Aplikasi Fungsi *Hash* sebagai Metode Alternatif Penjaminan Integritas Pengumpulan Tugas Daring

Bimo Adityarahman Wiraputra 13517004
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13517004@std.stei.itb.ac.id

***Abstrak*—Dalam kasus pengumpulan tugas secara daring dalam lingkungan akademis, sewaktu-waktu dapat ditemui kebutuhan untuk memverifikasi kesamaan suatu berkas dengan berkas sebelumnya yang tersedia sebelum suatu batas waktu, seperti saat demo aplikasi. Sebagai alternatif dari cara yang sebelumnya sudah digunakan seperti meninjau *git log*, makalah ini menawarkan fungsi *hash* sebagai alternatif cara yang lebih serba guna, terpercaya, dan ringan. Cara ini juga lebih mudah dibandingkan dengan protokol *timestamp authority* yang sudah ada sebelumnya.**

Keywords*—fungsi *hash*, integritas berkas, *timestamp authority

I. Pendahuluan

Dalam lingkungan akademis seperti Teknik Informatika Institut Teknologi Bandung, tugas yang diberikan kepada mahasiswa secara daring secara umum memiliki metode pengumpulan berupa pengunggahan dokumen atau berkas terkait melalui suatu saluran daring seperti surel, situs *file hosting* seperti Google Drive atau Dropbox, ataupun situs *remote repository* seperti Github atau Gitlab Informatika. Dalam beberapa waktu, dapat terjadi gangguan ketersediaan situs pengumpulan seperti Gitlab Informatika mendekati waktu pengumpulan karena tingginya permintaan akses ke situs tersebut. Selain itu, untuk beberapa jenis tugas seperti pembuatan aplikasi, seringkali kegiatan demo aplikasi dilakukan oleh mahasiswa terkait di lain waktu untuk menunjukkan fungsionalitas aplikasi tersebut. Untuk kedua kasus tersebut, keberadaan suatu cara untuk menjamin integritas suatu berkas yang dikumpulkan atau ditunjukkan di waktu kemudian sebagai berkas yang sama dengan berkas yang tersedia sebelum batas waktu pengumpulan berkas tersebut.

Salah satu dari pendekatan yang telah dilakukan untuk menjamin hal tersebut adalah mengecek *log* dari repositori tugas untuk menunjukkan apakah ada perubahan semenjak *commit* terakhir yang dilakukan sebelum batas waktu. Akan tetapi, pendekatan ini hanya terbatas untuk kasus demo aplikasi yang melakukan pengumpulan menggunakan situs repositori, dan rentan terhadap manipulasi repositori yang susah untuk dideteksi tanpa memiliki akses khusus di situs repositori terkait.

Pendekatan yang diusulkan di makalah ini adalah dengan memanfaatkan sifat khusus dari fungsi *hash* sebagai alternatif metode yang lebih serba guna, terpercaya, dan ringan. Metode ini mengambil inspirasi dari *timestamp authority* atau suatu *server third-party* yang bertugas untuk menjamin integritas suatu dokumen dengan menandatangani dokumen tersebut yang sudah diberikan *timestamp*. Memperhatikan sudah maraknya penanda waktu terintegrasi dengan berbagai aplikasi secara umum yang dapat dipercaya, penggunaan fungsi *hash* secara langsung ditinjau lebih mudah dan sama terpercayanya daripada menggunakan *timestamp authority*.

II. Dasar Teori

A. Fungsi *Hash*

Fungsi *hash* merupakan suatu fungsi yang digunakan untuk memetakan data dengan ukuran berapapun menjadi suatu nilai berukuran pasti. Fungsi *hash* yang baik bersifat acak dan akan berubah secara drastis terhadap perubahan sekecil apapun di data awal, sehingga sangat sulit untuk mencari *collision* atau masukan data lainnya yang akan menghasilkan *hash* yang sama dengan suatu masukan data. Fungsi *hash* memiliki beberapa aplikasi diantaranya adalah sebagai indeks dari *hash table* untuk menyimpan dan mengambil objek sembarang secara efisien. Selain itu, *hash* juga memiliki aplikasi di bidang kriptografi sebagai bagian dari teknik penandatanganan digital.

Beberapa properti yang dimiliki dari fungsi *hash* yang baik secara kriptografi diantaranya adalah :

1) Bersifat deterministik menghasilkan luaran yang sama untuk semua masukan data yang sama pada saat kapanpun
2) Cepat dalam mengkomputasi nilai *hash*
3) Sangat sulit untuk membangkitkan suatu masukan data yang akan menghasilkan nilai *hash* tertentu
4) Sangat sulit untuk mencari dua masukan data berbeda yang akan menghasilkan nilai *hash* yang sama
5) Perubahan kecil pada data masukan sangat mengubah nilai *hash* sehingga terlihat tidak terkait dengan nilai *hash* yang lama

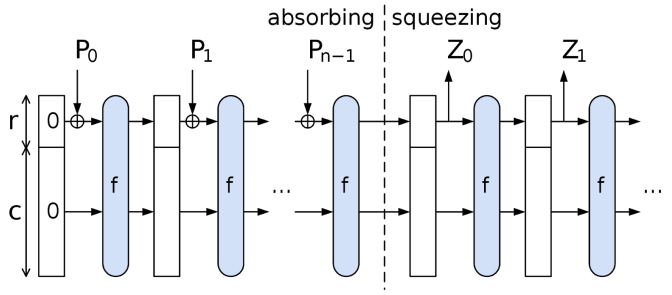
Diantara dari fungsi *hash* yang sering digunakan adalah MD-5, SHA-1, SHA-2, dan SHA-3.

B. SHA-3

SHA-3 (*Secure Hash Algorithm 3)* merupakan salah satu standar fungsi *hash* yang dipublikasikan oleh badan *National*

Institute of Standards and Technology (*NIST*) dari Amerika Serikat. Fungsi *hash* ini diadopsi dari fungsi *hash* Keccak yang dikembangkan oleh Guido Bertoni dan kawan-kawan sebagai pemenang dari kompetisi fungsi *hash* yang diadakan *NIST* sendiri untuk mencari pengganti dari fungsi *SHA-2*, pendahulu dari fungsi *SHA-3*.

Fungsi *SHA-3* memiliki arsitektur yang sangat berbeda dari pendahulunya. *Hash* ini menggunakan prinsip *sponge construction*, dimana terdapat suatu blok data yang terus dipermutasikan dan dicampurkan dengan bagian-bagian data masukan yang disebut sebagai proses absorpsi, dan dipermutasikan dan mengeluarkan bagian-bagian data luaran yang disebut sebagai proses *squeeze*. Proses ini digambarkan pada gambar di bawah, dimana di blok data tersebut terdapat bagian awal yang digunakan saat proses absorpsi maupun *squeeze*. Di tiap tahap dilakukan permutasi dari blok data yang ada melalui beberapa sub tahap diantaranya adalah xor, permutasi, fungsi *and* dan xor dengan suatu konstan.



Sebagai standar terbaru dari keluarga *SHA*, fungsi *SHA-3* masih dianggap sebagai fungsi *hash* yang aman secara kriptografi dan masih belum ditemukan celah kerentanan dari fungsi *hash* tersebut untuk dapat diserang secara efisien, mencakup jenis serangan *collision attack* dan *length extension attack*.

Ada berbagai jenis *SHA-3* yang disediakan oleh standar yang ada, diantaranya adalah *SHA3-224*, *SHA3-256*, *SHA3-384*, *SHA3-512*, *SHAKE128*, dan *SHAKE256* dimana penamaan *SHA3* berdasarkan dari ukuran *hash* yang dibangkitkan dan *SHAKE* dapat menghasilkan luaran dengan ukuran sembarang dan nomornya menandakan *rate* absorpsinya, walaupun menjadi ada variasi di dalam algoritmanya sendiri seperti berapa data masukan yang diproses dalam satu waktu.

C. Tanda Tangan Digital

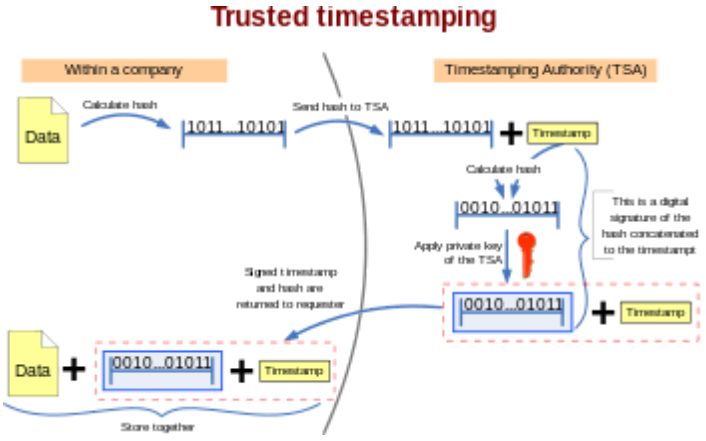
Tanda tangan digital merupakan suatu teknik kriptografi untuk menjamin keaslian dari suatu pesan maupun suatu data. Teknik ini sangat mirip dengan teknik enkripsi kunci publik di kriptografi dimana pembuat tanda tangan memiliki suatu kunci privat yang hanya dimiliki oleh dirinya sendiri, dan pasangan kunci publiknya yang dapat disebarkan secara terbuka ke siapapun. Prinsip dari teknik seperti ini adalah kunci privat sangat sulit untuk dibangkitkan walaupun kita mengetahui pasangan kunci publiknya. Secara umum, teknik tanda tangan digital dilakukan dengan pertama mencari nilai *hash* dari data yang hendak ditandatangani. Berdasarkan sifat dari fungsi *hash*, sangat sulit untuk memanipulasi data awal dengan tetap mempertahankan nilai *hash*nya. Fungsi *hash* ini lalu dienkripsi menggunakan kunci privat yang dimiliki oleh penanda tangan dan disisipkan di ujung data yang akan ditandatangani. Dengan ini, siapapun dapat memverifikasi keaslian dari data yang telah ditandatangani dengan membandingkan *hash* dari isi data yang ada dengan hasil dekripsi dari tanda tangan yang telah diberikan di ujung data. Karena hasil enkripsi hanya dapat dibuat oleh pemegang kunci privat, maka akan sangat sulit oleh orang lain memalsukan tanda tangan dari seseorang secara kriptografi.

Tanda tangan digital memiliki banyak aplikasi dalam memastikan keaslian dokumen digital, seperti penandatanganan dokumen formal secara digital dan memastikan keaslian dari identitas suatu pihak yang berkomunikasi secara digital.

Beberapa jenis algoritma tanda tangan digital yang sering digunakan adalah RSA yang berdasarkan dari masalah sulitnya memfaktorkan suatu bilangan yang besar, DSA berdasarkan algoritma kriptografi Elgamal yang memanfaatkan masalah sulitnya melakukan logaritma diskrit di suatu grup modulo prima, dan EcDSA maupun edDSA yang berdasarkan algoritma kriptografi di ruang *elliptic curve* yang memanfaatkan masalah sulitnya melakukan logaritma diskrit di grup *elliptic curve* modulo prima.

D. Timestamp Authority

Proses *timestamp* terpercaya adalah proses memastikan waktu pembuatan dan modifikasi dari suatu berkas sehingga tidak ada seorangpun mencakup pembuat berkas tersebut yang bisa mengubah berkas tersebut setelah waktu pemberian *timestamp* tanpa merusak integritas dari *timestamp* tersebut. Proses ini dapat digunakan untuk menjaga keaslian dari suatu berkas setelah suatu waktu dalam bisnis dan aplikasi lainnya seperti dalam transaksi *blockchain*.



Walaupun proses *timestamp* ini dapat dilakukan secara terdesentralisasi pada lingkungan seperti *blockchain*, salah satu cara termudah untuk mendapatkan *timestamp* ini adalah dengan mengandalkan *timestamp authority*, yang merupakan pihak ketiga yang terpercaya berdasarkan suatu standar untuk menyediakan *timestamp* ini. Diatur dalam beberapa standar seperti RFC 3161, teknik *timestamp* ini didasarkan pada tanda tangan digital dan fungsi *hash*. Berkas yang hendak ditandatangani pertama dimasukkan ke dalam fungsi *hash* untuk mendapatkan penanda dari berkas ini. Nilai *hash* ini lalu dikirim ke pihak *timestamp authority* yang kemudian akan membubuhkan *timestamp* ke *hash* tersebut dan menghitung

hash dari gabungan data ini. *Hash* ini lalu ditandatangani dengan kunci privat yang hanya dimiliki oleh otoritas ini. Protokol ini digambarkan di diagram yang ada di atas.

Pengguna dapat memverifikasi keaslian dari tanda tangan ini dengan membandingkan *hash* berkas, waktu *timestamp* yang diberikan, dan kunci publik yang disediakan secara terbuka dari *timestamp authority*. Secara umum, standar protokol *timestamp authority* juga mengatur berbagai hal seperti pengambilan sumber yang terpercaya untuk mendapatkan *timestamp* untuk menjaga integritas dan keandalan dari *timestamp* yang diberikan.

Beberapa dari *server* publik yang menyediakan jasa dari *timestamp authority* diantaranya adalah freeTSA.org, safestamper TSA, dan time.certum.pl .

E. Git

Git merupakan suatu *version control system* terdistribusi yang digunakan untuk melacak perubahan dari suatu berkas. Kakas ini sering digunakan oleh programmer dalam mengembangkan kode dimana perubahan dari suatu kumpulan kode dapat dilacak dengan baik dan dapat dikembalikan keadaannya seperti semula apabila ditemukan masalah. Kakas ini juga marak digunakan untuk sekelompok programmer dalam mengembangkan suatu kode bersama, dengan berkoordinasi dengan suatu sumber penyimpanan kode secara daring bernama repositori *remote* dimana perubahan kode yang dilakukan oleh seorang programmer dapat diunggah dan diintegrasikan dengan perubahan yang dilakukan oleh programmer lain.

Dalam sejarahnya, *git* dikembangkan oleh Linus Torvald untuk digunakan dalam pengembangan kernel Linux. Beberapa filosofi yang ada saat pengembangan *git* diantaranya adalah mendukung alur kerja terdistribusi seperti produk BitKeeper yang ada sebelumnya, mendukung sistem keamanan yang tinggi terhadap korupsi data, dan cepat untuk digunakan tidak seperti *concurrent versions system*.

Dengan adanya situs repositori *remote* yang terkenal seperti Github dan Gitlab sekarang, hampir semua pengembangan aplikasi skala menengah ke atas mengandalkan *git* untuk mengkoordinasikan pekerjaan dari beberapa programmer secara sekaligus.

Beberapa fungsi dasar yang tersedia dari kakas *git* diantaranya:

1) *git config*, untuk memberi tahu *git* informasi seperti nama maupun surel pengguna
2) *git init*, untuk memulai repositori lokal baru
3) *git clone*, untuk membuat kopian lokal dari suatu repositori *remote*
4) *git add*, untuk menambahkan satu atau lebih *file* ke dalam tahap *staging*
5) *git commit*, untuk menambahkan *file* yang ada dalam tahap *staging* untuk disimpan perubahannya sebagai bagian dari *version control*
6) *git push*, untuk mengunggah perubahan di repositori lokal ke repositori *remote* yang terhubung
7) *git pull*, untuk mengunduh perubahan di repositori *remote* yang terhubung ke repositori lokal
8) *git status*, untuk menunjukkan status dari repositori seperti *file* yang ada di *staging* dan perbedaan dengan repositori *remote*
9) *git log*, untuk menunjukkan sejarah *commit* yang ada di repositori

Pengumpulan berkas daring dapat dilakukan menggunakan situs repositori *remote* apabila pekerjaannya berbentuk kode dan dilacak menggunakan kakas *git* dengan penggunaan situs repositori *remote*. Mahasiswa harus melakukan *git push commit* yang telah dia lakukan ke repositori *remote* sebelum batas waktu pengumpulan yang telah dilakukan. Pihak penilai tugas dapat meminta akses ke situs tersebut untuk melihat kode apa yang telah diunggah oleh mahasiswa.

III. Rancangan Solusi dan Implementasi

Dalam membangun suatu teknik untuk memverifikasi suatu berkas dengan berkas yang tersedia di waktu sebelumnya, dapat digunakan fungsi *hash* analog dengan prinsip dibalik *timestamp authority*. Tahapan dari prosedur yang dilakukan adalah seperti berikut:

1) Mahasiswa membuat nilai *hash* dari tugas yang telah dia kerjakan sebelum batas waktu yang telah ditentukan menggunakan fungsi yang aman seperti *SHA-3*
2) Mahasiswa menyimpan nilai *hash* tersebut di suatu tempat yang bisa menjamin keberadaan nilai tersebut sebelum batas waktu yang telah ditentukan

Untuk tahap kedua, hal ini dapat dilakukan menggunakan *timestamp authority*, tetapi melihat banyak sumber-sumber yang dapat dipercaya untuk memberikan *timestamp*, sumber lain tersebut bisa digunakan agar lebih mudah dibandingkan menggunakan *timestamp authority*. Sumber lain ini diantaranya adalah dengan mengirimkan *hash* melalui aplikasi *chat* seperti *Whatsapp*, Telegram, maupun LINE, menyimpan *hash* tersebut di tempat penyimpanan daring seperti pastebin maupun Google Drive, ataupun disamakan dengan teknik pengumpulan tugas konvensional seperti menggunakan surel maupun Google Form.

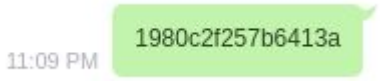
Dalam melakukan verifikasi untuk berkas berikutnya, dilakukan menggunakan tahapan berikut:

1) Mahasiswa atau pemilik berkas membuat nilai *hash* dari berkas yang dimiliki menggunakan jenis fungsi yang sama
2) Nilai *hash* tersebut dibandingkan dengan nilai *hash* yang ada sebelumnya

Sebagai contoh cara kerja dari solusi ini, penulis melakukan *hash* dari contoh berkas yang dia miliki. Menggunakan kakas *sha3sum* yang tersedia di *platform* yang digunakan penulis yaitu Ubuntu dalam *package libdigest-sha3-perl*, dipanggil suatu fungsi `sha3sum 13517004_TA1_Laporan.pdf`. Fungsi ini mengeluarkan hasil seperti yang terlihat di gambar.

Dari hasil *hash* yang didapat, mahasiswa dapat menyimpan nilai *hash* ini (atau bahkan beberapa karakter pertamanya saja) ke dalam saluran dengan sumber waktu yang terpercaya seperti aplikasi *chat*. Sebagai contoh oleh penulis dimasukkan ke

dalam aplikasi LINE seperti gambar di bawah.



Di sini, proses penjaminan ini sudah selesai dengan waktu 11.09 PM. Nantinya nilai *hash* ini dapat dibandingkan dengan nilai *hash* berkas di waktu berikutnya menggunakan proses yang sama.

IV. Analisis Solusi

Teknik ini dapat digunakan untuk berbagai jenis berkas digital yang mungkin, baik dari dokumen biasa sampai repositori kode. Aplikasi yang dapat membuat fungsi *hash* tersedia di berbagai *platform*, dari Windows, Linux, bahkan di *web*. Dalam mengumpulkan nilai *hash* ini seperti yang telah disebutkan di bagian sebelumnya, berbagai cara ini memiliki tingkat ketersediaan yang tinggi yang dijamin oleh perusahaan-perusahaan berskala multinasional, cukup mudah dan sudah marak digunakan, dan mudah untuk memverifikasi *timestamp* dari nilai *hash* yang dimasukkan dan sangat sulit untuk dimanipulasi oleh pihak mahasiswa yang tidak memiliki akses ke server dan basis data yang dipegang oleh perusahaan tersebut. Dengan cara ini, dapat dibuktikan bahwa akan sangat sulit untuk memanipulasi berkas awal setelah batas waktu yang diberikan. Selain aman dan terpercaya, cara ini tergolong mudah untuk dilakukan dan serba guna untuk dilakukan di berbagai kasus.

A. Analisis Kecepatan

Untuk membuktikan bahwa cara ini relatif lebih cepat apabila dibandingkan dengan mengunggah berkas tersebut secara langsung, penulis membuat suatu kode sederhana yang membangkitkan suatu berkas dengan ukuran sembarang, lalu dihitung waktu yang dibutuhkan untuk mendapatkan nilai *hash*nya. Kode sederhana yang dibuat menggunakan bahasa C++ dengan menghasilkan suatu barisan huruf yang dibuat dengan panjang yang dapat ditentukan sehingga berkas yang menyimpan barisan huruf tersebut memiliki ukuran yang dapat ditentukan juga. Berikut merupakan sumber kode yang penulis gunakan:

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    long long n;
    cin >> n;
    for (long long i = 0; i + 1 < n; ++i) {
        cout << 'a';
    }
    cout << '\n';
}
```

Menggunakan kode ini, dibangkitkan berbagai berkas dengan berbagai ukuran, lalu berkas itu dihitung nilai *hash*nya menggunakan kakas *sha3sum*. Waktu perhitungan ini lalu di*benchmark* untuk menunjukkan performa dari fungsi *hash* menggunakan fungsi *time* yang tersedia di Ubuntu. Walaupun *benchmark* ini tentunya bergantung pada *hardware* yang dimiliki penulis, hasil waktunya dianggap dapat menjadi gambaran untuk menunjukkan keefisienan dari cara ini.

Berikut tabel yang menunjukkan hasil *benchmark* tersebut:

Ukuran berkas	Waktu fungsi *hash* (s)
1 KB	0,043
10 KB	0,036
100 KB	0,036
1 MB	0,046
10 MB	0,084
100 MB	0,424
1 GB	4,206

Dapat dilihat bahwa waktu fungsi *hash* sangatlah cepat dan efisien dimana memproses ukuran berkas yang cukup kecil bahkan dikalahkan dari waktu *overhead* yang dibutuhkan untuk memanggil fungsi tersebut.

Jika dibandingkan dengan waktu pengunggahan suatu berkas ke internet, mengasumsikan kecepatan unggah konstan sebesar 10 Mbps yang relatif masuk akal untuk ukuran kecepatan internet Indonesia, didapat waktu yang dibutuhkan dengan ukuran berkas yang sama dengan yang di atas seperti berikut:

Ukuran berkas	Waktu pengunggahan (s)
1 KB	0,0008
10 KB	0,008
100 KB	0,08
1 MB	0,8
10 MB	8
100 MB	83
1 GB	858

Dapat dilihat bahwa kecepatan untuk mengunggah berkas berukuran cukup besar setaraf lebih lambat daripada melakukan fungsi *hash* dan mengunggah fungsi *hash* tersebut.

Mempertimbangkan kemungkinan situs pengumpulan seperti Gitlab Informatika mengalami gangguan jaringan saat dekat batas waktu pengumpulan, teknik ini ditinjau lebih cepat dan dapat diandalkan daripada mengunggah keseluruhan berkas ke internet.

B. Perbandingan dengan *Timestamp Authority*

Dari beberapa pihak yang menyediakan jasa *timestamp authority* secara gratis, seperti freeTSA.org, menyediakan klien berbasis TCP yang dapat ditembak oleh para penggunanya. Diantara dari tahapan yang harus dilakukan saat meminta *timestamp* adalah:

1) Membuat berkas *tsq* (*TimeStampRequest*) yang mengandung *hash* dari berkas yang ingin ditandatangani. Perintah di linux untuk melakukan hal ini adalah `openssl ts -query -data file.png -no_nonce -sha512 -cert -out file.tsq`
2) Mengirimkan berkas tersebut ke freeTSA.org untuk mendapat berkas *tsr* (*TimeStampResponse*) yang menunjukkan bahwa berkas itu tersedia pada waktu dimana permintaan dibuat. Perintah di linux untuk melakukan hal ini adalah `curl -H "Content-Type: application/timestamp-query" --data-binary '@file.tsq' https://freetsa.org/tsr > file.tsr`

Setelah prosedur ini dilakukan, *timestamp* dapat diverifikasi menggunakan berkas *tsq* dan *tsr*. Perintah di linux untuk melakukan hal ini adalah `openssl ts -verify -in file.tsr -queryfile file.tsq -CAfile cacert.pem -untrusted tsa.crt`

Selain menggunakan klien TCP, freeTSA.org juga menyediakan cara lain seperti klien di situsnya yang dapat digunakan dengan mengunggah berkas terkait untuk mengunduh *tsq* dan *tsr* terkait.

Dari prosedur yang telah digunakan, ada beberapa kekurangan yang dapat ditunjukkan dibandingkan dengan metode yang telah disebutkan di makalah ini. Pertama, dibutuhkan *tools* tambahan seperti *openssl* yang dapat menambah pekerjaan yang harus dilakukan dan mungkin tidak jelas pasangannya di sistem operasi lain seperti Windows; kedua, menggunakan klien di situsnya sama saja dengan mengunggah keseluruhan berkas ke internet, sehingga tidak menawarkan kecepatan tambahan seperti melakukan fungsi *hash* secara lokal; ketiga, penggunaan *timestamp authority* mengandalkan keberadaan jasa pihak ketiga yang lebih sulit dibuktikan integritasnya dan mungkin tidak tersedia sewaktu-waktu dibandingkan dengan proses pengumpulan nilai *hash* yang telah dijelaskan di bagian sebelumnya.

Oleh karena itu, dapat dilihat bahwa penggunaan *timestamp authority* relatif lebih sulit, rumit, dan tidak memberikan jaminan tambahan terhadap integritas berkas yang diberikan.

C. Perbandingan dengan Pengecekan *Git Log*

Beberapa usaha sebelumnya untuk mengecek integritas berkas diantara waktu pengumpulan adalah dengan mengecek luaran dari fungsi *git status* dan fungsi *git log*. Fungsi *git status* berfungsi untuk menunjukkan apakah ada perubahan yang tersimpan dari berkas yang dilacak oleh *git* dan juga apakah ada perbedaan dengan repositori *remote*. Fungsi *git log* berfungsi untuk menunjukkan daftar *commit* yang ada, termasuk waktu dilakukan *commit* tersebut. Apabila ada perubahan yang dilakukan setelah waktu pengumpulan, dianggap dapat dideteksi dengan melihat apakah ada *commit* tambahan yang dilakukan setelah waktu pengumpulan maupun perubahan di berkas yang tidak di*commit* menggunakan fungsi *git log*.

Kekurangan dari pendekatan ini adalah bahwa mahasiswa sebagai pembuat repositori dapat membuang suatu *commit* maupun memanipulasi suatu *commit* yang ada dengan mengubah konten perubahan di *commit* tersebut, bahkan mengganti *timestamp* dari suatu *commit* yang ada. Walaupun perubahan tersebut dilakukan secara lokal, perubahan ini juga dapat dibawa ke repositori *remote* dengan mem*push* perubahan yang dilakukan ini. Hal ini sebenarnya dapat dideteksi apabila seseorang memiliki akses admin dalam situs repositori tersebut, tetapi hal ini tidak dapat dilakukan apabila menggunakan situs pihak ketiga seperti Github dan Gitlab, juga membutuhkan usaha tambahan untuk melakukan pendeteksian ini.

Melihat masalah tersebut, pihak yang menerima tugas haruslah awas dalam mengecek *commit* maupun isi repositori yang ada di repositori *remote* tepat pada waktu pengumpulan. Oleh karena itu, pengecekan menggunakan fungsi *git* dianggap tidak cukup aman untuk menjamin integritas data tanpa membutuhkan kemampuan dan usaha tambahan.

V. Simpulan dan Saran

Penjaminan integritas dalam pengumpulan tugas daring menggunakan fungsi *hash* dilakukan dengan mengambil nilai *hash* dari berkas terkait dan mengumpulkannya melalui saluran dengan sumber waktu terpercaya seperti aplikasi *chat*. Cara ini ditinjau lebih serba guna, mudah, dan terjamin saat dibandingkan dengan alternatif yang sudah ada seperti *timestamp authority* maupun menggunakan fungsi *git* di repositori tugas.

VI. Ucapan Terima Kasih

Penulis ingin mengucapkan terima kasih kepada Bapak Dr. Ir. Rinaldi Munir, MT. sebagai dosen pengajar mata kuliah Kriptografi 2020/2021 yang telah mengajarkan penulis berbagai ilmu-ilmu penting yang penulis butuhkan untuk mengerti teori kriptografi modern. Penulis juga ingin mengucapkan terima kasih kepada teman-teman penulis yang telah membantu penulis dalam mengerjakan makalah ini, terutama teman saya Aidil Rezjski, Rayza Mahendra, Rakhmad Budiono, Ricky Yuliawan, Ahmad Rizal Alifio dan Vinsen Marselino yang telah memberi saya inspirasi dalam mengerjakan makalah ini.

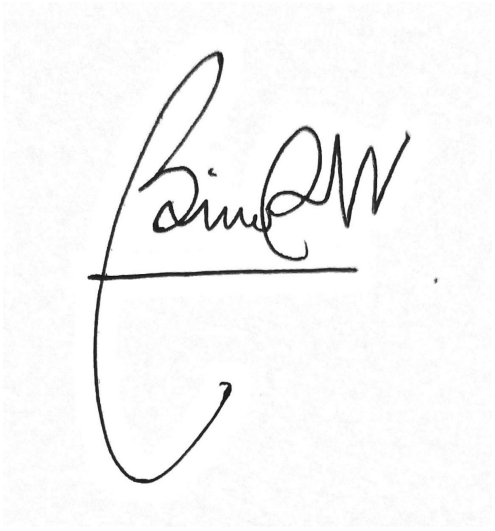
Referensi

[1] Munir, R. (2020) *Bahan Kuliah IF4020 Kriptografi*.
[2] Public Key Infrastructure Time-Stamp Protocol, dilansir dari https://www.ietf.org/rfc/rfc3161.txt
[3] Basic Git commands, dilansir dari https://confluence.atlassian.com/bitbucketserver/basic-git-commands-776639767.html

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, 21 Desember 2020



Bimo Adityarahman Wiraputra 13517004