untuk format file (.jpg) memiliki performa paling cepat pada fungsi hash SHA1. Performa fungsi hash xxhash tidak berjalan sesuai klaimnya. Hal ini bisa jadi disebabkan karena algoritma pembacaan file untuk format (.jpg) kurang efisien sehingga menjadi kurang kompatibel dengan algoritma RSA maupun algoritma fungsi xxhash.



Gambar 5 Performa Tanda Tangan Digital untuk File (.mp3)
(sumber: dokumentasi penulis)

Dapat dilihat pada grafik, performa tanda tangan digital untuk format file (.mp3) memiliki performa paling cepat pada fungsi hash SHA1. Sekali lagi, performa fungsi hash xxhash tidak berjalan sesuai klaimnya. Hal ini bisa jadi disebabkan karena algoritma pembacaan file untuk format (.jpg) kurang efisien sehingga menjadi kurang kompatibel dengan algoritma RSA maupun algoritma fungsi xxhash.



Gambar 6 Performa Tanda Tangan Digital untuk File (.mp4)
(sumber: dokumentasi penulis)

Dapat dilihat pada grafik, performa tanda tangan digital untuk file dengan format (.mp4) memiliki performa paling baik pada fungsi hash SHA1. Sekali lagi, performa fungsi hash xxhash tidak berjalan sesuai klaimnya. Hal ini bisa jadi disebabkan karena algoritma pembacaan file untuk format (.jpg) kurang efisien sehingga menjadi kurang kompatibel dengan algoritma RSA maupun algoritma fungsi xxhash.

V. KESIMPULAN

Durasi *signing* milik fungsi hashing xxhash tidak lebih cepat daripada SHA1 untuk semua file. Xxhash justru memiliki durasi yang lebih lama daripada SHA1 untuk format file (.jpg), (.mp3), (.mp4). Akan tetapi, dibanding tanda tangan digital dengan fungsi hash MD5 dan SHA3-256, xxhash memiliki rata-rata durasi yang lebih singkat.

Hal ini bisa jadi disebabkan oleh adanya implementasi kode yang kurang efisien. Kejadian ini sangat mungkin terjadi mengingat *source code* xxhash yang digunakan bukan berasal dari *standard library* Python yang di-*maintain* secara rutin.

Hipotesis lain dari mengapa performa xxhash tidak seperti klaim [2] adalah ada kemungkinan performa *hash* secara individu, akan berbeda dari performa *hash* jika diimplementasikan bersama algoritma kriptografi lain. Dalam percobaan ini yaitu algoritma RSA untuk membangun suatu program tanda tangan digital. Bisa jadi jika diuji secara individu, performa xxhash jauh lebih baik, namun kurang *compatible* dengan implementasi tanda tangan digital.

Selain itu, algoritma yang penulis bangun juga dapat menjadi alasan mengapa xxhash tidak memiliki performa sesuai klaimnya [2]. Dalam membangun kode untuk menadatangani berbagai format file, penulis membangun algoritma untuk mengurainya agar dapat dibubuhi tanda tangan digital. Penguraian berbagai format file tersebut dapat menjadikan performa xxhash dalam proses *signing* maupun verifikasi menurun.

Dari hasil analisis dan eksperimen yang dilakukan, dapat disimpulkan bahwa secara umum performa tanda tangan digital dari xxhash tidak lebih baik dari fungsi *hash* lain. Hal ini dapat menjadi alasan mengapa berbagai fungsi *hash* yang secara teoritis memiliki performa lebih cepat dibandingkan SHA1, SHA3-256, dan MD5 tidak lazim digunakan.

VI. UCAPAN TERIMA KASIH

Dengan ini, penulis mengucapkan syukur kepada Tuhan Yang Maha Esa atas berkatNya telah diberi kesempatan untuk mempelajari topik yang diangkat dalam makalah ini dan dipermudah dalam proses penyelesaiannya. Penulis juga mengucapkan terima kasih kepada orang tua saya yang tak henti memberikan dukungan kepada saya, dan kepada Pak Rinaldi Munir selaku dosen mata kuliah II4031 Kriptografi dan Koding yang telah membimbing saya dalam belajar ilmu terkait kriptografi. Selain itu, saya juga mengucapkan terima kasih kepada teman-teman Sistem dan Teknologi Informasi yang telah membantu dalam proses belajar saya di kelas Kriptografi dan Koding.

REFERENSI

[1] US Department of Homeland Security CISA Cyber + Infrastructure. (2020, Aug. 20). *Understanding Digital Signatures* [Online]. Available: https://us-cert.cisa.gov/ncas/tips/ST04-018

[2] xxHash. (n.d.). *xxHash Benchmarks, Multiple Languages*, [Online]. Available: https://cyan4973.github.io/xxHash/

[3] we are social. (2020, Jan. 20). *Digital 2020: 3.8 Billion People Use Social Media* [Online]. Available: https://wearesocial.com/uk/blog/2020/01/digital-2020-3-8-billion-people-use-social-media/

[4] Badan Siber dan Sandi Negara. (2020, Apr. 2020). Rekap Serangan Siber (Januari-April 2020) [Online]. Available: https://bssn.go.id/rekap-serangan-siber-januari-april-2020/

[5] Github. (2022, Apr. 17). *Tugas4Kripto* [Online]. Available: https://github.com/rezamnf/Tugas4Kripto

[6] Pypi. (2022, Feb. 24). *xxhash 3.0.0* [Online]. Available: https://pypi.org/project/xxhash/

[7] Python. (n.d.). hashlib — Secure hashes and message digests [Online]. Available: https://docs.python.org/3/library/hashlib.html

[8] Github. (2022, May. 14). *Percobaan Beberapa Fungsi Hash* [Online]. Available: https://github.com/faisalhelmi/xxhashDigitalSigningProject

[9] R. Munir, "Algoritma RSA", [online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi-dan-Koding/2021-2022/12%20-%20Algoritma-RSA-2021.pdf

[10] R. Munir, "Algoritma MD5", [online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi-dan-Koding/2021-2022/18%20-%20Fungsi-hash-MD5-2021.pdf

[11] R. Munir, "Secure Hash Algorithm (SHA)", [online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi-dan-Koding/2021-2022/19%20-%20Fungsi-hash-SHA-2021.pdf

[12] R. Munir, "SHA-3 (Keccak)", [online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi-dan-Koding/2021-2022/20%20-%20SHA-3-2020

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, 25 Mei 2022

Faisal Helmi Wicaksono (18219025)