Penggunaan Merkle Tree pada Incremental Backup untuk Mendeteksi Perubahan Berkas

Muhammad Reza Ramadhan
School of Electrical Engineering and Informatics
Bandung Institute of Technology
Bandung, Indonesia
rezaramadhan.m@gmail.com

***Abstract*—Incremental backup adalah sebuah proses yang penting untuk dilakukan secara berkala. Salah satu alternatif yang dapat digunakan untuk melakukan incremental backup adalah dengan menggunakan merkle tree sebagai cara mendeteksi perubahan berkas. Struktur merkle tree pada incremental backup dibuat mirip dengan stuktur directory tree dengan tambahan nilai hash di setiap simpulnya. Kakas incremental backup yang dibuat sudah berhasil menyelesaikan pekerjaan dengan relatif cepat walaupun kinerjanya masih tidak lebih baik dibandingkan dengan kakas yang sudah ada seperti cp dan rsync.**

Keywords—Merkle Tree, incremental backup, tree, hash function, SHA-1, cryptography

I. PENDAHULUAN

Penyimpanan data sudah menjadi suatu kebutuhan umum pada masa digital dewasa ini. Data yang disimpan tidak hanya data penting seperti dokumen pekerjaan, buku, kumpulan paper, ataupun juga database namun juga berbagai data personal seperti koleksi foto dan video pribadi, musik, ataupun koleksi media lainnya. Namun, dengan rata-rata 2% hard disk mengalami kegagalan per tahun[1], kehilangan data yang hanya disimpan dalam satu buah media penyimpanan data bukanlah hal yang mustahil.

Penggunaan cloud storage merupakan salah satu solusi terbaik sebagai media backup berbagai berkas yang kita miliki. Namun, dengan kecepatan internet di Indonesia yang hanya memiliki rata-rata sebesar 13.79 Mbps (~1.7MB/s)[2] maka proses backup akan memakan waktu yang tidak sebentar. Untuk masyarakat Indonesia, salah satu solusi terbaik untuk melakukan backup masih dengan menggunakan sebuah media penyimpanan cadangan.

Berbagai perangkat lunak telah menyediakan layanan backup antar media penyimpanan dengan sangat baik. Penggunaan rsync pada sistem operasi Linux serta berbagai aplikasi pihak ketiga seperti Backup4All pada Windows adalah salah satu alternatif terbaik dalam melakukan backup dengan adanya kemampuan untuk melakukan backup secara incremental. Dengan penggunaan backup secara incremental, tidak seluruh berkas akan disalin ke media backup. Incremental backup hanya akan menyalin berkas yang telah dimodifikasi serta berkas baru jika dilihat dari proses backup sebelumnya. Dengan demikian, incremental backup akan memakan waktu yang lebih singkat dibandingkan proses backup konvensional.

Berbagai algoritma telah ada dan dapat digunakan sebagai sarana pendeteksian perubahan berkas seperti Common Prefix/Suffix, Singular Insertion/Deletion, Two Edits, ataupun algoritma diff Myers. Salah satu cara sederhana yang dapat digunakan untuk mendeteksi apakah suatu berkas sudah berubah adalah dengan menggunakan fungsi hash. Kita hanya perlu membandingkan dua hash dari sebuah berkas dari dua periode waktu untuk dapat menentukan apakah berkas telah berubah dalam periode waktu tersebut.

Fungsi hash hanya bisa digunakan untuk mendeteksi perubahan dari sebuah berkas, pendeteksian perubahan dari sebuah isi direktori tidak dapat dilakukan dengan hash secara langsung mengingat pada berbagai filesystem direktori hanyalah metadata mengenai lokasi dari file. Untuk mengetahui perubahan dari sebuah folder dengan fungsi hash, hal yang mungkin dilakukan adalah menjadikan seluruh berkas menjadi masukan dari hash untuk direktori, atau menjadikan hash dari setiap berkas sebagai masukan hash untuk direktori itu. Merkle Tree adalah sebuah struktur data yang digunakan untuk menyimpan data hash dimana sebuah hash merupakan nilai hash dari penggabungan beberapa nilai hash lain. Penggunaan merkle tree akan sangat cocok untuk digunakan dalam proses incremental backup ini.

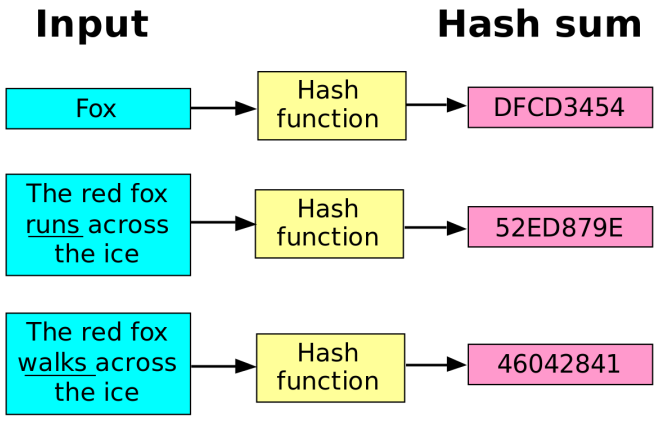
II. STUDI LITERATUR

A. *Fungsi Hash*

Fungsi hash adalah fungsi yang menerima masukan berupa string dengan panjang sembarang lalu mentransformasikannya

menjadi sebuah string dengan ukuran tertentu[3]. Secara matematis fungsi hash umum digambarkan dalam persamaan:

$$h = H(x)$$



Gambar 1. Ilustrasi Fungsi Hash

Fungsi hash kriptografis adalah sebuah fungsi hash yang didesain secara matematis untuk hanya bisa dijalankan dalam satu arah dan tidak bisa dicari inversnya[4]. Fungsi hash kriptografis sering disebut juga fungsi hash satu arah. Fungsi hash kriptografis memiliki beberapa sifat tertentu[4] seperti:

1. Hash dapat digunakan untuk input berukuran berapapun.
2. Hash menghasilkan output dengan panjang tetap. Output fungsi hash sering disebut *digest*.
3. Perhitungan *H(x)* bukan merupakan komputasi yang sulit.
4. Untuk setiap digest yang dihasilkan, tidak mungkin ditemukan *x* dimana *H(x) = digest.*
5. Untuk setiap nilai *x*, tidak mungkin mencari *y != x* dimana *H(y) = H(x)*
6. Tidak mungkin terdapat pasangan *x* dan *y* dengan *x != y* dan *H(y) = H(x)*

Fungsi hash dapat digunakan untuk berbagai macam kepentingan seperti menjaga integritas data, menghemat waktu verifikasi data, mempermudah pengiriman data yang ingin diverifikasi, serta untuk menormalkan panjang data[3]. Sifat hash dimana dua input berbeda tidak mungkin menghasilkan output hash yang sama merupakan sifat yang dapat digunakan untuk menguji apakah sebuah berkas telah dimodifkasi.

B. Secure Hash Algorithm (SHA)

Secure Hash Algorithm (SHA) adalah salah satu keluarga fungsi hash kriptografis populer yang saat ini sering digunakan dalam berbagai perangkat lunak. SHA memiliki beberapa varian dimana setiap varian memiliki jumlah byte digest dan juga algoritma yang berbeda.

Varian SHA-0, SHA-1, SHA-2, dan SHA-3 memiliki tingkat keamanan yang berbeda. Collision pada SHA-0 dan SHA-1 sudah berhasil ditemukan dengan masing-masing didapatkan fungsi memiliki collision kurang dari 34 dan 63 bit.

Waktu eksekusi dari masing-masing varian secara umum meningkat pada setiap iterasi versi SHA. Bersama dengan penambahan waktu eksekusi, kemanan hash dari collision juga bertambah pada setiap iterasi. Secara sederhana SHA-3 adalah varian paling lambat namun paling aman, SHA-2 lebih cepat dari SHA-3 namun lebih tidak aman, dan seterusnya.

Perbandingan terhadap spesifikasi dari beberapa varian SHA dapat terlihat pada tabel 1.

Tabel 1. Perbandingan Spesifikasi Varian SHA

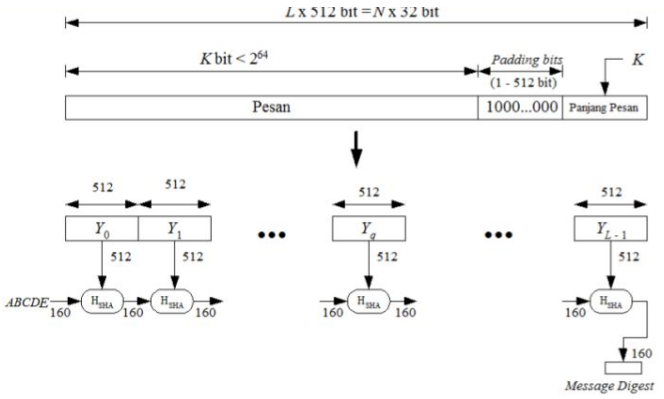
SHA Variant	Digest size (bits)	Block Size (bits)	Security Bits	Perfor-mance (cpb/ 8 bytes)
SHA-0	160	512	<34	~52.00
SHA-1	160	512	<63	52.00
SHA-256	256	512	128	85.25
SHA-512	512	1024	256	135.50
SHA3-256	256	1088	128	155.50
SHA3-512	512	576	256	164.00

cpb adalah cycle per bytes, yaitu berapa jumlah cycle CPU rata-rata yang dibutuhkan sebuah algoritma untuk memproses satu byte data, algoritma yang memiliki cpb lebih besar akan memiliki waktu eksekusi lebih kecil.

C. SHA-1

SHA-1 adalah fungsi hash kriptografis yang dikembangkan oleh National Institute of Standards and Technology (NIST). SHA-1 dibuat berdasarkan pada fungsi hash MD4 yang dibuat oleh R.L. Rivest. SHA-1 memiliki ukuran blok sebesar 160, maksimum input sebesar 2^{64} bit dan menghasilkan digest sepanjang 160 bit.

Pembuatan digest pada SHA-1 dilakukan dengan beberapa proses. Secara sederhana, proses tersebut diantaranya adalah penambahan padding bits dan nilai panjang pesan, inisialisasi buffer vector sebagai cikal bakal message digest, serta pengolahan input data secara bertahap dalam blok sebesar 512 bit.

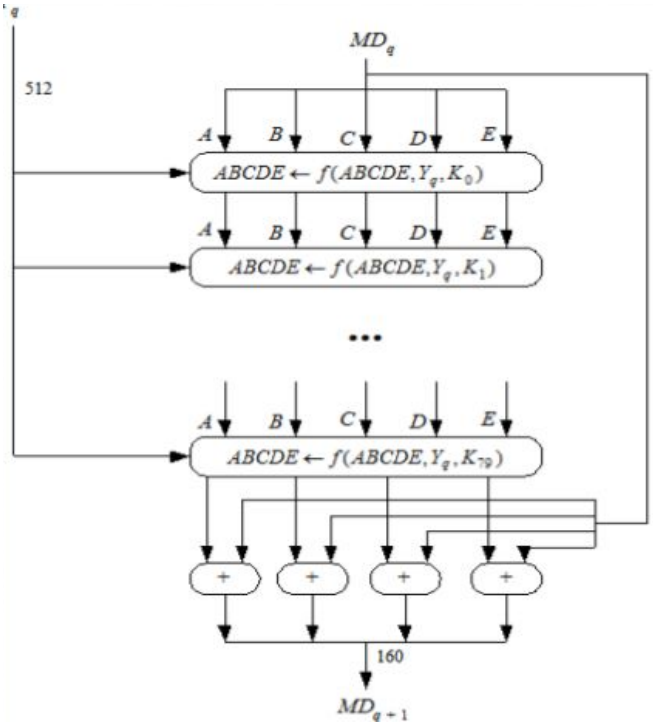


Gambar 2. Skema Pembuatan Digest pada SHA-1[3]

Pengolahan input data secara bertahap dapat digambarkan dengan persamaan matematis sebagai berikut:

$$MD_n = H_{SHA}(Y_n, MD_{n-1})$$

Y adalah blok data yang saat itu sedang diproses, MD adalah message digest untuk blok data saat itu, serta H_{SHA} adalah fungsi SHA dalam sebuah blok. H_{SHA} memiliki skema sebagai berikut:



Gambar 3. Skema H_{SHA}[3]

Collision pada SHA-1 sudah ditemukan pada 23 Februari 2017 oleh tim dari Google dan CWI Amsterdam. Tim tersebut berhasil menemukan collision dari dua pdf yang berbeda. Effort yang diperlukan untuk membuat collision tersebut memang cukup besar, yaitu sekitar 9500 tahun komputasi CPU dan 110 tahun komputasi GPU. Namun dengan banyaknya komputer yang saat ini terkoneksi dengan internet maka bukan SHAttered attack sudah menjadi attack yang practical. Menurut tim tersebut, penggunaan metode SHAttered untuk menemukan SHA 100.000 kali lipat lebih cepat dibandingkan brute force.

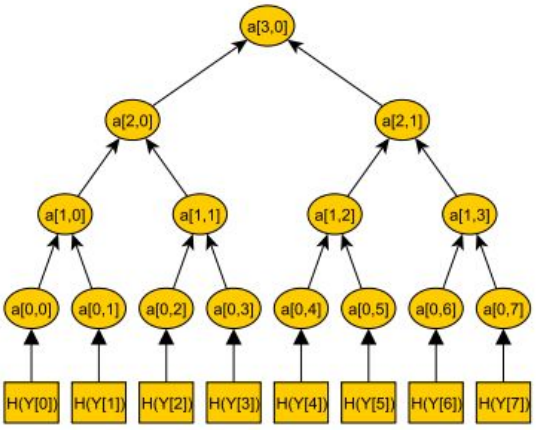
D. Merkle Tree

Merkle tree adalah sebuah struktur pohon dimana setiap daun dari pohon berisi nilai hash dari sebuah blok data dan simpul lainnya berisi nilai hash dari anak-anaknya. Merkle tree pada umumnya diimplementasikan sebagai sebuah pohon biner, namun konsep awal merkle tree tidak menjelaskan secara spesifik mengenai jumlah anak yang mungkin dimiliki sebuah simpul.

Pada merkle tree, proses verifikasi dari sekelompok blok data dapat dilakukan dengan membandingkan hash yang dimiliki akar. Kelebihan merkle tree dari hash biasa adalah merkle tree dapat melakukan verifikasi terhadap bagian tertentu dari sekelompok blok data. Dengan kemampuan verifikasi terhadap sekelompok blok data, maka kita juga dapat menggunakan merkle tree untuk mendeteksi blok data yang telah berubah.

Jika kita mengacu pada Gambar 4 sebagai contoh, perubahan blok data ***H(Y\|6)*** dan ***H(Y\|7)*** akan mempengaruhi nilai hash pada simpul ***a[0,6]***, ***a[0,7]***, ***a[1,3,]***, ***a[2,1]*** serta ***a[3,0]***. Karena kita tahu bahwa nilai hash pada simpul yang berubah hanyalah pada simpul tertentu di bagian 'kanan' pohon, maka kita juga dapat mengetahui bagian mana dari sekelompok blok data yang telah mengalami perubahan.

Saat ini, salah satu kegunaan merkle tree digunakan untuk verifikasi setiap data transaksi pada cryptocurrency berbasis blockchain seperti Bitcoin, Bitcoin Cash, atau Ethereum. Beberapa filesystem menggunakan merkle tree untuk menghindari kerusakan data seperti ZFS, IPFS, dan Btrfs. Beberapa database NoSQL terdistribusi seperti Cassandra dan Riak menggunakan merkle tree untuk melakukan replikasi data sehingga database dapat mencapai eventual consistency.



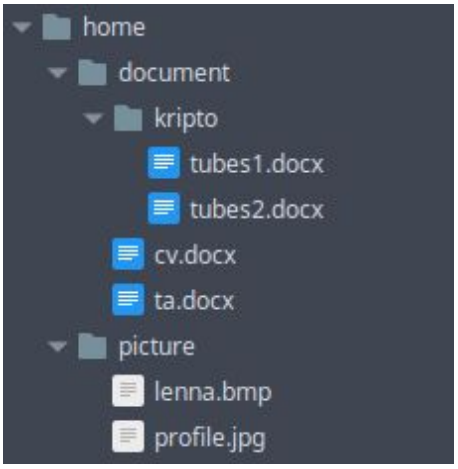
Gambar 4. Merkle Tree[5]

III. Rancangan Implementasi

Kakas **mtreebackup** yang dibuat dalam bahasa Python merupakan implementasi dari penggunaan merkle tree dalam incremental backup. Kakas ini akan menerima dua buah input yaitu sebuah direktori sumber backup serta direktori lain yang akan menjadi direktori tujuan backup. Kakas mtreebackup akan membuat merkle tree untuk masing-masing direktori sumber backup dan direktori tujuan backup. Backup dilakukan dengan membandingkan kedua merkle tree dan menyalin file yang berbeda dari direktori sumber backup ke direktori tujuan backup berdasarkan deteksi menggunakan merkle tree.

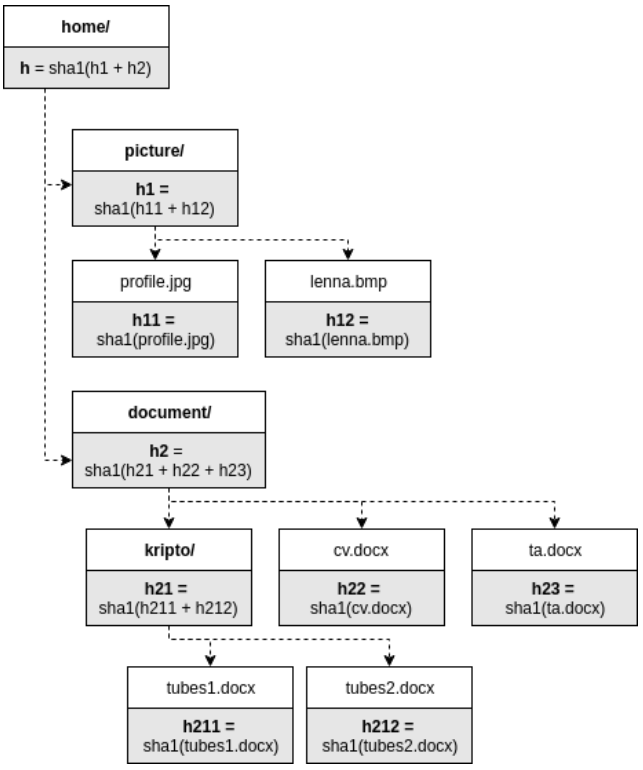
A. Struktur Merkle Tree

Pada mtreebackup, kakas dapat mendeteksi berkas mana dari file pada sumber backup dan tujuan backup yang telah berubah. Merkle tree yang digunakan dalam kasus ini adalah sebuah merkle tree dimana sebuah berkas adalah simpul daun dari merkle tree serta direktori tempat berkas tersebut berada adalah simpul parent dari daun tersebut. Secara sederhana, merkle tree akan berbentuk sama directory tree dan setiap simpul akan menyimpan informasi hash yang merepresentasikan kondisi berkas saat itu.



Gambar 5. Contoh Struktur Direktori Backup

Sebagai contoh, sebuah sumber direktori backup seperti pada Gambar 5 akan memiliki struktur merkle tree seperti pada Gambar 6.



Gambar 6. Struktur Merkle Tree

Fungsi hash yang digunakan pada merkle tree ini adalah SHA1. SHA1 digunakan karena SHA1 memiliki perbandingan antara security dan kecepatan yang cocok dalam kasus incremental backup. Penggunaan fungsi hash yang memiliki waktu komputasi lama akan mengurangi kinerja dari kakas ini.

SHA0 memiliki cps yang relatif sama dengan SHA1, namun memiliki security bits yang lebih kecil. Sementara itu, SHA2 dan SHA3 memiliki security bits yang lebih besar namun cps yang juga besar. Pada kasus incremental backup ini tidak dibutuhkan security bits > 100 mengingat kemungkinan $1/10^{30}$ terjadinya collision sudah sangat kecil. Dengan demikian, SHA1 tepat berada pada titik tengah diantara kecepatan dan juga keamanan.

B. Proses Pembangkitan Merkle Tree

Merkle Tree dibangkitkan bersamaan dengan proses iterasi pada sebuah direktori untuk mendapatkan list file dan direktori. Awalnya dibuat sebuah simpul akar yang merepresentasikan sumber direktori backup. Untuk setiap file dan sub-direktori yang ada di dalam sebuah direktori, akan dibuat simpul anak yang merepresentasikan masing-masing file dan direktori tersebut. Untuk setiap sub-direktori yang ada, maka akan dilakukan proses pembangkitan merkle tree lagi secara rekursif.

Nilai hash file akan dihitung ketika simpul yang merepresentasikan file tersebut dibuat pada merkle tree. Sementara itu, nilai hash untuk setiap direktori akan dihitung

ketika merkle tree sudah selesai dibuat. Nilai hash pada sebuah simpul direktori akan menggunakan nilai hash simpul anak-anaknya sebagai input dari SHA1.

Pada mtreebackup, merkle tree diimplementasikan dengan memanfaatkan struktur data pohon dari library anytree. Library ini telah menyediakan proses pembuatan simpul, pembuatan anak simpul, penggambaran pohon, pemberian atribut pada setiap simpul, iterasi pohon, serta pencarian node pada pohon ynag telah dibuat.

C. Proses Incremental Backup

Setelah merkle tree untuk direktori sumber dan tujuan backup dibuat, maka proses incremental backup dimulai. Proses ini akan membandingkan kedua merkle tree dan hanya akan melakukan penyalinan pada berkas atau direktori jika nilai hash di merkle tree sumber backup tidak sama dengan nilai hash pada merkle tree hasil backup.

Proses perbandingan dan penyalinan tersebut dapat dilakukan dengan menggunakan Pseudocode 1.

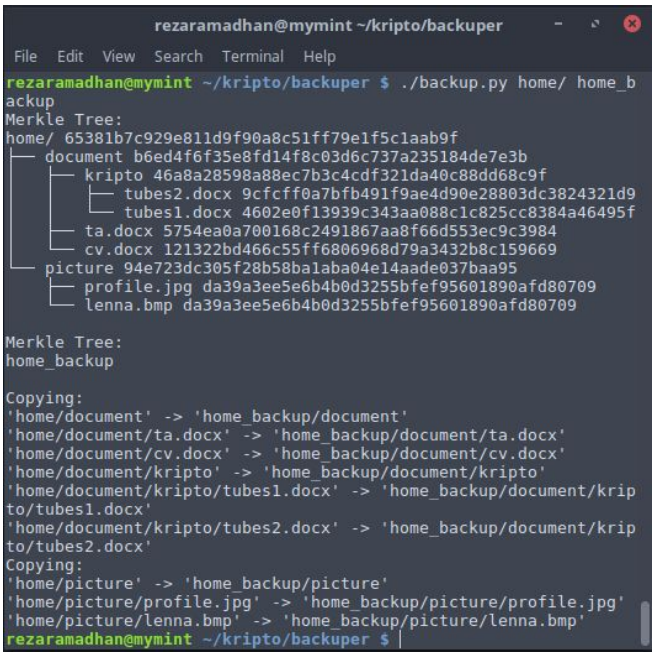
Pseudocode 1. Compare Tree

```
procedure compare_tree(src_root, dest_root)

if src_root.hash = dest_root.hash:
   do nothing
else
   foreach children of src_root
            as src_child:
      if src_child is_not_child_of dst_root:
          backup(srcchild)
      else:
          if is_file(src_child):
              backup(src_child)
          else:
              compare_tree(src_child,
                           dest_child)

# delete old file/directory
# in backup destination
   foreach children of dest_root as child:
      if dst_child is_not_child_of src_root:
            delete_from_disk(dst_child)
```

Pada implementasi ini, **delete_from_disk** dan **backup** diimplementasikan dengan melakukan pemanggilan command pada command line sistem operasi. Pada **delete_from_disk** akan dilakukan pemanggilan command *rm -rf*, sementara itu **backup** akan melakukan pemanggilan command *cp -v*.

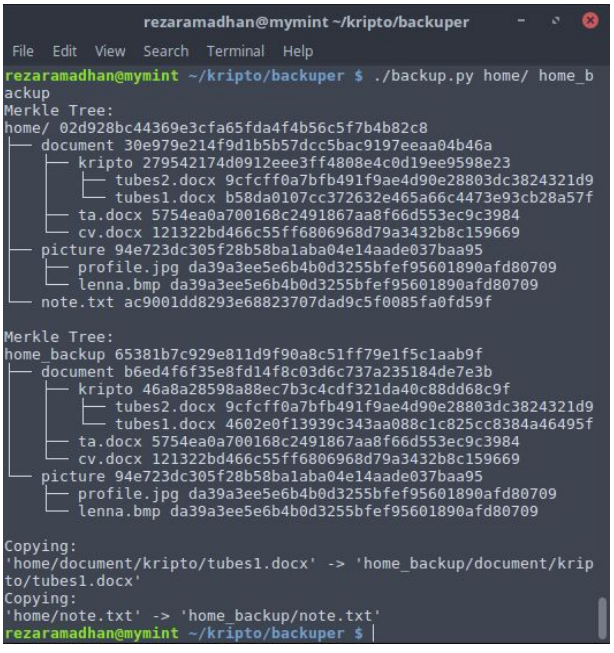
IV. HASIL IMPLEMENTASI

Berikut merupakan hasil penjalanan program untuk melakukan backup untuk direktori yang ada pada Gambar 3 untuk pertama kali.



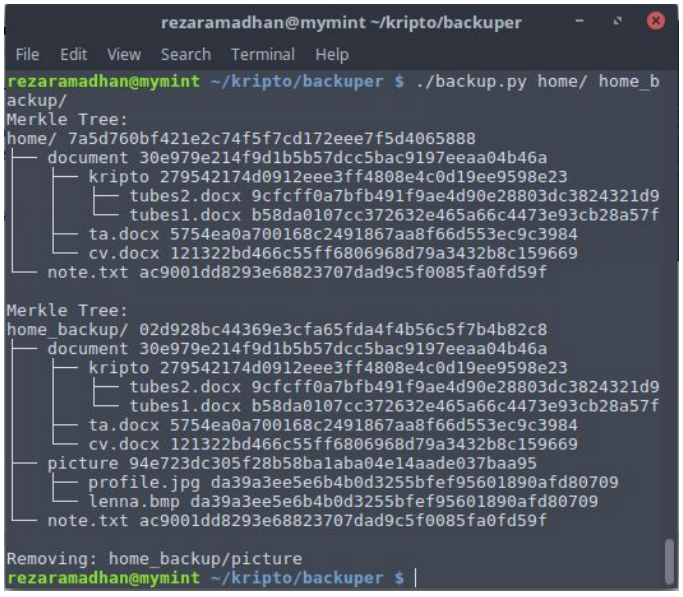
Gambar 7. Tampilan Program mtreebackup 1

Selain itu, berikut adalah hasil program untuk melakukan backup ketika pada direktori awal telah ditambahkan file **note.txt** dan isi dari file **tubes2.docx** telah berubah.



Gambar 8. Tampilan Program mtreebackup 2

Berikut adalah tampilan program ketika backup dijalankan setelah direktori **picture** dihapus dari sumber backup.



Gambar 9. Tampilan Program mtreebackup 3

V. PENGUJIAN

Pengujian yang akan dilakukan pada tahap ini adalah perbandingan mtreebackup dengan kakas lain yang dapat digunakan untuk melakukan backup. Kakas lain yang digunakan adalah **cp** untuk melakukan full backup terhadap sebuah direktori serta **rsync** untuk melakukan incremental backup terhadap sebuah directory.

Command **cp** yang dipanggil adalah **cp -rv,** dengan tujuan mengetahui file mana saja yang disalin oleh kakas tersebut dan juga untuk melakukan penyalinan pada direktori secara rekursif. Demikian juga dengan rsync yang melakukan pemanggilan command **rsync -v**, untuk menampilkan file mana saja yang disalin.

Pengujian akan dilakukan terhadap beberapa kasus uji, diantaranya adalah:

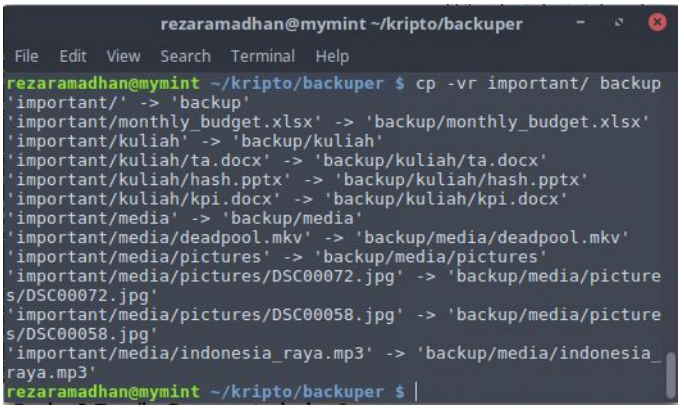
- Backup 0, dimana direktori tujuan backup masih kosong.
- Backup 1, dimana sedikit bagian dari berkas telah berubah dan beberapa berkas telah dihapus
- Backup 2, dimana sebagian besar berkas dalam direktori telah berubah.

Setiap kasus uji tersebut akan dilakukan kepada tiga direktori yang masing-masing memiliki jumlah file yang sedikit, sedang, dan juga banyak. Masing-masing tiga direktori tersebut akan memiliki jumlah dan ukuran berkas sesuai dengan yang disampaikan pada Tabel 2.

Tabel 2. Total Ukuran dan Jumlah Berkas pada Tiap Tipe Direktori

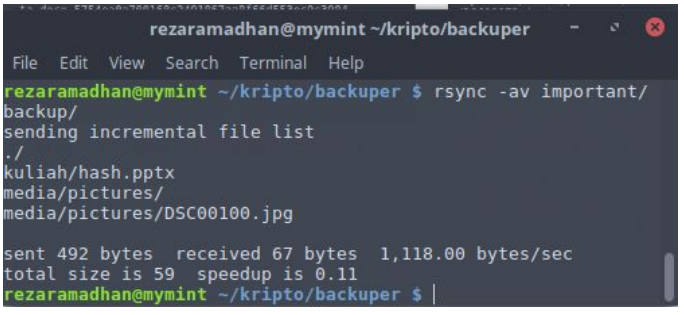
Tipe Ukuran Direktori	Total Jumlah Berkas	Total Ukuran Berkas
Kecil	13	323.7 KB
Sedang	157	1.3 MB
Besar	239	4.4 MB

Berikut adalah contoh tampilan kakas **cp** ketika melakukan backup kosong pada direktori dengan sedikit file:



Gambar 10. Tampilan Program cp

Berikut adalah contoh tampilan kakas **rsync** ketika melakukan backup 1 pada direktori dengan sedikit file:



Gambar 11. Tampilan Program rsync

Pengukuran waktu dilakukan dengan kakas **time** yang ada pada sistem operasi Linux. Data yang diambil dari kakas tersebut adalah real time dengan satuan waktu milisecond.

Pengujian dilakukan masing-masing sebesar tiga kali dan akan diambil data rata-rata dengan pembulatan kebawah. Berikut adalah data hasil proses pengujian yang telah dilakukan:

Tabel 2. Hasil Pengujian terhadap mtreebackup

Tipe Backup	Ukuran sumber direktori backup	Waktu Eksekusi (ms)		
		mtree backup	cp	rsync
Backup 0	Kecil	67	6	44
Backup 1		95	5	43
Backup 2		84	4	46
Backup 0	Sedang	122	21	72
Backup 1		189	22	68
Backup 2		177	19	70
Backup 0	Besar	149	35	100
Backup 1		228	37	87
Backup 2		210	32	93

Berdasarkan data yang terdapat pada tabel diatas, terlihat bahwa mtreebackup merupakan kakas dengan waktu eksekusi paling besar jika dibandingkan dengan cp dan rsync. Full backup dengan cp memiliki waktu yang relatif konstan antara berbagai tipe backup yang dilakukan, mengingat cp tidak peduli terhadap status perubahan dari direktori sumber backup ke direktori tujuan backup. Sementara itu incremental backup dengan rsync memiliki waktu yang bergantung pada status perbedaan isi antara direktori sumber backup dan direktori tujuan backup, dengan rata-rata backup 1 dan backup 0 akan memiliki waktu eksekusi yang lebih kecil daripada backup 0.

Kakas mtreebackup membutuhkan waktu hingga kurang lebih sekitar satu setengah kali lipat saat melakukan backup 1 dibandingkan dengan waktu eksekusi pada backup 0. Hal ini karena apada backup 0 merkle tree pada tujuan backup masih kosong dan tidak perlu dibuat secara kompleks. Pada backup 1, merkle tree pada tujuan backup sudah terisi dan perlu untuk dihitung nilai hash untuk masing-masing simpulnya. Pada backup 1, terjadi pula perbandingan pada dua merkle tree. Walaupun memang tidak signifikan, proses perbandingan tersebut tentu membutuhkan waktu eksekusi juga.

Penyebab penggunaan kakas mtreebackup memiliki waktu eksekusi yang lebih lama daripada cp dan rsync diantaranya adalah kakas perlu untuk membangun merkle tree untuk masing-masing direktori sumber dan tujuan backup. Proses perhitungan nilai SHA1 untuk setiap file memerlukan waktu yang cukup signifikan, apalagi mengingat kakas ini tidak memanfaatkan multithreading untuk memproses nilai hash setiap berkas seacara paralel.

Penggunaan Python sebagai bahasa pemrograman juga berpengaruh pada waktu eksekusi. Kakas cp dan rsync masing-masing dibuat dalam bahasa C dan telah mengalami berbagai peningkatan kecepatan selama bertahun-tahun. Python merupakan bahasa tingkat tinggi yang mengorbankan waktu eksekusi dengan kemudahan pembuatan sebuah program. Hal ini sangat terlihat karena kakas mtreebackup dapat selesai dibuat hanya dalam waktu satu setengah hari namun memiliki waktu eksekusi yang lebih lama dibandingkan dengan cp ataupun rsync.

VI. Kesimpulan dan Saran Pengembangan Selanjutnya

Berdasarkan kakas yang telah dibuat, terlihat bahwa merkle tree merupakan salah satu struktur data yang dapat digunakan untuk membantu proses incremental backup. Merkle tree dapat mendeteksi perubahan file dan isi sebuah direktori. Pada kakas yang dibuat, penyalinan berkas dan direktori hanya dilakukan pada berkas dan direktori yang telah berubah sejak proses backup terakhir. Penggunaan incremental backup dapat mempercepat proses backup dan juga menghemat bandwith juga backup tersebut dilakukan melalui jaringan komputer.

Kakas yang dibuat belum sempurna dan masih memerlukan perbaikan pada beberapa sisi. Salah satu penyempurnaan yang mungkin dilakukan adalah menyimpan merkle tree dalam file eksternal tertentu pada direktori hasil backup. Merkle tree tidak memerlukan kapasitas penyimpanan yang relatif besar, namun penyimpanannya dapat mengurangi waktu yang diperlukan untuk melakukan komputasi pembuatan merkle tree berdasarkan struktur direktori backup.

Penambahan level merkle tree sehingga mencakup blok-blok pada file juga mungkin dapat menambah kinerja kakas. Penggunaan merkle tree pada tiap blok berkas dapat membuat proses penyalinan file hanya pada blok berkas tertentu yang telah berubah saja. Namun implementasi ini tentu memerlukan penggunaan bahasa tingkat rendah serta pemahaman lebih lanjut mengenai struktur file dalam filesystem.

Selain itu, penggunaan bahasa yang lebih rendah seperti C, C++, Rust, atau Go serta pengunaan multithreading yang

efektif juga dapat meningkatkan kinerja kakas mengingat Python bukanlah bahasa yang dirancang dalam kecepatan eksekusi. Penggunaan struktur data merkle tree yang dibuat secara khusus juga akan membuat proses pembuatan dan perbandingan merkle tree lebih efisien.

Acknowledgment

Puji syukur kepada Tuhan Yang Maha Esa sehingga tulisan ini dapat diselesaikan dengan baik. Terima kasih kepada Dr. Ir. Rinaldi Munir, M.T. sebagai dosen mata kuliah kriptografi yang sudah memberikan ilmu dan pengetahuan mengenai kriptografi terutama fungsi *hash*.

References

[1] Andy Klein. "Backblaze Hard Drive Stats for 2017". https://www.backblaze.com/blog/hard-drive-stats-for-2017/ , diakses pada 15 Mei 2018.

[2] Tommy Kurnia. "Rata-rata Kecepatan Internet Dunia 42,7 Mbps, Indonesia?". https://www.liputan6.com/tekno/read/3466585/rata-rata-kecepatan-internet-dunia-427-mbps-indonesia diakses pada 15 Mei 2018.

[3] Rinaldi M., Bahan Kuliah IF4020 Kriptografi: Fungsi Hash.

[4] A. Menezes, P. van Oorschot, and S. Vanstone, "Handbook of Applied Cryptography," 5th ed., CRC Press, pp. 223–282, 1996.

[5] Becker, Georg. "Merkle Signature Schemes, Merkle Trees and Their Cryptanalysis". Ruhr-Universität Bochum. 2008.

VII. Pernyataan

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

Bandung, 18 Mei 2018

Muhammad Reza Ramadhan
13514107