Verifikasi Data dengan Pembuktian Keanggotaan pada Pohon Merkle

Daffa Pratama Putra 13518033
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13518033@std.stei.itb.ac.id

***Abstrak*—Data yang beredar di dunia digital sangat banyak, sehingga faktor keamanan menjadi hal penting agar tidak sembarang orang bisa mengakses data tersebut. Fungsi *hash* akan memanipulasi data tersebut menjadi suatu kode – kode tertentu yang hanya bisa diakses oleh orang tertentu saja. Dalam penerapannya, fungsi *hash* dikombinasikan dengan Pohon Merkle. Makalah ini membahas mengenai penerapan fungsi *hash* dan Pohon Merkle untuk melakukan pencocokan data.**

***Kata Kunci*—Merkle, Hash, Graf, Pohon**

I. PENDAHULUAN

Di dalam dunia digital seperti sekarang ini, data menjadi salah satu faktor penting dan banyak digunakan di berbagai bidang. Kemudahan mengakses dan menyimpan data menjadikan data semakin diminati oleh banyak orang. Ditambah lagi dengan teknologi *cloud computing* yang membuat manusia bisa mengakses data di mana saja dan kapan saja. Tetapi semakin bebas tersebarnya data ke publik, data tersebut semakin tidak aman.

Keamanan sebuah data biasanya dilakukan dengan metode *hashing*. Metode ini memiliki banyak kegunaan di bidang komputasi dan informatika, seperti penghapusan data, penambahan data, pencarian data, penyimpanan data, enkripsi dan dekripsi data, hingga proteksi data. Dengan adanya metode *hashing* ini, proses komputasi menjadi semakin cepat dan efisien. Penerapan dari teknik *hashing* adalah pada pencocokan sidik jari atau *fingerprint* untuk memastikan keamanan dari pengguna. Cara kerja dari *hashing* adalah mengubah isi data menjadi *string* tertentu yang bentuknya hampir tidak menyerupai kata – kata pada umumnya dengan perhitungan tertentu.

Pohon Merkle merupkan salah satu bagian yang penting dalam pencarian dan pencocokan data. Selain itu, Pohon Merkle banyak digunakan dalam teknologi *blockchain* dan kriptografi. Pohon ini berbentuk seperti pohon biner yang memiliki dua anak dengan orang tua adalah hasil penjumlahan dari kedua anak tersebut. Sistem yang dipakai dalam Pohon Merkle adalah *bottom-up*, yakni melakukan perhitungan dari bawah sampai bertemu akar atau *root* di bagian atas. Jika dilakukan perhitungan hasilnya berbeda dengan akar yang asli, maka data tersebut bukan bagian dari data di pohon. Biasanya perhitungan pada Pohon Merkle menggunakan metode *hashing* tertentu, seperti SHA-256.

II. DASAR TEORI

A. Graf

1. Pengertian Graf

Graf G membentuk suatu graf jika terdapat pasangan himpunan yang didefinisikan sebagai

$$G = (V,E)$$

dengan V adalah himpunan tidak kosong dari simpul – simpul atau *vertices* yang terdapat pada graf, sedangkan E atau *edges* adalah himpuan sisi yang menghubungkan antar simpul pada graf. Keduanya didefinisikan dengan

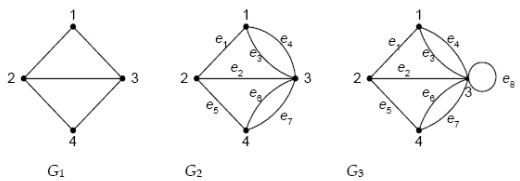
$$V = \{v_1 , v_2 , v_3 , \dots , v_n\}$$
$$E = \{e_1 , e_2 , e_3 , \dots , e_n\}$$

Sebuah graf mungkin saja tidak mempunyai sisi, tetapi tidak mungkin tidak punya simpul. Suatu graf dikatakan mempunyai gelang atau *loop* jika ada sisi yang berawal dan berakhir pada simpul yang sama. Jika komponen graf memiliki dua sisi yang menghubungkan dua simpul yang sama, misalnya e_1 = (1, 3) dan e_2 = (1, 3), maka graf tersebut memiliki sisi ganda (*multiple edges* atau *paralel edges*).

2. Jenis – Jenis Graf

Pengelompokan graf dibagi menjadi beberapa bagian. Berdasarkan ada atau tidaknya sisi gelang, graf dikelompokkan menjadi dua jenis, yakni

a) Graf Sederhana (*Simple graph*)
   Graf sederhana adalah graf yang tidak mengandung sisi gelang dan sisi ganda.
b) Graf Tak-Sederhana (*Unsimple graph*)
   Graf tak-sederhana adalah graf yang memiliki sisi gelang atau sisi ganda.



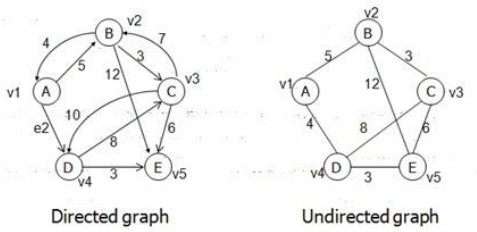
(G_1) graf sederhana, (G_2) multigraf, dan (G_3) multigraf

Gambar 1 : Graf sederhana dan graf tak-sederhana.
(Sumber : http://athayaniimtinan.blogspot.com/2017/12/pewarnaan-graf.html diakses pada 17 November 2019)

Berdasarkan orientasi arah, menurut Rosen, graf

dikelompokkan menjadi dua jenis, yaitu

a) Graf Berarah (*Directed Graph*)
   Graf berarah adalah graf yang sisinya memiliki orientasi arah.
b) Graf Tak-Berarah (*Undirected Graph*)
   Graf tak-berarah adalah graf yang sisinya tidak memiliki orientasi arah.



Gambar 2 : Graf berarah dan graf tak-berarah
(Sumber : https://arlgm.blogspot.com/2018/04/hola.html diakses pada 17 November 2019)

Berdasarkan ada atau tidaknya bobot, menurut Rosen, graf dikelompokkan menjadi dua jenis, yaitu

a) Graf Berbobot (*Weighted Graph*)
   Graf berbobot adalah graf yang pada setiap sisinya memiliki nilai tertentu yang dinotasikan dengan w_{ij} dengan i dan j merupakan simpul yang terhubung dengan sisi tersebut.
b) Graf Tak-Berbobot
   Graf tak-berbobot adalah graf yang pada sisi – sisinya tidak mempunyai nilai tertentu.

3. Terminologi Graf

Pada teori graf, terdapat beberapa terminologi atau istilah yang sering digunakan. Istilah tersebut antara lain :

a) Ketetanggaan (*Adjacent*)
   Dua simpul dikatakan bertetangga apabila keduanya terhubung langsung, yakni dihubungkan langsung oleh satu simpul.
b) Bersisian (*Incidency*)
   Untuk sembarang sisi $e = (v_i, v_j)$ dikatakan e bersisian dengan simpul v_i dan bersisian dengan simpul v_j.
c) Simpul Terpencil (*Isolated Vertex*)
   Simpul terpencil adalah simpul yang tidak mempunyai sisi yang menghubungkannya dengan simpul lain.
d) Graf Kosong (*Null Graph*)
   Graf kosong adalah graf yang himpunan sisinya merupakan himpunan kosong, yang berarti seluruh simpulnya adalah simpul terpencil.
e) Derajat (*Degree*)
   Derajat dari simpul adalah jumlah sisi yang bersisian dengan simpul tersebut. Jika ada sisi ganda, tetap dihitung menjadi dua derajat. Derajat pada graf berarah dibedakan menjadi d_{in} dan d_{out}.
f) Lintasan (*Path*)
   Lintasan yang berawal dari simpul v_0 dan berakhir di simpul v_n adalah barisan bergantian / selang – seling antara simpul dan sisi.
g) Sirkuit (*Circuit*)
   Sirkuit adalah lintasan yang berawal dan berakhir pada simpul yang sama.
h) Terhubung (*Connected*)
   Dua simpul, v_i dan v_j ,dikatakan terhubung jika terdapat lintasan dari v_i ke v_j. Graf G dikatakan terhubung jika untuk setiap pasang simpul v_i dan v_j terdapat lintasan yang dari v_i ke v_j.
i) Upagraf (*Subgraph*)
   Graf $G_1 = (V_1, E_1)$ dikatakan upagraf dari graf $G = (V, E)$ jika $V_1 \subseteq V$ dan $E_1 \subseteq E$.
j) Upagraf Rentang (*Spanning Upagraph*)
   Upagraf $G_1 = (V_1, E_1)$ dari $G = (V, E)$ dikatakan upagraf rentang jika G_1 mengandung semua simpul yang terdapat pada G atau $V_1 = V$.
k) *Cut-Set*
   Graf G merupakan graf terhubung. *Cut-set* dari graf G adalah himpunan sisi yang jika sisi tersebut dihilangkan akan menyebabkan graf G menjadi tidak terhubung.

B. Pohon

1. Pengertian Pohon dan Hutan

Pohon adalah graf tak-berarah terhubung yang tidak mengandung sirkuit. sederhana. Pohon telah digunakan oleh matematikawan Inggris bernama Arthur Cayley sejak tahun 1857 , untuk menghitung jumlah senyawa kimia. Diagram pohon seringkali digunakan untuk memecahkan persoalan dengan menggambarkan semua kemungkinan solusi yang ada.

Selain pohon, juga terdapat istilah hutan. Hutan adalah kumpulan pohon yang saling lepas. Hutan juga merupakan graf tak-terhubung yang tidak punya sirkuit.

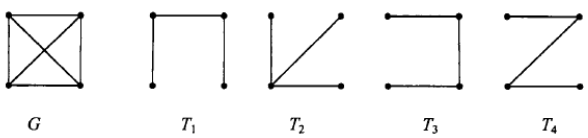
2. Sifat – sifat Pohon

Misalkan graf $G = (V, E)$ graf tak-berarah dengan simpul sebanyak n. Pernyataan berikut adalah ekivalen.

a) Graf G adalah pohon.
b) Setiap simpul pada graf G terhubung satu sama lain dengan hanya satu lintasan.
c) Graf G memiliki sisi sebanyak n – 1 sisi.
d) Semua sisi pada graf G adalah jembatan, yakni sisi yang bila sisi tersebut dihapus, menyebabkan graf terbagi menjadi dua bagian berbeda.
e) Penambahan satu sisi pada graf, menyebabkan graf mempunyai 1 sirkuit.

3. Pohon Merentang (Spanning Tree)

Pohon merentang adalah upagraf dari graf G yang mencakup semua simpul pada graf G. Misalkan graf $G = (V, E)$ dan pohon $T = (V_T, E_T)$. Pohon T disebut pohon merentang jika $V_T = V$ dan $E_T \subseteq E$. Pohon merentang didapat dengan cara memutus semua sirkuit yang terdapat pada graf G. Setiap graf terhubung paling sedikit mempunyai satu buah pohon merentang.



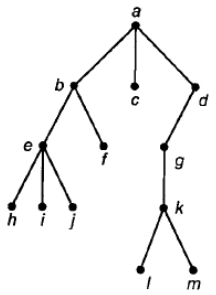
Gambar 3 : Graf G dengan T_1, T_2, T_3, T_4 adalah pohon merentang
(Sumber:http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2013 2014/Pohon%20(2013).pdf diakses pada 18 November 2019)

4. Pohon Merentang Minimum (Minimum Spanning Tree)

Pohon merentang minimum adalah pohon merentang yang mempunyai bobot minimum. Graf terhubung-berbobot mungkin saja mempunyai lebih dari satu pohon merentang. Penerapan pohon merentang minimum sangat luas, contohnya adalah untuk membangun jalur kereta dengan biaya minimum. Untuk membentuk pohon merentang minimum, dapat menggunakan algoritma Prim dan algoritma Kruskal.

5. Pohon Berakar (Rooted Tree)

Pohon berakar adalah pohon yang simpulnya diperlakukan layakya akar. Sedangkan sisi sisinya menjauh dari akar dengan diberi arah. Pada pohon berakar, terdapat istilah – istilah yang sering digunakan, yakni antara lain

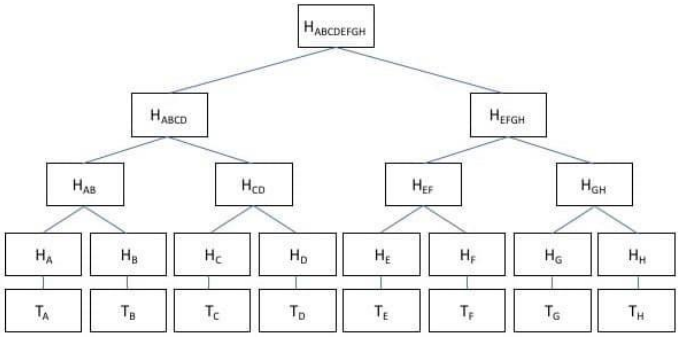


Gambar 4 : Pohon Berakar.
(Sumber:http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2013 2014/Pohon%20(2013).pdf diakses pada 19 November 2019)

a) Anak (*Child* atau *Children*) dan Orang Tua (*Parents*)
   Misalkan *a* adalah simpul pada pohon berakar. Simpul *b* dikatakan anak dari simpul *a* jika ada sisi dari simpul *a* ke simpul *b*. Maka simpul *a* dikatakan orang tua dari simpul *b*.
b) Lintasan (*Path*)
   Misalkan terdapat himpunan simpul *V*. Lintasan dari simpul v_1 menuju v_j adalah barisan simpul simpul v_1 , v_2 , v_3, …., v_j , sehingga v_i adalah orang tua dari v_{i+1} untuk i diantara 1 hingga j. Panjang lintasan adalah jumlah sisi yang dilalui selama lintasan. Panjang lintasan dirumuskan dengan $k - 1$ dengan *k* adalah simpul akhir dari lintasan.
c) Saudara Kandung (*Sibling*)
   Dua simpul merupakan saudara kandung jika keduanya memiliki simpul orang tua yang sama.
d) Upapohon (*Subtree*)
   Misalkan *x* merupakan simpul pada pohon *T*. Upapohon dengan *x* sebagai akar adalah upagraf $T' = (V', E')$, sehingga V' mengandung *x* dan semua keturunannya. E' adalah sisi dari semua lintasan yang berawal dari *x*.
e) Derajat (*Degree*)
   Derajat pada pohon berakar adalah jumlah upapohon atau anak yang terdapat pada simpul. Pengertian ini berbeda dengan pengertian derajat pada graf. Derajat maksimum dari simpul adalah derajat pohon tersebut.
f) Daun (*Leaf*)
   Daun adalah simpul yang berderajat nol atau tidak mempunyai anak.
g) Aras (*Level*) atau Tingkat
   Aras ditentukan dari akar. Akar mempunyai aras 0. Sedangkan simpul – simpul dibawahnya mempunyai 1 + panjang lintasan dari akar ke simpul tersebut.
h) Simpul Dalam (*Internal Nodes*)
   Simpul dalam adalah simpul yang memiliki anak di bawahnya.
i) Tinggi (*Height*) atau Kedalaman (*Depth*)
   Tinggi atau kedalaman adalah aras maksimum yang terdapat pada pohon. Tinggi juga bisa disebut sebagai panjang lintasan dari akar ke daun.

C. Pohon Merkle

Pohon Merkle dalam dunia informatika, sering disebut juga dengan pohon *hash* atau *hash tree*. Definisi dari pohon Merkle sendiri adalah sebuah pohon yang setiap simpulnya merupakan hasil *hash* dari penggabungan kedua simpul anaknya. Konsep pohon *hash* ini ditemukan oleh Ralph Merkle yang kemudian mematenkannya sebagai Pohon Merkle pada tahun 1979. Kegunaan dari pohon Merkle sangat banyak, terutama pada bidang kriptografi, keamanan data, *blockchain*, dan *Bitcoin*.



Gambar 5 : Ilustrasi Pohon Merkle.
(Sumber : https://medium.com/coinmonks/how-to-manually-verify-the-merkle-root-of-a-bitcoin-block-command-line-7881397d4db1 diakses pada 29 November 2019)

Struktur dari pohon Merkle dapat dilihat pada Gambar 5. Pada gambar tersebut terlihat bahwa simpul H_A adalah hasil *hashing* dari data T_A yang berada di bawahnya, begitu juga dengan simpul H_B, H_C, H_D, dan seterusnya. Lalu simpul H_A dan H_B adalah saudara, sehingga dapat dibuat orang tua nya yang merupakan hasil *hashing* dari penggabungan H_A dan H_B. Proses *hashing* dan penggabungan terus dilakukan hingga didapatkan akar pohon, yang merupakan $H_{ABCDEFGH}$.

Pohon Merkle banyak digunakan untuk mendukung koneksi *peer-to-peer*. Selain itu juga dalam pengecekan integritas data terhadap suatu bagian data atau blok, menjadi lebih efisien, karena pengecekan tidak perlu dilakukan sampai ke *top hash* atau *hash root*, melainkan cukup pada daunnya saja. Ketika sebuah bagian data ada yang rusak, hal ini cukup membantu, karena proses *recovery* yang dilakukan hanya pada bagian yang rusak saja, bukan keseluruhan data.

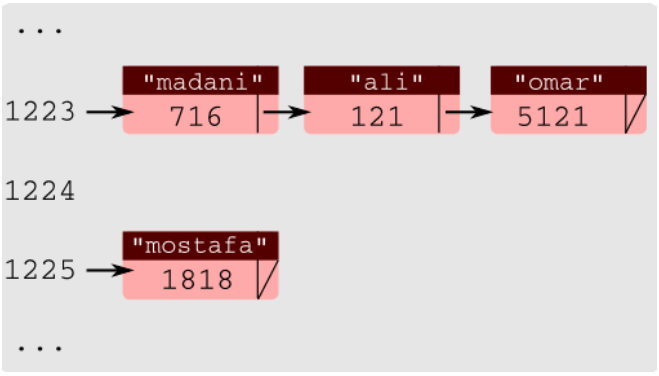
D. Hashing

Hashing adalah suatu proses mengubah sebuah *string* masukan menjadi sekumpulan angka/karakter dengan ukuran yang tetap. Keluaran dari proses *hashing* adalah berupa kunci dan nilai atau biasa disebut *key* value. Proses *hashing* tidak dilakukan manual satu-persatu karakter, tetapi dengan

menggunakan algoritma atau fungsi *hash*. Fungsi *hashing* yang baik, sebaiknya memenuhi persyaratan berikut :

a) Mudah untuk dikomputasi
b) Distribusi yang seragam
c) Minim terjadinya tabrakan atau *collision*

Dalam *hashing*, kunci yang berukuran besar akan dikonversi menjadi yang lebih kecil menggunakan fungsi *hash*. Nilai keluaran tersebut akan disimpan pada *hash table* dengan pendistribusian yang seragam ke seluruh tabel. Kunci tersebut digunakan untuk mencari entri yang dimaksud pada tabel *hash*.



Gambar 6 : Ilustrasi Collision pada Hash Table
(Sumber : http://kupaskode.blogspot.com/2015/09/tentang-hashing.html diakses pada 29 November)

Keluaran dari fungsi *hash* tidak selalu unik, sehingga ada kemungkinan terjadi *collision*. Misalkan ada beberapa masukan yang dikonversi dengan fungsi *hash* modulo sederhana

$$h(K) = K \bmod m$$

$h(K)$ = lokasi memori untuk kunci K
K = kunci masukan
m = jumlah memori pada tabel *hash*

Misalkan jumlah memori yang tersedia sebanyak m, sehingga nilai $h(K)$ bernilai antara 0 hingga m-1. Ketika diberi masukan kunci K, maka ada kemungkinan menemukan hasil modulo yaitu $h(K)$ yang sama. Kejadian ini yang dinamakan *collision*.

Saat terjadi *collision*, perlu dilakukan pencarian lokasi alternatif untuk menyimpan hasil *key value* tersebut. Pencarian lokasi dilakukan hingga ada tempat penyimpanan yang belum memiliki *key value* yang sama. Penanganan saat terjadi *collision* dapat dilakukan dengan beberapa cara, yakni :

a) *Linear Probing*
   Pencarian tempat dilakukan dengan interval yang tetap biasanya 1. Misalkan pada lokasi ke 6012 sudah ada yang mengisi, maka akan mengecek ke lokasi 6013, 6014, dan seterusnya hingga didapati lokasi penyimpanan pada tabel *hash* kosong.
b) *Quadratic Probing*
   Pencarian dilakukan dengan interval yang dideskripsikan dengan fungsi kuadratik. Cara ini hampir sama dengan *linear probing*, hanya saja berbeda penentuan intervalnya.
c) *Double Hashing*
   Pencarian lokasi menggunakan interval pencarian yang ditentukan dengan fungsi *hash* yang lain.

Ada beberapa algoritma *hashing* yang sering digunakan, diantaranya adalah MD4, MD5, RSA, SHA-256, SHA-512, dan Base64. Algoritma MD4 dan MD5 dikembangkan oleh Ronald L. Rivest dengan MD sendiri artinya adalah *Message Digest*. Generasi pertama dan kedua dari MD digunakan untuk membantu algoritma RSA untuk memecahkan permasalahan komputasi. Hingga saat ini sudah ada lima generasi dari MD.

IV. Verifikasi Data dengan Pembuktian Keanggotaan pada Pohon Merkle

1. Struktur Pohon Merkle

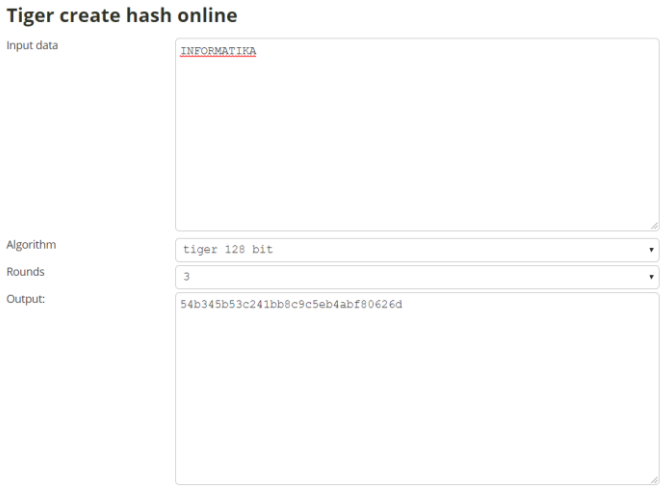
Pohon Merkle merupakan salah satu aplikasi dari teori graf dan pohon pada dunia informatika. Struktur dari pohon Merkle adalah seperti pohon biner, yakni memiliki dua percabangan atau dua anak pada setiap simpulnya. Nilai-nilai tiap simpulnya merupakan keluaran fungsi *hash* dari konkatenasi kedua anaknya. Simpul yang paling ujung atau yang disebut dengan daun merepresentasikan hasil fungsi *hash* yang dilakukan pertama kali. Fungsi *hash* yang digunakan pada tiap simpul adalah sama dan dilakukan berulang kali hingga ditemukan akar pohonnya.

2. Pembentukan Pohon Merkle

Dalam membangun pohon Merkle, dibutuhkan beberapa sampel data. Data yang akan digunakan untuk membangun pohon Merkle adalah sekumpulan teks yang diambil dari mars SMA Negeri 5 Surabaya yang terdiri dari :

a) Smalane suci dalam pikiran.
b) Smalane benar jika berkata.
c) Smalane tepat dalam tindakan.
d) Smalane dapat dipercaya.

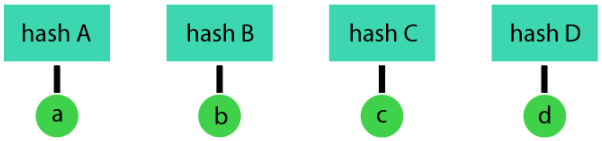
Fungsi *hash* yang digunakan untuk membangun pohon Merkle adalah bebas. Pada kali ini fungsi *hash* yang digunakan adalah Tiger Tree Hash. Proses *hashing* dilakukan pada pranala http://www.unit-conversion.info/texttools/tiger/. Pranala tersebut dapat melakukan *hashing* secara otomatis. Pengguna hanya tinggal memasukkan teks masukan dan memilih algoritma yang ingin digunakan. Lalu muncul keluaran berupa hasil *hashing* tersebut.



Gambar 7 : Tampilan Pranala Hash Online
(Sumber : http://www.unit-conversion.info/texttools/tiger/ diakses pada 1 Desember 2019)

Berikut adalah keluaran fungsi *hash* menggunakan algoritma tiger 128 bit dari masukan seperti di atas.

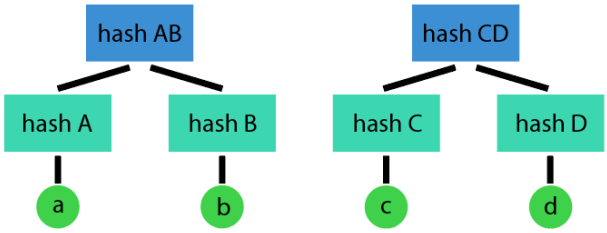
- hash A : Smalane suci dalam pikiran → "34d5ae3dd40778a552faa2803ad18dc2"
- hash B : Smalane benar jika berkata → "af6370109b72bfc4b9b304b777027092"
- hash C : Smalane tepat dalam tindakan → "a2095467b9a677c944508be1938af736"
- hash D : Smalane dapat dipercaya → "42736c1b3b0569d2ed1120589c1be994"



Gambar 8 : Pembentukan Daun Pohon Merkle

Seperti yang telah dijelaskan sebelumnya, bahwa keluaran fungsi *hash* dari keempat teks di atas akan menjadi daun pohon Merkle. Daun – daun tersebut diberi nama hash A, hash B, hash C, dan hash D. Lalu daun yang bersaudara akan dibentuk menjadi simpul orang tua dengan cara dilakukan konkatenasi dan jadikan masukan fungsi *hash*. Pada kasus ini, daun hash A dan hash B adalah bersaudara dan daun hash C dan hash D juga bersaudara. Sehingga didapatkan orang tua dari daun – daun tersebut yang diberi nama hash AB dan hash CD.

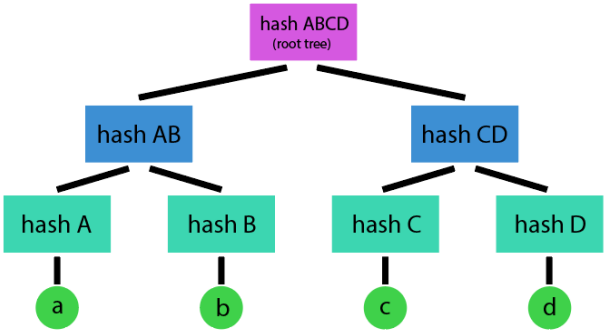
- hash AB : H(hash A) + H(hash B) → "88eca2f58edcf0ac5c3726b6d4746611"
- hash CD : H(hash C) + H(hash D) → "ce2975cd525bc74658a2fc15e1bb8c0b"



Gambar 9 : Pengembangan Pohon Merkle

Setelah didapat hash AB dan hash CD, karena *root tree* belum didapatkan, proses *hashing* dilanjutkan. *Root tree* didapat ketika simpul pohon tidak mempunyai saudara. Simpul hash AB bersaudara dengan simpul hash CD, sehingga keduanya dikonkatenasi dan dilakukan *hashing* agar didapatkan orang tuanya. Simpul orang tua diberi nama dengan hash ABCD.

- hash ABCD : H(hash AB) + H(hash CD) → "7712c9fa8a1faf07d9dba121ae96e504"



Gambar 10 : Bentuk Akhir Pohon Merkle dengan 4 daun

Simpul hash ABCD merupakan *root tree* karena simpul tersebut tidak memiliki saudara. Jadi bentuk akhir dari pohon Merkle dengan empat data masukan adalah seperti pada Gambar 10. Daun yang terdapat pada pohon Merkle tidak harus berjumlah 4 seperti pada contoh, namun harus sejumlah 2^n, n adalah bilangan bulat positif.

3. Verifikasi Data dengan Pembuktian Keanggotaan

Melakukan verifikasi data pada pohon Merkle adalah dengan menggunakan pembuktian keanggotaan. Pembuktian pada pohon Merkle menjadi salah satu kelebihan tersendiri sebab dinilai lebih efisien dan cepat. Proses pengecekan keanggotaan pada pohon Merkle dilakukan dengan memberikan simpul saudara dari data sampel dan dilakukan proses konkatenasi dan *hashing* seperti pada pembentukan pohon Merkle. Apabila data sampel merupakan anggota dari pohon Merkle, maka akan mengembalikan keluaran *root tree* yang sama seperti pada pohon Merkle yang sebenarnya.

Untuk mempermudah penjelasan mengenai pembuktian keanggotaan, akan diberikan uji kasus. Misalkan pohon Merkle pada gambar 10 terdapat pada suatu server. Seorang pengguna yang tidak mengetahui keanggotaan pohon Merkle tersebut, ingin mengecek apakah teks "Smalane benar jika berkata" merupakan anggotanya. Dalam melakukan pembuktian ini, pengguna akan mengirimkan data sampel tersebut ke server dan server akan melakukan verifikasi.

Berikut langkah – langkah yang dilakukan untuk melakukan pembuktian keanggotaan :

1. Teks yang diverifikasi akan di-*hash* menggunakan fungsi *hash* seperti pada pembentukan pohon Merkle.
2. Keluaran dari fungsi *hash* akan dicocokkan dengan daun pada pohon Merkle.
3. Keluaran fungsi *hash* merupakan salah satu daun, maka akan dikirimkan daun yang bersaudara dengannya.
4. Konkatenasikan keluaran fungsi *hash* dari data sampel dengan daun saudaranya dan lakukan *hashing* pada hasil konkatenasi tersebut.
5. Lakukan proses konkatenasi dan *hashing* sampai ditemukan *root* pohon Merkle.
6. Apabila *root* pohon pada pengecekan sama seperti *root* pohon Merkle yang sesungguhnya, maka data sampel tersebut merupakan anggota dari pohon.

Pada uji kasus ini fungsi *hash* yang digunakan adalah algoritma Tiger Hash maka data sampel, yakni teks "Smalane benar jika berkata" akan di-*hash* menggunakan fungsi tersebut. Keluaran data samepel ditemukan adalah daun hash B. Lalu server akan mengirimkan hash A sebagai saudara dari hash B dan hash CD sebagai saudara dari hash AB. Proses berhenti ketika telah mendapatkan *root tree*. Pada pencocokan data sampel ini, *root tree* yang didapat adalah sama seperti *root tree* pada pohon Merkle yang terdapat pada server. Maka data sampel, yakni teks "Smalane benar jika berkata" merupakan salah satu anggota dari pohon Merkle.

Proses pencocokan seperti ini banyak digunakan dalam sistem *peer-to-peer*. Contohnya adalah pada sistem pengunduhan menggunakan Torrent. Ketika mendownload dengan Torrent, pengunduh akan menerima "*seeds*" dari orang lain. Tetapi pengunduh tidak dapat memastikan apakah data

yang diunduh adalah benar atau tidak. Untuk memastikannya, digunakan verifikasi dengan pembuktian keanggotaan pada pohon Merkle. Dalam melakukan pembuktian keanggotaan pada pohon Merkle, dibutuhkan kompleksitas ruang dan waktu sebesar O(log n), dengan n adalah jumlah daun pada pohon Merkle.

V. KESIMPULAN

Pohon Merkle merupakan salah satu *hash tree* yang sangat berguna, terutama dalam bidang kriptografi dan *cryptocurrency*. Pohon Merkle dibangun menggunakan teori graf, pohon, dan algoritma *hashing*. Struktur pohon Merkle yang sedemikian rupa membuat keamanan data yang disimpan lebih terjamin. Selain itu verifikasi data menggunakan pembuktian keanggotaan pada pohon Merkle menjadi kelebihan tersendiri, karena proses verifikasi tidak perlu dilakukan untuk semua simpul yang ada pada pohon Merkle, melainkan hanya simpul – simpul yang bersangkutan saja. Kompleksitas ruang dan waktu dalam melakukan pembuktian keanggotaan terbilang efisien, yaitu O(log n).

VI. UCAPAN TERIMA KASIH

Penulis mengucapkan rasa syukur kepada Allah SWT. karena dengan rahmat dan berkahnya, penulis bisa menyelesaikan penulisan makalah ini dengan baik. Penulis juga mengucapkan terima kasih kepada dosen mata kuliah IF2120 Matematika Diskrit, yakni Bapak Dr. Ir. Rinaldi Munir, M.T, Ibu Dra. Harlili, M.Sc., dan Ibu Fariska Zakhralativa Ruskanda, S.T., M.T, karena telah membimbing dalam mengerjakan makalah ini. Tak lupa juga, penulis mengucapkan terima kasih kepada teman- teman IF 2018 yang telah memberikan dukungan dan bantuan selama pengerjaan makalah ini.

REFERENSI

[1] Munir, Rinaldi. 2006. *Diktat Kuliah IF2120 Matematika Diskrit*. Bandung:Program Studi Teknik Informatika.
[2] Toisutta, Eka Yusrianto. 2007. *Pengaksesan Daun Secara Random Pada Hash Tree*. Bandung:Program Studi Teknik Informatika.
[3] Sutarno, Muhammad Visat. 2016. *Implementasi ECDSA untuk Verifikasi Berkas Berukuran Besar dengan Menggunakan Merkle Tree*. Bandung:Program Studi Teknik Informatika.
[4] https://hackernoon.com/merkle-trees-181cb4bc30b4 diakses 16 November 2019 pada 08.15 WIB.
[5] https://medium.com/crypto-0-nite/merkle-proofs-explained 6dd429623dc5 diakses 16 November 2019 pada 08.24 WIB.
[6] https://medium.com/byzantine-studio/blockchain-fundamentals-what-is-a-merkle-tree-d44c529391d7 diakses 29 November 2019 pada 08.53 WIB.
[7] https://www.codeproject.com/Articles/1176140/Understanding-Merkle-Trees-Why-use-them-who-uses-t diakses pada 2 Desember 2019 pada 11.35 WIB.
[8] https://medium.com/coinmonks/merkle-tree-101-a3ca025dc318 diakses 4 Desember 2019 pada 17.12 WIB.

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, 6 Desember 2019

Daffa Pratama Putra
13518033