

Aslan Alwi
Munirah

BLOCKCHAIN

FILOSOFI DAN KONSTRUKSI



**UNDANG-UNDANG REPUBLIK INDONESIA
NOMOR 28 TAHUN 2014
TENTANG HAK CIPTA**

PASAL 113

KETENTUAN PIDANA SANGSI PELANGGARAN

1. Setiap orang yang dengan tanpa hak melakukan pelanggaran hak ekonomi sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf i untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 1 (satu) tahun dan/atau pidana denda paling banyak Rp. 100.000.000 (seratus juta rupiah).
2. Setiap Orang yang tanpa hak dan/atau tanpa izin Pencipta atau pemegang Hak Cipta melakukan pelanggaran hak ekonomi Pencipta sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf c, huruf d, huruf f, da/atau huruf h untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 3 (tiga) tahun dan/atau pidana denda paling banyak Rp. 500.000.000,00 (lima ratus juta rupiah)
3. Setiap Orang yang tanpa hak dan/atau tanpa izin Pencipta atau pemegang Hak Cipta melakukan pelanggaran hak ekonomi Pencipta sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf a, huruf b, huruf e, dan/atau huruf g untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 4 (empat) tahun dan/atau pidana denda paling banyak Rp 1.000.000.000,00 (satu miliar rupiah)
4. Setiap orang yang memenuhi unsur sebagaimana dimaksud pada ayat (3) yang dilakukan dalam bentuk pembajakan, dipidana dengan pidana penjara paling lama 10 (sepuluh) tahun dan/atau pidana denda paling banyak Rp. 4.000.000.000,00 (empat miliar rupiah)

BLOCKCHAIN

Filosofi dan Konstruksi



BLOCKCHAIN Filosofi dan Konstruksi

Penulis :

Aslan Alwi, Munirah

Hak Cipta © 2019, Penulis

Hak Terbit © 2019, Penerbit : Unmuh Ponorogo Press

Jalan Budi Utomo Nomor 10 Ponorogo-63471

Telp. (0352) 481124, 487662

Faks. (0352) 461796

E-mail : unmuhpress@umpo.ac.id

Desain Sampul: Tim Unmuh Ponorogo Press

ISBN : 978-602-0791-37-1

Cetakan Kedua, Maret 2020

167 halaman, 15,5 x 23 cm

Dilarang keras mengutip, menjiplak, memfotocopi, atau memperbanyak dalam bentuk apa pun, baik sebagian maupun keseluruhan isi buku ini, serta memperjualbelikannya tanpa izin tertulis dari penerbit Unmuh Ponorogo Press.

KATA PENGANTAR

Segala puji bagi Allah dan terima kasih kepada Nya, Tuhan Semesta Alam, yang telah memberikan kemampuan dan pengetahuan kepada kami untuk menyusun buku ini, serta Shalawat dan Salam kepada Rasulullah Nabi Muhammad Shallallahuaihi Wasallam.

Buku ini kami buat sebagai bentuk dari semangat kami untuk menyumbang bagi kemajuan dan kejayaan bangsa agar dapat menjadi sebuah bangsa yang berdiri sama tinggi dan sejajar dengan bangsa-bangsa lain, dapat menjadi pemimpin bagi kemajuan peradaban dan meretas jalan kepada kerjasama dan kolaborasi yang membawa kita kepada lapis demi lapis ketinggian ilmu pengetahuan dan teknologi yang universal yang pada gilirannya diharapkan dapat membawa kepada kesejahteraan dan kemakmuran bersama. Baik bersama sebagai sebuah bangsa dan bersama sebagai kelangsungan species manusia di bumi.

Pada hari ini, mungkin kita tidak memiliki musuh bersama yang dapat mempersatukan kita, sehingga kita mengambil sebagian dari kita sebagai musuh bagi sebagian yang lain, lalu kita menjadi berpecah belah dan menyukai peperangan. Demikian semata mungkin karena perang adalah sifat alami di dalam alam semesta dan menjadi sifat intrinsik yang bersembunyi di dalam bawah sadar kita. Akan tetapi, semestinya kita mengarahkan bakat alami itu kepada musuh yang bisa menyatukan kita sehingga dia menjadi sebuah berkah bagi kelangsungan hidup manusia.

Kebodohan adalah musuh bersama dan sebenarnya bagi kita. Dia menyebabkan kemiskinan yang global di seluruh dunia, menyelipkan rasa permusuhan antara kita, melahirkan informasi-informasi keliru dan membakar ketentraman hidup kita, membawa manusia kepada pencemaran yang luas di lautan sehingga meruntuhkan rantai makanan dan membuat species manusia yang berdiri di puncaknya berada diambang kepunahan.

Karena itu, dengan langkah kecil ini, kami berusaha memeranginya itu dengan membakar semangat literasi kita akan ilmu

pengetahuan dan teknologi. Kami memilih teknologi blockchain sebagai salah satu ranah spesifik yang penting untuk kita pahami, kuasai dan pandai mengkonstruksikannya karena dia adalah sebuah jalan yang berpotensi merevolusi dunia kita hari ini. Merevolusi seluruh proses bisnis yang menjadi mekanisme cara kerja dunia.

Di dalam buku, kami usahakan semua ilustrasi adalah asli, akan tetapi terdapat beberapa yang diambil dari internet. Semua gambar yang terpaksa diambil dari internet seluruhnya adalah *Unknown Author* dan di bawah lisensi CC BY atau CC BY-SA. Demikianlah sepatah kata dari kami.

Dari kami

Penulis

DAFTAR ISI

KATA PENGANTAR	v
DAFTAR ISI.....	vii
BAB 1.....	1
Pendahuluan	1
Filosofi Blockchain	2
1. Blockchain sebagai sebuah penyimpanan terdistribusi	3
2. Blockchain sebagai sebuah <i>ledger</i> (jurnal besar) terdistribusi ...	8
3. Blockchain sebagai sistem basisdata terdistribusi	16
a) Sebagai struktur data yang terstruktur (<i>structured data</i>)	20
b) Sebagai struktur data yang tak terstruktur (<i>unstructured data</i>).....	21
4. Blockchain sebagai sebuah objek terdistribusi	23
5. Blockchain sebagai agen terdistribusi	23
6. Perbedaan pengertian tentang wallet, node, peer, fullnode, miner dan masternode di dalam arsitektur blockchain	24
7. Bagaimana mekanisme blockchain mencegah penggunaan dua kali uang crypto (Double Spending).....	26
8. Kemungkinan keberhasilan serangan double attack pada blockchain	29
9. Arti confirmation di dalam blockchain terhadap transaksi yang berhasil ditambahkan sebagai block.....	30
10. Tentang bagaimana proses penambahan blok di dalam blockchain terjadi	31
11. Merkle Trees, sebuah cara hashing untuk menjelaskan urutan transaksi	43
12. Difficulty dalam blok blockchain bitcoin.....	46
13. Konstruksi Alamat (Address) Wallet di dalam blockchain bitcoin	
49	

14. Anatomi transaksi blockchain bitcoin.....	53
BAB 2.....	85
Penerapan Blockchain.....	85
Bidang Farmasi dan Medis	85
Bidang Pendidikan dan akademik.....	95
BAB 3.....	100
Contoh Sederhana Konstruksi Blockchain.....	100
3.1. Spesifikasi Blockchain.....	101
3.2. Spesifikasi Node Blockchain.....	105
3.3. Spesifikasi Protokol Autentikasi Penambahan Blok	107
3.4. Kode untuk login pada aplikasi node:.....	116
3.5. Kode untuk file blockchain versi json lain (sebagai perbandingan)	118
3.6. Kode untuk membaca blok dan membaca hash serta membuat hash baru untuk blok berikut	120
BAB 4.....	149
Filosofi Blockchain Ethereum	149
A. Blockchain sebagai automata (transaction-based state machine)	
149	
1. World state di dalam blockchain Ethereum	152
2. Anatomi blok di dalam blockchain Ethereum.....	153
B. Konstruksi Blockchain menggunakan Azure Blockchain Workbench	153
Glossary.....	154
Indeks.....	155
Daftar Pustaka	156

BAB 1

Pendahuluan

Blockchain adalah salah satu komponen teknologi di dalam Revolusi Industri 4.0 dan menjadi isu penting bagi semua negara di dunia untuk bisa masuk dengan sempurna di dalam era revolusi tersebut. Karena itu, sudah menjadi kewajiban kita sebagai sebuah bangsa untuk berusaha menguasai teknologi ini dan meretas jalan kita sendiri yang merdeka dan berdaulat di dalam sains dan teknologi.

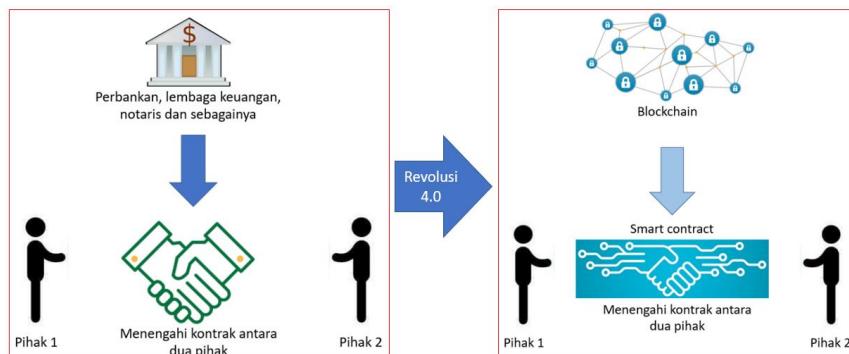
Blockchain pada hakikatnya adalah sebuah sistem penyimpanan (*storage*) terdistribusi di dalam segala pengertian penyimpanan (penyimpanan data, penyimpanan uang, penyimpanan surat-surat berharga, penyimpanan kode komputer dan sebagainya). Tetapi di dalam konsep terdistribusi ini, file data blockchain tidak terfragmentasi, namun hanya tersalin utuh dan terdistribusi ke seluruh node. Selain itu secara keseluruhan, blockchain ini tidak terpusat tetapi terdesentralisasi. Bahkan kita dapat melihatnya saja lebih sederhana sebagai sebuah sistem basisdata yang terdistribusi dan terdesentralisasi. Hanya saja, teknologi untuk mengkonstruksikan sistem ini dibangun di atas fondasi kriptografi (pensandian) dan protokol-protokol keamanan. Sehingga dia menjadi bersifat aman secara sendirian, independen dan terdistribusi. Dengan segala sifat-sifat baru itu, dia dapat menggantikan berbagai proses bisnis yang telah mapan selama ini untuk menjalankan mesin-mesin ekonomi, bisnis, politik, sosial dan berbagai hal di dunia. Karena sifat-sifat itulah blockchain berpotensi merevolusi dunia sehingga dia termasuk sebagai komponen revolusi industri 4.0.

Revolusi yang ditimbulkan oleh blockchain terutama menyarar kepada segala proses bisnis di dalam segala bidang. Secara umum, blockchain berpotensi merubah segala proses bisnis yang terpusat dan dipegang oleh lembaga-lembaga bisnis, sosial, pemerintah dan sebagainya menjadi terdistribusi dan independen, merubah segala rule-rule regulasi menjadi otomatis dan merdeka.

Secara tersirat, sebenarnya blockchain memberikan fondasi

masa depan bagi semua mesin-mesin cerdas buatan untuk bekerja di berbagai bidang menggantikan kekuasaan orang per orang atau lembaga swasta atau pemerintah. Dia menggantikan seluruh aturan berkekuatan hukum yang dikendalikan, dieksekusi dan dievaluasi oleh manusia atau sekelompok manusia menjadi sekumpulan rule yang tertulis di dalam program komputer yang dikendalikan, dieksekusi dan dievaluasi oleh sekelompok mesin cerdas secara otomatis, dan semuanya itu juga berkekuatan hukum karena tertulis di atas fondasi blockchain yang memeliharanya dengan prinsip-prinsip kriptografi dan protokol-protokol keamanan yang kuat dan dapat diverifikasi oleh peninjau-peninjau ahli akan kebenaran ilmiahnya.

Gambar 1 memberikan ilustrasi tentang bagaimana blockchain dapat merevolusi industri.



Gambar 1. Bagaimana blockchain merevolusi industri

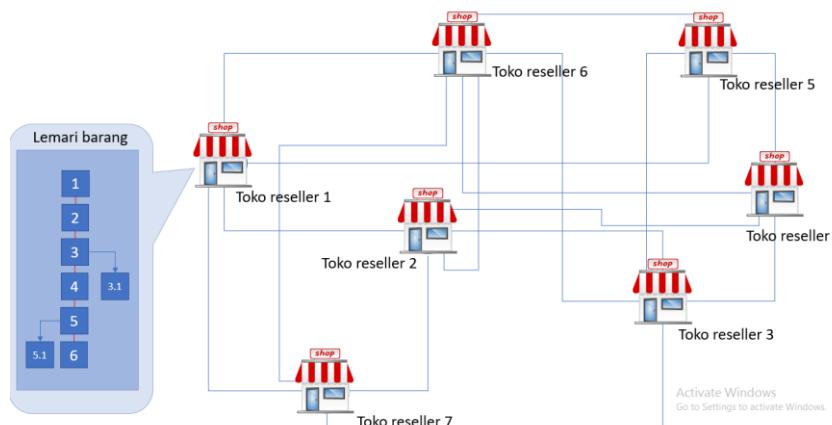
Filosofi Blockchain

Blockchain sebagai sebuah teknologi, dapat kita lihat fondasinya pada makalah Satoshi (Nakamoto, no date) dapat kita lihat dalam beberapa cara pandang. Cara pandang pertama yaitu bahwa kita bisa melihatnya sebagai sebuah sistem penyimpanan terdistribusi terenkripsi. Cara pandang kedua dia dapat dilihat sebagai sebuah jurnal atau ledger terdistribusi. Cara pandang ketiga kita bisa melihatnya sebagai sebuah sistem basisdata terdistribusi dan terakhir di dalam cara pandang yang mungkin lebih teknis yaitu bahwa kita bisa melihatnya sebagai sebuah jejala objek terdistribusi atau jejala agen terdistribusi.

1. Blockchain sebagai sebuah penyimpanan terdistribusi

Disana ada beberapa penyimpanan yang jumlahnya bisa ratusan, bisa ribuan, bahkan jauh lebih banyak lagi. Perumpamaan pada dunia keseharian kita, tempat penyimpanan itu dapat dibayangkan sebagai sebuah toko reseller dari sebuah distributor barang. Ada mungkin terdapat toko reseller di berbagai kota. Tiap-tiap toko reseller saling menyimpan nomor kontak masing-masing dan mereka dapat saling berkomunikasi secara satu ke satu (peer to peer) satu sama lain.

Perumpamaan ini dapat dilihat pada Gambar 2 Terdapat 6 buah toko reseller yang selalu menjual barang-barang dari distributor yang sama setiap saat.



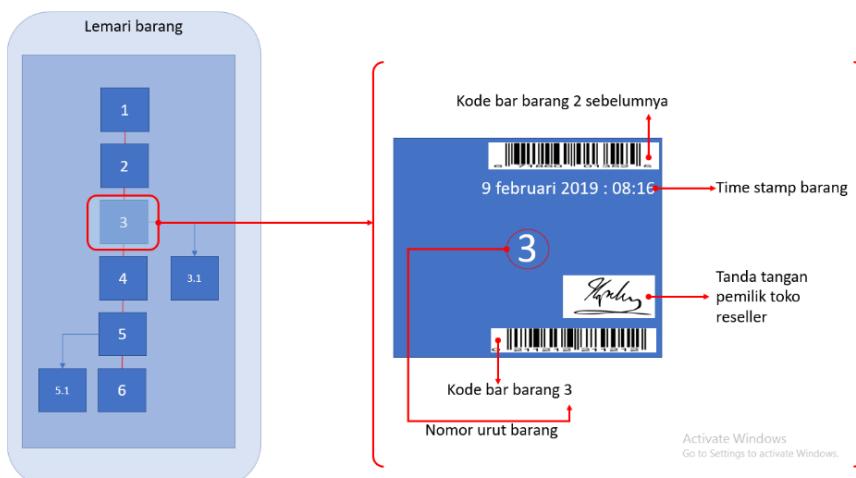
Gambar 2. Jejala toko reseller barang

Manakala distributor hendak memperkenalkan sebuah barang baru, maka semua toko reseller itu menerima stok barang baru yang sama. Barang-barang yang didistribusikan ke toko-toko reseller itu disimpan oleh setiap toko ke dalam lemari-lemari penyimpanan barang. Setiap lemari terdiri dari kotak-kotak penyimpanan yang diberi nomor dan kunci. Sehingga tidak sebarang orang boleh membukanya.

Setiap barang ditandatangani dan diberi stempel oleh pemilik toko yang menyatakan nomor urut penerimaan barang, tanggal penerimaan dan dua kode bar. Kode bar pertama menyatakan kode bar barang yang datang sebelumnya dan kode bar kedua menyatakan kode

bar barang itu sendiri. Ini menyebabkan setiap barang disetiap kotak saling berkaitan kode bar. Gambar 3 memberikan gambar sederhana bagaimana pemilik toko reseller menyimpan barangnya di lemari kotak.

Sistem penyimpanan barang yang saling berkaitan kode bar oleh toko reselller ini menyebabkan orang-orang iseng kesulitan untuk menukar posisi barang antar kotak dan setiap barang yang diambil dan ditukar oleh orang yang bermaksud jahat akan menyebabkan ketidakcocokan urutan kode bar barang yang saling berkaitan.



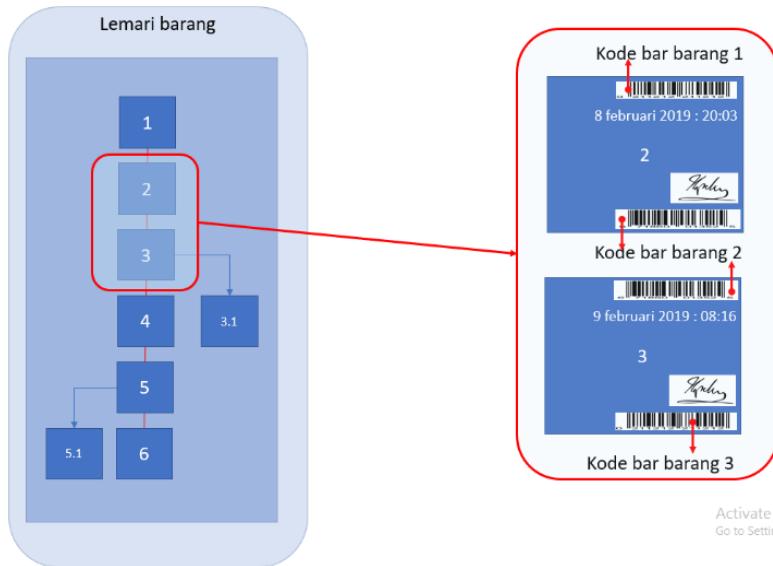
Gambar 3. Cara pemilik toko menyimpan barang di lemari kotak

Cara pemilik toko reseller barang dengan membuat keterkaitan kodebar antar barang membentuk sebuah rantai barang yang saling yang diharapkan dapat saling mengamankan membentuk rantai (*chain*) barang di dalam lemari. Keterkaitan itu diilustrasikan pada gambar 4.

Keterkaitan atau rantai kode antar barang dibentuk oleh kode bar masing-masing barang. Barang pada kotak 2 menyimpan kode bar barang pada kotak 1, demikian pula bahwa barang pada kotak 3 menyimpan kode bar barang pada kotak 3. Rantai kode bar ini mencegah orang untuk menukar barang dari dalam lemari. Karena jika terdapat seseorang yang mencoba menukar sebuah barang misal pada kotak 2, barang penggantinya mestilah memiliki kode bar sendiri yang berbeda. Misalkan si penukar hendak menyesuaikan kode bar pada

barang baru agar konsisten, dia akan menulis kode bar barang 1 pada barang penukar tersebut, kemudian dia mesti harus menulis ulang juga kode bar pada barang 3.

Pekerjaan menukar kode bar ini akan menjadi sangat sulit manakala pada kode-kode bar itu diterapkan enkripsi atau hash antar kode bar satu sama lain. Kita dapat melihat bagaimana kode bar itu saling bertautan membentuk rantai kode bar pada gambar 4.



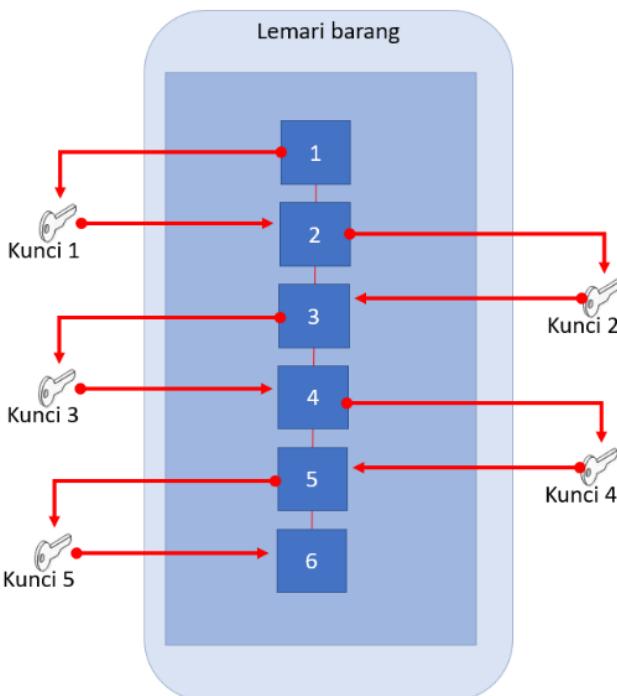
Gambar 4. Keterkaitan kode bar antar barang (chain) di lemari penyimpanan toko reseller

Pemilik toko reseller dapat juga menerapkan cara kunci berantai antar kotak barang. Kunci berantai ini dilakukan dengan aturan sebagai berikut:

- Setelah kotak 1 selesai ditambahkan barang maka kunci kotak 1 disimpan di dalam kotak 2.
- Setelah kotak 2 dimasukkan barang baru, maka kotak 2 terdiri dari kunci kotak 1 dan barang.
- Lalu kunci kotak 2 sendiri disimpan pada kotak 3.
- Jika ada barang baru lagi yang masuk pada kotak 3, lakukan proses serupa dengan langkah 2.

e) Demikian seterusnya jika setiap ada barang baru masuk.

Gambar 5 mengilustrasikan cara pemilik toko mengunci setiap kotak. Cara ini membentuk rantai kunci antar kotak. Sehingga setiap orang yang berusaha untuk membukanya menggunakan kunci haruslah membuka sebagian kotak yang lain secara berantai.



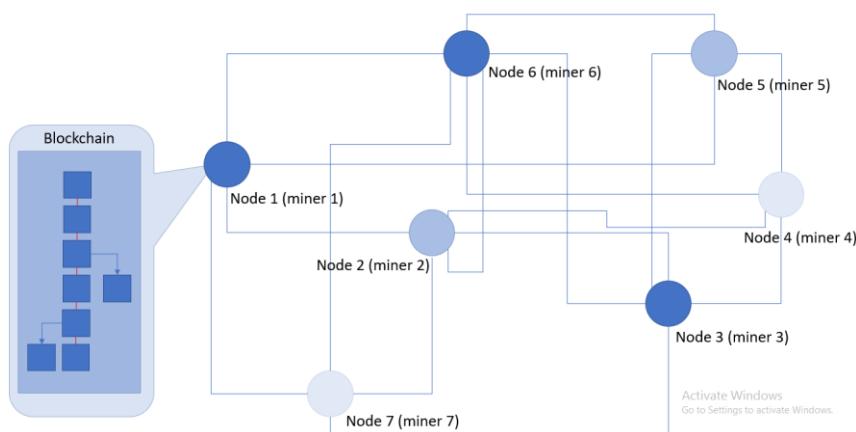
Gambar 5. Struktur kunci kotak dari lemari

Perumpamaan ini sebenarnya tidak benar-benar sepadan dengan hakikat arsitektur blockchain, karena hanya menggambarkan konsep distribusi, penyimpanan barang, tanda tangan dan keterkaitan kode dan kunci antar barang. Tetapi untuk pertama, kita dapat menjelaskan bahwa mirip seperti itulah blockchain.

Jika kita membawa pengertian barang sesungguhnya di dalam perumpamaan di atas ke dalam blockchain maka barang yang dimaksud adalah barang digital. Barang digital yang berbentuk catatan digital, uang digital (misal bitcoin), surat-surat berharga, perjanjian kontrak, transaksi jual-beli dan sebagainya.

Gambaran sesungguhnya sebuah blockchain, toko-toko reseller itu dilihat sebagai masing-masing adalah sebuah node yang masing-masing dari setiap node itu menyimpan salinan blockchain. Setiap node itu dapat saling berkomunikasi satu sama lain di dalam apa yang disebut sebagai komunikasi peer to peer (P2P). Setiap node juga disebut sebagai penambang (*miner*) karena setiap node yang berhasil menambahkan blok baru dilihat sebagai sebuah usaha komputasi dan mereka mendapat *fee* atau upah dari usaha itu yang berupa reward mata uang digital (bitcoin dan sebagainya yang sejenis) dan karena secara berkala blockchain memproduksi koin blockchain yang disebar kepada penambang dalam sebuah probabilitas tertentu yang mempertimbangkan usaha komputasi atau kekuatan komputasi yang telah dilakukan oleh node tersebut. Gambar 6 memberikan ilustrasi sebuah blockchain dalam cara pandang jejala node.

Arsitektur ini adalah sebuah arsitektur standar. Artinya bahwa orang-orang disana masih terus mengembangkan arsitektur blockchain yang berbeda-beda. Perbedaan arsitektur yang melahirkan arsitektur blockchain baru menyebabkan aturan-aturan (berbentuk protokol) pemberian bitcoin sebagai upah kepada node-node di dalam blockchain menjadi tidak kompatibel lagi sehingga mengharuskan mereka membentuk aturan-aturan baru atau protokol-protokol baru yang itu juga berarti membuat mata uang digital baru yang bukan bitcoin. Dengan alasan inilah mata uang seperti ether di dalam arsitektur blockchain Ethereum dibuat orang atau mata uang seperti XEM di dalam arsitektur blockchain NEM, dan sebagainya.



Gambar 6. Blockchain sebagai sebuah jejala node

2. Blockchain sebagai sebuah *ledger* (jurnal besar) terdistribusi

Di awal mula kemunculan teknologi blockchain oleh Satoshi Nakamoto. Blockchain dimaksudkan sebagai fondasi untuk ide mata uang digital (bitcoin) yang dicetuskan oleh Satoshi Nakamoto. Ini seperti uang kertas dan uang logam, yang basis hitungnya diletakkan pada buku jurnal transaksi.

Blockchain dimaksudkan sebagai buku jurnal digital dengan perluasan pengertian yang mencakup keamanan digital di internet. Oleh karena itu, kita bisa melihat secara mendasar bahwa blockchain dapat digambarkan sebagai sebuah buku jurnal transaksi. Secara khusus semua transaksi yang memiliki nilai bitcoin. Akan tetapi konsep transaksi diperluas untuk segala jenis transaksi di dalam segala jenis proses bisnis di internet ataupun di dunia nyata yang di terjemahkan ke digital.

Blockchain sebagai buku jurnal transaksi secara sederhana adalah sebuah tabel yang rekord-rekord nya adalah catatan transaksi. Gambar 7 mengilustrasikan sebuah jurnal yang mencatat 7 buah transaksi dalam kehidupan sehari-hari.

Transaksi	Nilai
Membayar pulsa listrik	20000
Menerima kiriman uang	5000000
Membayar sewa rumah	200000
Menerima hadiah	300000
Beli sepatu	100000
Beli tas	100000
Terima gaji bulan ini	2000000



Buku jurnal (*ledger*)

Gambar 7. Contoh tabel jurnal transaksi

Tabel jurnal sebagai sebuah blockchain memiliki arti sebagai tabel jurnal sebagai rantai rekord atau rantai catatan. Dimana setiap rekord digambarkan sebagai sebuah blok dan setiap blok ke blok selanjutnya atau rekord selanjutnya membangun suatu rantai.

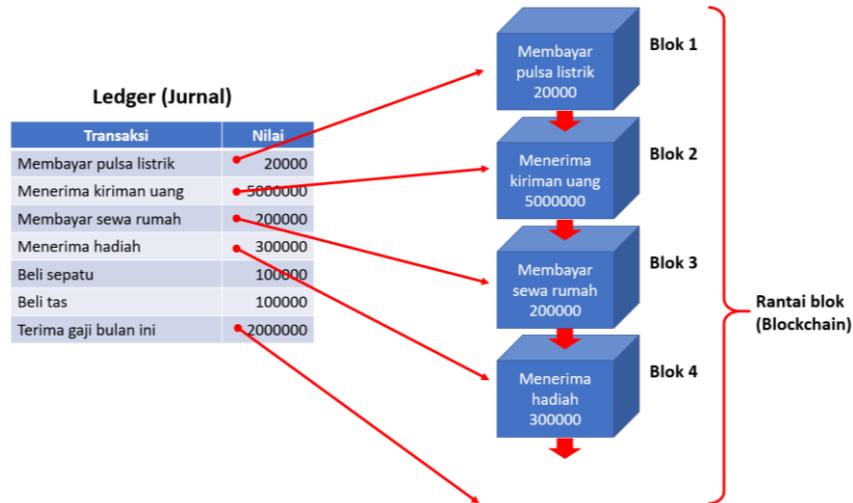
Pandangan sederhana tentang sebuah jurnal menjadi sebuah blockchain digambarkan oleh gambar 8. Gambar 8 memberikan ilustrasi yang simpel bagaimana membangun sebuah blockchain dari sebuah tabel jurnal.

Gambar 8 menunjukkan bahwa blok 1 mewakili rekord 1 dari tabel jurnal, blok 2 mewakili rekord 2, blok 3 mewakili rekord 3 dan seterusnya secara berturut-turut. Sehingga secara utuh, sebuah tabel jurnal direpresentasikan oleh sebuah rantai blok (block chain, ditulis blockchain).

Setiap penambahan sebuah blok di dalam blockchain memiliki arti sebagai penulisan transaksi baru di dalam tabel jurnal atau penambahan rekord baru di dalam tabel jurnal.

Penulisan rekord di dalam tabel jurnal seperti pada gambar 7 adalah penulisan yang sederhana. Di dalam kenyataanya, rekord boleh jadi tidak saja memiliki 2 kolom tetapi boleh saja 3 kolom, 4 kolom atau 5 kolom dan seterusnya. Sesuai dengan jenis transaksi yang hendak ditulis atau jenis buku jurnal yang hendak dibuat. Sebagai contoh seperti buku jurnal penjualan toko, buku jurnal gudang, buku jurnal data pasien di rumah sakit dan sebagainya. Demikian juga berarti cara blok di buat di dalam blockchain. Sebuah blok bisa jadi terdiri dari 2 baris catatan saja, 3 baris catatan saja, 4 baris catatan saja dan seterusnya.

Isi blok ini serupa dengan sebuah daftar atau *list*. Daftarnya boleh hanya ada 2 item, 3 item, 4 item dan seterusnya. Item pada blok menunjukkan kolom pada tabel jurnal.



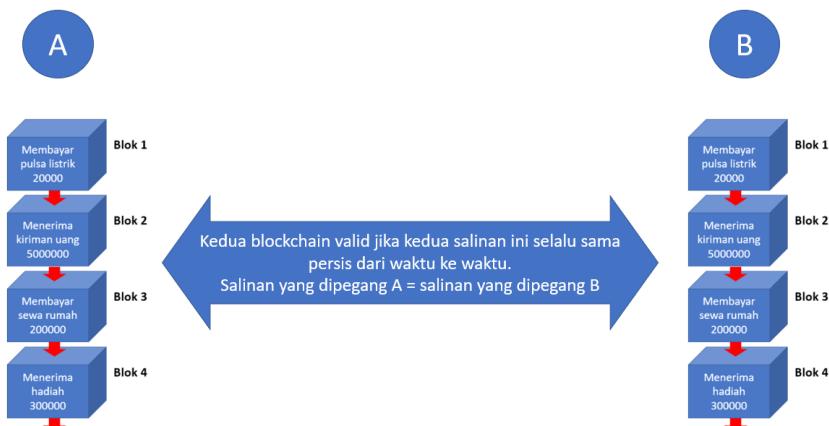
Gambar 8. Jurnal transaksi yang dinyatakan sebagai blockchain

Filosofi konsep keamanan dari blockchain dapat kita runut atau terinspirasi dari mekanisme validasi pencatatan pada buku jurnal. Seperti misalnya bahwa sebuah buku jurnal terbagi atas kolom rugi dan kolom laba atau kolom debit dan kolom kredit. Sebuah tabel jurnal adalah valid atau memiliki pencatatan yang valid jika hanya jika jumlah nilai pada kolom debit adalah sama dengan jumlah nilai pada kolom kredit. Gambar 9 memberi ilustrasi tentang tabel jurnal yang pencatatannya setimbang.

Transaksi	Debit	Kredit
Membayar pulsa listrik		20000
Membayar sewa rumah		200000
Beli sepatu		100000
Beli tas		100000
Ikhtisar rugi/laba	420000	
Menerima kiriman uang	5000000	
Menerima hadiah	300000	
Terima gaji bulan ini	2000000	
Ikhtisar rugi/laba		7300000
Jumlah	7720000	7720000

Gambar 9. Tabel jurnal dengan kolom debit dan kredit yang jumlahnya sama

Untuk memimik atau meniru cara memvalidasi ini, blockchain tidak dibagi ke dalam blok debit atau blok kredit. Tetapi blockchain dibuat di dalam dua salinan yang sama persis dan diletakkan pada dua lokasi yang berbeda. Blockchain dikatakan valid jika hanya jika salinannya menyatakan dia valid atau blockchain itu tetap sama persis dengan salinannya di dalam masa yang berjalan.



Gambar 10. Dua salinan blockchain yang sama persis, analogi dengan kolom debit dan kolom kredit yang jumlah nilainya sama

Kemudian konsep keamanan ini diperluas dengan ide distribusi. Yaitu dengan membuat salinan blockchain tidak saja dalam 2 lokasi yang berbeda yang dipegang oleh 2 orang yang berbeda. Tetapi disalin dan dipegang oleh 3, 4, 5, 6, 7,..., 1000,...,3000,...dan seterusnya lokasi dan orang yang berbeda.

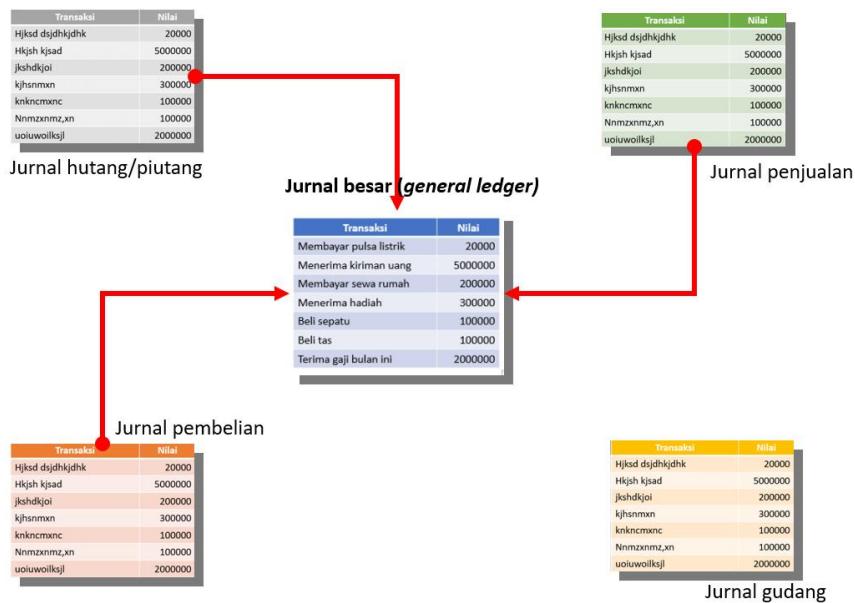
Sehingga gagasan validasi blockchain menjadi sebuah blockchain adalah valid jika hanya jika semua salinannya (atau paling tidak 51% salinan yang berbeda) menyatakan diri sama persis dengan blockchain tersebut.

Dari sinilah cara pandang lain tentang jurnal muncul. Yaitu gagasan jurnal terdistribusi (*distributed ledger*) dimana setiap penambahan blok, validasi, konfirmasi dilakukan secara terdistribusi. Ini berbeda dengan mekanisme buku jurnal tradisional yang sifatnya terpusat. Yaitu disana ada beberapa buku jurnal kecil yang saling berbeda, kemudian berkumpul membangun sebuah buku jurnal besar. Misal buku jurnal penjualan, buku jurnal gudang, buku jurnal transportasi, buku jurnal pembelian, buku jurnal pembayaran gaji pegawai dan sebagainya, disatukan secara terpusat pada sebuah buku jurnal besar yang merangkum seluruh buku jurnal tersebut.

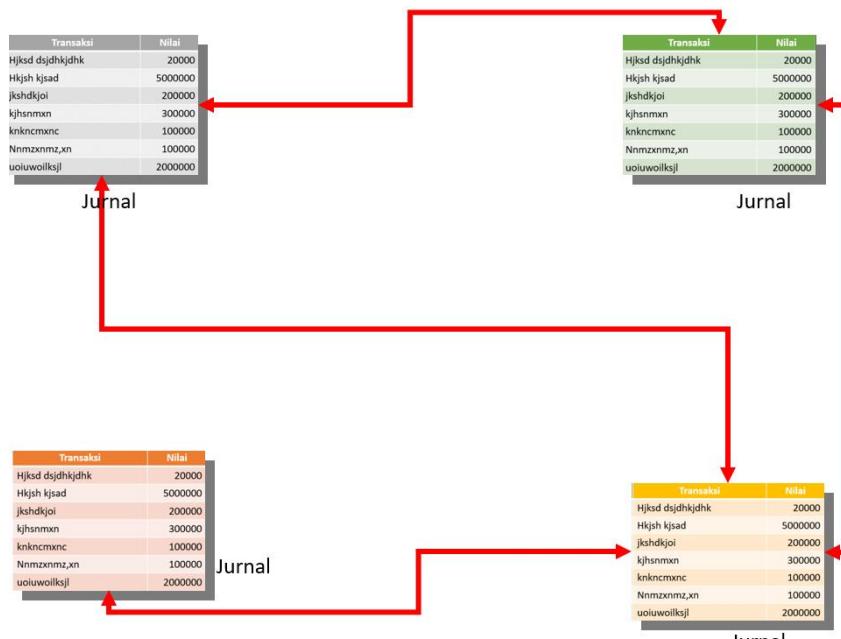
Gambar 10 menyajikan ilustrasi bagaimana buku jurnal besar menjadi pusat jurnal bagi semua buku jurnal yang lain. Gambar 10 ini mengilustrasikan sifat jurnal tradisional yang terpusat.

Gambar 11 menyajikan ilustrasi jurnal yang terdistribusi. Dalam pengertian ini, pencatatan transaksi ke sebuah jurnal adalah juga berarti pencatatan transaksi yang sama di semua jurnal yang lain. Sehingga pencatatan transaksi menjadi terdistribusi. Seluruh jurnal adalah jurnal besar atau *general ledger*. Yaitu berada dalam kedudukan yang setara dengan jurnal yang lain.

Jurnal terdistribusi ini menjadi dasar filosofi bagi arsitektur blockchain. Blockchain adalah dapat dilihat sebagai *general ledger* yang terdistribusi.

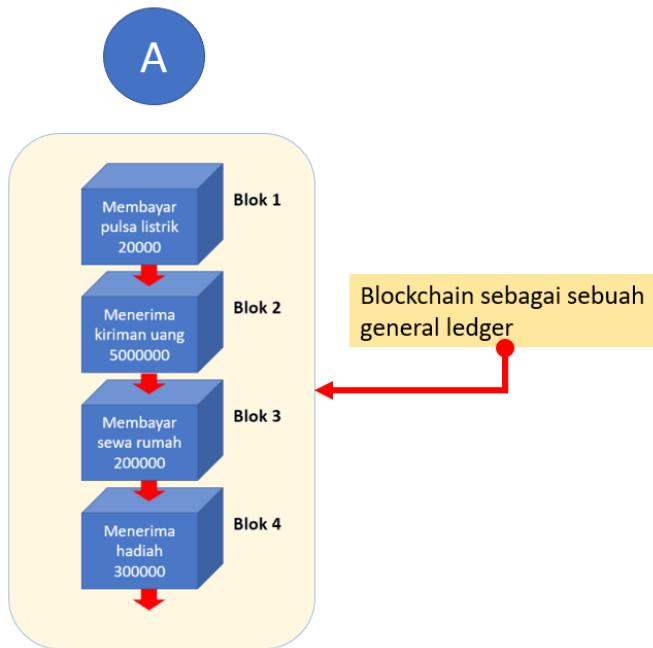


Gambar 11. Contoh jurnal terpusat



Gambar 12. Contoh jurnal terdistribusi

Blockchain sendiri adalah sebuah buku jurnal besar (*general ledger*), karena dia tidak terpisah-pisah dalam beberapa blockchain kecil. Tetapi dia sendiri adalah seluruhnya jurnal tersebut. Merangkum keseluruhan transaksi. Dimana setiap orang dari berbagai bidang boleh menambahkan blok ke dalam blockchain. Selain itu dia juga terdistribusi yaitu bahwa setiap orang pemegang salinan blockchain adalah memegang salinan sempurna blockchain keseluruhan, atau memegang salinan *general ledger* yang sama dengan yang lain.



Gambar 13. A sebagai pemegang salinan blockchain (*general ledger*)

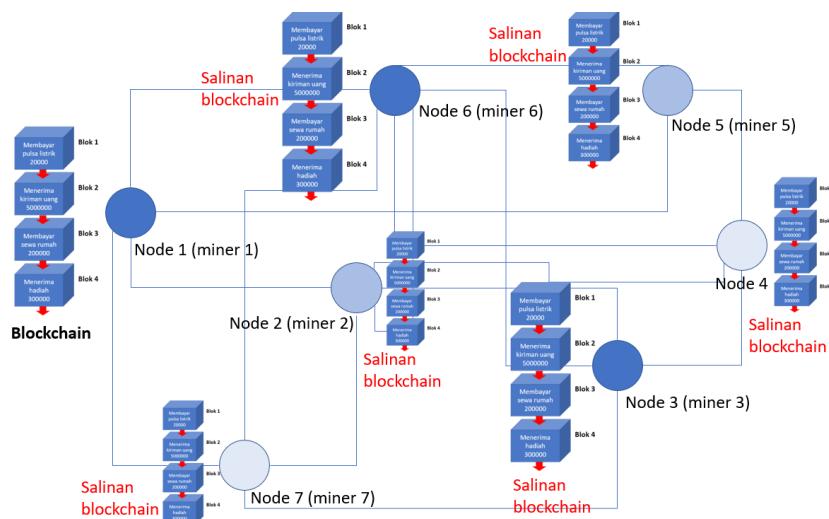
Jalinan semua *general ledger* yang terdistribusi membangun jaringan *general ledger* membangun sebuah *hyperledger fabric*. Yaitu semua blockchain yang membangun jaringan salinan blockchain dari node ke node. Gambar 14 menyajikan ilustrasi sebuah *hyperledger fabric* atau sebuah jejala (*network*) blockchain.

Setiap node di dalam *hyperledger fabric* adalah pemegang salinan blockchain. Dia memegang salinan yang sempurna dan selalu diperbarui menurut protokol konsensus jejala blockchain. Di dalam

protokol konsensus untuk menjalankan blockchain standar, setiap node akan berusaha menyalin atau menerima salinan blockchain dengan rantai terpanjang. Karena dalam protokol itu, rantai terpanjang adalah rantai yang paling valid diantara semua rantai blockchain.

Misalkan saja sebuah node terputus dari jalur blockchain, maka pada saat dia tersambung atau terkoneksi lagi dengan jaringan blockchain, maka dia mungkin memegang salinan yang sudah kadaluarsa (*outdated*) dimana node-node yang lain telah memegang rantai terpanjang sedangkan dia masih memegang salinan blockchain lama yang rantainya lebih pendek. Karena itu, node yang baru bergabung lagi tersebut harus memperbarui salinan blockchainnya.

Terdapat penjelasan yang lebih rinci, mengapa rantai terpanjang adalah selalu merupakan yang tervalid diantara semua rantai blockchain. Ini berkaitan dengan penambahan blok oleh sebuah node yang berhak. Penambahan blok ini bagi sebuah node bukanlah sesuatu yang sederhana hanya dengan tinggal menambahkan sebuah transaksi baru untuk menjadi blok baru. Akan tetapi pada node tersebut mungkin ada 2,3,4, dan seterusnya jumlah transaksi yang lebih dari 1 yang antri untuk ditambahkan oleh node tersebut. ini disebut sebagai sejumlah transaksi yang tersimpan di pool (memori penampung transaksi yang masuk) dalam keadaan antri menunggu untuk ditambahkan sebagai blok baru di dalam blockchain. Ini menjadi lebih rumit lagi manakala semua node juga demikian, sehingga mungkin saja terjadi dalam selang waktu yang sangat pendek terdapat 2 atau lebih node yang berusaha menambahkan blok tetapi dengan transaksi yang berbeda-beda karena mereka memilih transaksi yang berbeda pada pool mereka masing-masing. Untuk mengatasi masalah ini, protokol sedemikian rupa dibuat untuk lebih memilih rantai terpanjang blockchain sebagai yang paling valid.



Gambar 14. Distribusi salinan blockchain membangun jaringan blockchain atau *hyperledger fabric*

3. Blockchain sebagai sistem basisdata terdistribusi

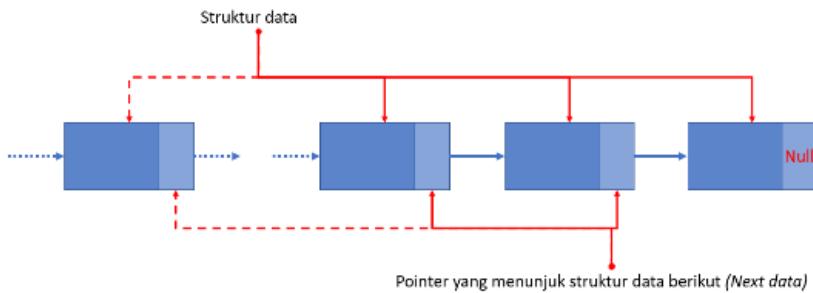
Seperi dijelaskan paling awal, blockchain pada dasarnya dapat dilihat sebagai sebuah tabel atau secara umum sebagai rantai struktur data. Karena itulah sebuah blockchain dapat kita lihat saja sebagai sebuah basisdata. Yaitu sebuah basisdata yang memiliki sifat keamanan yang khas berbeda dengan basisdata biasa atau standar.

Akan tetapi karena sifatnya yang terdistribusi dan terenkripsi maka blockchain adalah sebuah sistem basisdata terdistribusi terenkripsi. Walau dia termasuk sebagai sistem basisdata terdistribusi, blockchain memiliki sejumlah protokol sendiri terkait dengan penggunaan hash dan enkripsi yang sangat berbeda dengan sistem basisdata terdistribusi pada umumnya.

Blok di dalam blockchain adalah struktur data, sehingga secara keseluruhan blockchain adalah rantai struktur data. Kita dapat membayangkan ini sebagaimana konsep pointer di dalam berbagai bahasa pemrograman, seperti konsep *linked list*. *Linked list* secara keseluruhan adalah struktur data, akan tetapi setiap item yang di *list* olehnya dapat dilihat juga sebagai struktur data, sehingga dia menjalin

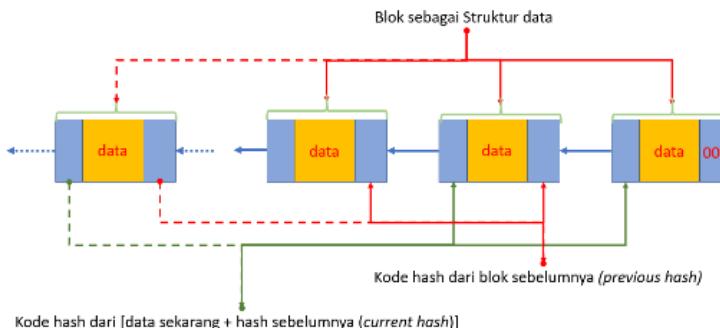
sebuah daftar yang menghubungkan antar struktur data.

Gambar 15 menyajikan ilustrasi dari sebuah struktur data *linked list*.



Gambar 15. Contoh sebuah struktur data *linked list*

Jika dibawa kepada pengertian blockchain, maka sebuah blockchain dapat dilihat sebagai sebuah struktur data *linked list*. Setiap bloknya adalah sebuah struktur data juga tetapi memuat kunci pengaman yang disebut hash. Gambar 16 memberikan ilustrasi bagaimana jika blockchain disajikan menyerupai sebuah *linked list*.



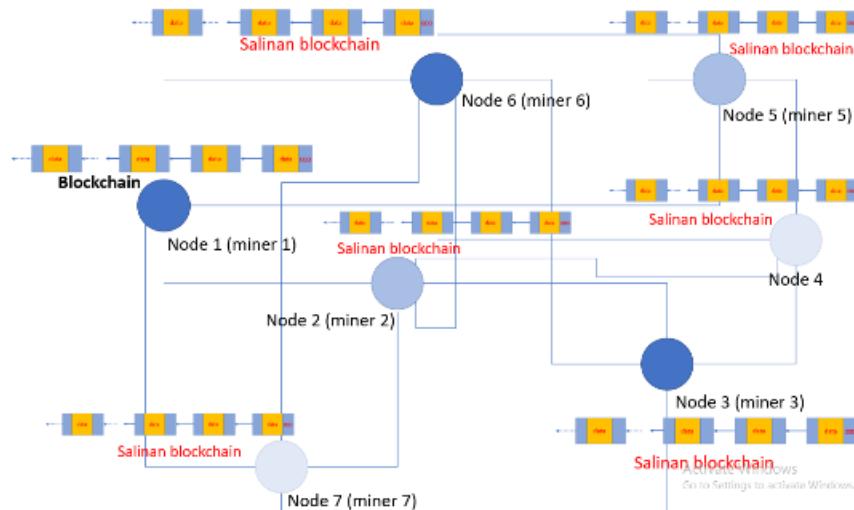
Gambar 16. Blockchain sebagai struktur data yang menyerupai *linked list*

Dengan demikian, sebuah jaringan atau jejala blockchain dapat dilihat di dalam cara pandang jejala struktur data yang menyerupai struktur data *linked list*. Gambar 17 menyajikan ilustrasi jaringan blockchain yang dilihat sebagai jejala struktur data *linked list*.

Gambar 17 juga mengilustrasikan sebuah struktur data yang terdistribusi. Karena sebuah struktur data dapat dikonstruksikan hingga membentuk sebuah basisdata, yaitu memiliki bahasa *query* sendiri dan antarmuka sistem manajemen basisdata (*database management system, DBMS*), maka gambar 17 juga mengilustrasikan sebuah sistem

basisdata terdistribusi. Jika kita melihat *field* (kolom) hash pada struktur data sebagai pointer, berserta segala mekanisme enkripsinya yang bisa ditambahkan, maka sistem basisdata terdistribusi ini dilihat sebagai sistem basisdata terdistribusi terenkripsi.

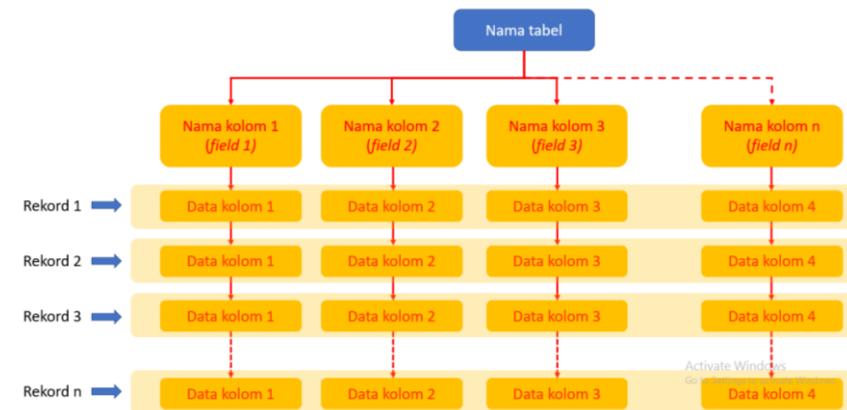
Di dalam cara pandang ini, setiap penambahan blok di dalam blockchain pada hakikatnya adalah penambahan data di dalam *linked list*.



Gambar 17. Jejala blockchain dilihat sebagai jejala *linked list*

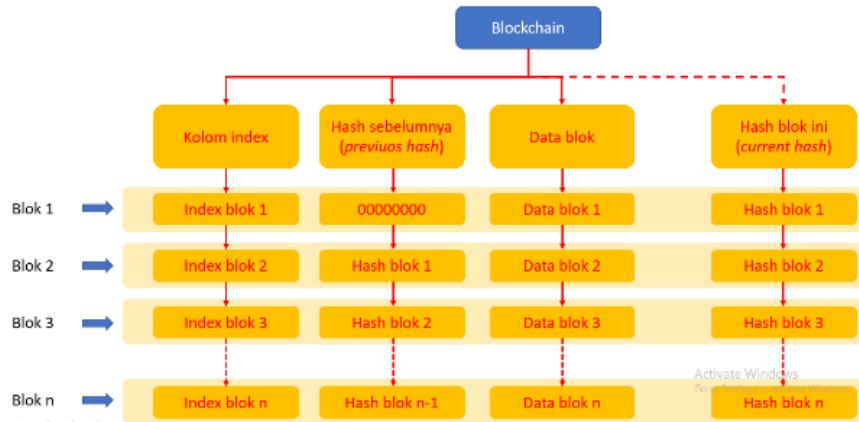
Pada implementasi struktur data, sebuah *linked list* atau sekumpulan *linked list* dapat digunakan untuk merepresentasikan sebuah tabel. Sehingga sebuah *linked list* menyajikan sebuah pertalian rekord yang membangun sebuah tabel. Sebenarnya tidak terbatas pada struktur *linked list*. Sebuah struktur pohon juga bisa merepresentasikan sebuah tabel. Ini juga berarti sebuah blockchain dapat disajikan dalam sebuah graf pohon sebagaimana tabel dapat disajikan ke dalam sebuah graf pohon.

Gambar 18 menyajikan bagaimana jika tabel disajikan ke dalam struktur data pohon.



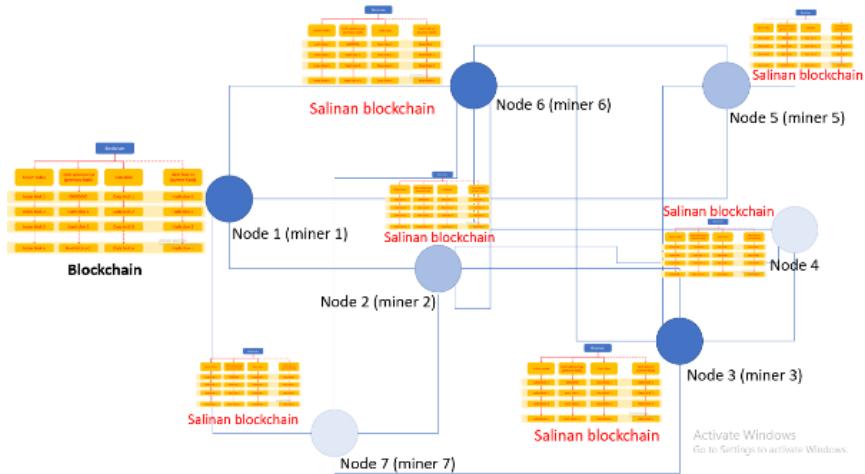
Gambar 18. Contoh penyajian tabel ke dalam graf pohon (tree)

Demikian pula bahwa kita dapat menyajikan blockchain sebagai sebuah struktur data pohon. Gambar 19 mengilustrasikan bagaimana jika blockchain disajikan dalam struktur data pohon.



Gambar 19. Contoh blockchain yang disajikan dalam struktur data pohon

Selanjutnya kita dapat membuat sebuah cara pandang (*view*) tentang bagaimana jaringan blockchain jika dilihat sebagai sebuah jaringan struktur data pohon. Gambar 20 memberikan ilustrasi tentang bagaimana jika struktur pohon menjadi terdistribusi membangun sebuah jaringan blockchain.



Gambar 20. Contoh jejala blockchain dalam cara pandang graf pohon

Sebuah struktur data pohon atau sebuah struktur data *linked list* atau kumpulannya dapat dikumpulkan dalam sebuah sistem yang memiliki relasi satu sama lain atau tanpa relasi. Sistem ini memiliki bahasa *query* sendiri dan memiliki antarmuka DBMS. Sistem membangun apa yang kita sebut sebagai sebuah sistem basisdata. Sama halnya seperti sistem basisdata yang diproduksi oleh berbagai vendor perusahaan teknologi seperti MySQL atau Oracle oleh perusahaan Oracle, atau SQL Microsoft Server, atau MariaDB, atau berbagai sistem basisdata relasional dan non-relasional yang ada di pasaran saat ini.

Semua struktur data ini, baik *linked list*, graf pohon dan sebaginya, dapat diisi dengan struktur data yang *structured* atau *unstructured*.

a) Sebagai struktur data yang terstruktur (*structured data*)

Kita dapat melihat keseluruhan blockchain adalah struktur data yang terstruktur manakala seluruh blok memiliki definisi *field* yang sama. Misal masing-masing blok memiliki kolom indeks, kolom hash sebelumnya, kolom data, kolom hash blok itu sendiri. Tidak ada yang berbeda. Mereka memiliki jumlah kolom yang sama. Ini berarti blockchain adalah sebuah *structured data*.

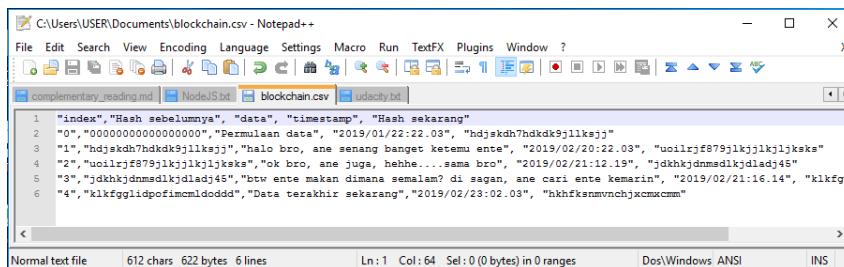
Karena sifat ini, kita dapat membangun blockchain di atas

sistem basisdata yang sudah ada. Yaitu bahwa kita dapat membangun blockchain di atas MySQL, Oracle dan SQL Microsoft Server dan sebagainya. Lalu memanfaatkan teknologi sistem basisdata terdistribusi oleh vendor-vendor tersebut untuk membangun jaringan blockchain. Gambar 21 memberikan sebuah contoh blockchain yang *structured data*.

Ind ex	Hash sebelumnya	Data	Timestamp	Hash sekarang	
0.	00000000000000000000	Permulaan data	2019/01/22:22.03	hdjskdh7hdksk9jllksjj	← Blok 1
1.	hdjskdh7hdksk9jllksjj	halo bro, ane senang banget ketemu ente	2019/02/20:22.03	uoilrjf879jlkjkljksks	← Blok 2
2.	uoilrjf879jlkjkljksks	ok bro, ane juga hehhe....sama bro	2019/02/21:12.19	jdkhkjdndmsdlkjdladj45	← Blok 3
3.	jdkhkjdndmsdlkjdladj45	btw ente makan dimana semalam? di sagan ane cari ente kemarin	2019/02/21:16.14	kikfggldpofimcmldoddd	← Blok 4
4.	kikfggldpofimcmldoddd	Data terakhir sekarang	2019/02/23:02.03	hkhfksnmvnchjxcmxcm	← Blok 5

Gambar 21. Contoh blockchain yang ditulis dalam bentuk *structured data*

Akan tetapi, kita juga dapat membangun blockchain tanpa semua sistem basisdata itu. Kita hanya perlu membuat file text biasa misal dengan format CSV atau SQL yang menyimpan rekord dengan jumlah kolom yang sama dengan rekord yang lain, kemudian kita dapat membuat algoritma-algoritma untuk melakukan query serta protokol-protokol keamanan yang mendukung blockchain yang kita bangun. Lalu membuat file itu terdistribusi di seluruh *node* blockchain.



```

C:\Users\USER\Documents\blockchain.csv - Notepad++
File Edit Search View Encoding Language Settings Macro Run TextFX Plugins Window ?
File NodeStxt blockchain.csv uaditya.txt
complementary_reading.mil
1 "index", "Hash sebelumnya", "data", "timestamp", "Hash sekarang"
2 "0", "0000000000000000", "Permulaan data", "2019/01/22:22.03", "hdjskdh7hdksk9jllksjj"
3 "1", "hdjskdh7hdksk9jllksjj", "halo bro, ane senang banget ketemu ente", "2019/02/20:22.03", "uoilrjf879jlkjkljksks"
4 "2", "uoilrjf879jlkjkljksks", "ok bro, ane juga, hehhe....sama bro", "2019/02/21:12.19", "jdkhkjdndmsdlkjdladj45"
5 "3", "jdkhkjdndmsdlkjdladj45", "btw ente makan dimana semalam? di sagan, ane cari ente kemarin", "2019/02/21:16.14", "kikfggldpofimcmldoddd"
6 "4", "kikfggldpofimcmldoddd", "Data terakhir sekarang", "2019/02/23:02.03", "hkhfksnmvnchjxcmxcm"

```

Gambar 22. Contoh blockchain yang ditulis ke dalam file csv

b) Sebagai struktur data yang tak terstruktur (*unstructured data*)

Sebuah pertanyaan mungkin dapat kita kemukakan. Yaitu apakah kita dapat membuat blok di dalam blockchain dengan definisi *field* yang berbeda-beda secara dinamis?

Ini seumpama membuat pertanyaan bahwa apakah kita bisa

menulis sebuah blockchain sebagai sebuah file XML saja atau file JSON saja atau file Graf saja?

Hal seperti ini tidaklah membatasi arsitektur sebuah blockchain. Kita dapat saja membuat blockchain dalam bentuk *unstructured data*. Tetapi itu berarti kita mungkin hanya perlu membangun sejumlah protokol yang berbeda dengan blockchain yang dibangun secara *structured data*.

Gambar 23 mengilustrasikan sebuah blockchain yang dinyatakan ke dalam sebuah file XML.

```
<blockchain>
  <block> <index>1</index>
    <hash_sebelumnya>hdjskdh7hdkdk9jllksjj</hash_sebelumnya>
    <data>halo bro, ane senang banget ketemu ente</data>
    <timestamp>2019/02/20:22.03</timestamp>
    <hash_sekarang>uoilrjf879jlkjllksks</hash_sekarang>
  </block>
  <block> <index>2</index>
    <hash_sebelumnya> uoilrjf879jlkjllksks</hash_sebelumnya>
    <data>ok bro, ane juga</data>
    <data-balasan>hehhe....sama bro</data-balasan>
    <timestamp>2019/02/21:12.19</timestamp>
    <hash_sekarang>jdkhkjdhnmsdlkjldadj45</hash_sekarang>
  </block>
  <block> <index>3</index>
    <hash_sebelumnya> jdkhkjdhnmsdlkjldadj45</hash_sebelumnya>
    <data>btw ente makan dimana semalam?</data>
    <data-balasan>di sagan</data-balasan>
    <data-jawab>ane cari ente kemarin</data-jawab>
    <timestamp>2019/02/21:16.14</timestamp>
    <hash_sekarang>klkfsglidpofimcmldoddd</hash_sekarang>
  </block>
</blockchain>
```

Gambar 23. Contoh blockchain yang ditulis ke dalam XML

Di dalam cara yang sama, sebuah blockchain dapat juga ditulis di dalam bentuk objek JSON. Gambar 24 memberikan ilustrasi bagaimana jika blockchain di tulis sebagai objek JSON.

```
{
  "nama": "blockchain",
  "node": "adil123",
  "blok-1": {
    "index": 1,
    "hash-sebelumnya": "hdjskdh7hdhdk9jllksjj",
    "data": "halo bro, ane senang banget ketemu ente",
    "timestamp": "2019/02/20:22.03",
    "hash-sekarang": "uoilrjf879jlkjjlkjljksks"
  },
  "blok-2": {
    "index": 2,
    "hash-sebelumnya": "uoilrjf879jlkjjlkjljksks",
    "data-1": "ok bro, ane juga",
    "data-2": "hehhe....sama bro",
    "timestamp": "2019/02/21:12.19",
    "hash-sekarang": "jdhkjdnmsdlkjdladj45"
  },
  "block-3": {
    "index": 3,
    "hash-sebelumnya": "jdhkjdnmsdlkjdladj45",
    "data-1": "btw ente makan dimana semalam",
    "data-2": "di sagan",
    "data-3": "ane cari ente kemarin",
    "timestamp": "2019/02/21:16.14",
    "hash-sekarang": "klkfegglidpofimcmldoddd"
  }
}
```

Gambar 24. Contoh blockchain yang ditulis dalam bentuk JSON

4. Blockchain sebagai sebuah objek terdistribusi

Orang pada dasarnya dapat memperluas konsep blockchain, sehingga tidak saja berarti sebuah struktur data yang hanya berisi data dan kode hash. Akan tetapi diperluas menjadi sebuah struktur yang berupa objek. Ini berarti, blok sebagai sebuah objek di dalam blockchain, tidak hanya berisi data dan kode hash, tetapi dapat bersama dengan kode program (analogi dengan *method* di dalam bahasa pemrograman berbasis objek, OOP).

Ini berarti jika selama ini kita melihat blockchain sebagai jejal data terenkripsi, maka pada konteks ini, kita dapat melihat blockchain sebagai jejal data dan kode program yang terenkripsi.

5. Blockchain sebagai agen terdistribusi

Lebih dari sekedar objek. Jika kita menyatakan blok sebagai sebuah agen, maka tidak saja bahwa blok itu berisi dengan data dan kode program, akan tetapi dia juga dapat berkedudukan sebagai sebuah agen yang independen dengan aturan-aturannya sendiri tetapi dapat berkomunikasi dengan agen-agen lain. Atau sebuah blok yang

independen dengan aturan-aturan di dalam dirinya sendiri tetapi juga dapat berkomunikasi dengan blok-blok lain. Terutama berkomunikasi membangun sebuah blockchain.

Pada konteks ini, kita boleh melihat blockchain sebagai sebuah kumpulan agen, sehingga kita bisa melihatnya sebagai sebuah *swarm* agen.

6. Perbedaan pengertian tentang wallet, node, peer, fullnode, miner dan masternode di dalam arsitektur blockchain

Blockchain secara fisik berjalan (*running*) di atas seluruh jejala node yang ada di internet. Yaitu semua jejala node yang dimilikinya. Akan tetapi terdapat beberapa filosofi yang mendeskripsikan perbedaan yang terjadi diantara node itu sendiri, dimulai pada aplikasi *wallet* yang dapat dipegang oleh siapa saja walaupun dia bukan node di dalam blockchain.

Pada keseluruhannya, pemilik aplikasi *wallet* (dompet bitcoin dan sebagainya) adalah siapa saja orang baik yang datang dari eksternal sistem blockchain ataupun yang datang dari internal sistem blockchain yang dapat memanfaatkan blockchain melalui aplikasi *wallet* tersebut. ini adalah posisi orang kebanyakan yang tidak perlu tau dan terlibat ke dalam arsitektur blockchain. Dengan memiliki *wallet* dia dapat melakukan transaksi di atas blockchain.

Istilah node adalah sebuah istilah umum untuk semua orang yang memegang server komputer, baik besar atau kecil, serta menyimpan salinan blockchain baik seluruh salinan atau sebagian di komputernya. Secara umum, sebuah node melaksanakan tugas di dalam blockchain sebagai berikut:

1. Setiap node memeriksa jika sebuah blok transaksi adalah valid sehingga dia bisa mebgusulkan penerimaan atau penolakan.
2. Setiap node menyimpan blok-blok transaksi, sebagai sebuah salinan blockchain.
3. Setiap node melakukan *broadcast* ke beberapa node yang lain yang mungkin membutuhkan sinkronisasi dengan blockchain.

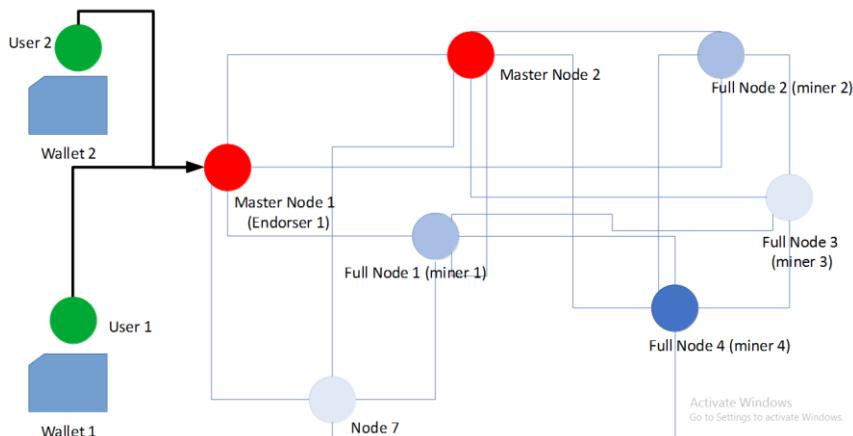
Peer adalah node dalam pengertian sebagai bagian dari titik di dalam jaringan, apapun posisinya di dalam arsitektur blockchain.

Endorser adalah node yang mengusulkan sebuah transaksi untuk ditambahkan ke blockchain kemudian membroadcast ke seluruh node untuk ditambahkan sebagai antrian di *pool* masing-masing node agar mereka bisa menambahkan ke blockchain.

Fullnode adalah istilah untuk sebuah node yang menyimpan secara penuh seluruh salinan blockchain.

Miner adalah istilah untuk sebuah node yang selain berkedudukan sebagai *fullnode*, dia juga dapat melakukan penambahan blok di dalam blockchain dan melakukan verifikasi penambahan blok dari node yang lain.

Master Node selain berkedudukan sebagai miner, dia juga memiliki kekuasaan untuk mengeksekusi protokol-protokol global blockchain.



Gambar 25. Arsitektur blockchain dengan berbagai jenis node di dalamnya

Gambar 25 memberikan ilustrasi bagaimana semua jenis-jenis node itu membangun arsitektur blockchain keseluruhan.

Akan tetapi, ini hanyalah satu jenis arsitektur. Untuk arsitektur yang lain dengan protokol konsensus yang berbeda mendefinisikan sendiri jenis-jenis node nya sesuai dengan tugas mereka di dalam arsitektur blockchain keseluruhan.

Sumber bacaan tambahan:

<https://medium.com/coinmonks/blockchain-what-is-a-node-or-masternode-and-what-does-it-do-4d9a4200938f>

7. Bagaimana mekanisme blockchain mencegah penggunaan dua kali uang crypto (Double Spending)

Kasus double spending terjadi dalam skenario jika seseorang hendak berbuat curang untuk menggunakan dua kali atau lebih uang crypto yang dia miliki dimana semestinya uang crypto bersifat hanya digunakan untuk satu transaksi.

Di dalam konteks umum sehari-hari, *double spending* adalah seperti memiliki uang kertas Rp. 5000,- lalu digunakan untuk membeli sekotak permen yang harganya pas Rp. 5000,-. Akan tetapi, setelah seakan-akan anda membayar pemilik kotak permen, ketika kasir menerima uang anda dan meletakkan ke laci kasir, tanpa sepengetahuan kasir, anda mengambil kembali uang tersebut dan menggunakan lagi uang tersebut untuk membeli sekotak kue yang juga harganya pas Rp. 5000,-.

Dalam hal ini, uang Rp. 5000,- milik anda digunakan untuk melakukan 2 kali pembayaran (*double spending*) untuk transaksi membeli barang yang berbeda yang masing-masing harganya Rp. 5000,-.

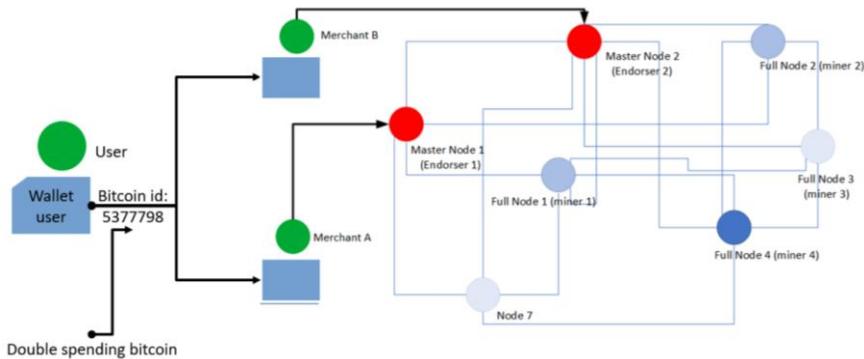
Ini membuat kerugian disisi merchant atau penjual, dan bagi blockchain sendiri menyebabkan runtuhnya kepercayaan publik kepada keamanan blockchain.

Untuk memecahkan masalah ini, arsitektur blockchain menerapkan protokol sebagai berikut:

Misalkan anda melakukan transaksi uang crypto dan berniat melakukan kegiatan *double spending*. Anda lalu melakukan transaksi pada sebuah *merchant A* dengan menggunakan bitcoin, mengirim ke alamat dompet bitcoin *merchant A*. Lalu pada saat yang sama anda membelanjakan lagi bitcoin yang sama ke *merchant B* dengan mengirim bitcoin tersebut ke alamat dompet *B*. Sehingga terjadi transaksi yang *double spending*.

Gambar 26 memberikan ilustrasi bagaimana percobaan *double*

spending oleh user dapat dilakukan dengan mengirim bitcoin yang sama ke dua merchant yang berbeda.



Gambar 26. Skema double spending pada blockchain

Akan terjadi dua kasus yang berbeda ketika transaksi A dan B dikirim ke blockchain untuk ditambahkan. Pertama adalah ketika transaksi A atau B salah satunya lebih dulu ditambahkan ke blockchain dibanding yang lain. Kedua jika transaksi A dan B bersamaan ditambahkan ke dalam blockchain oleh dua node yang berbeda.

Kasus pertama:

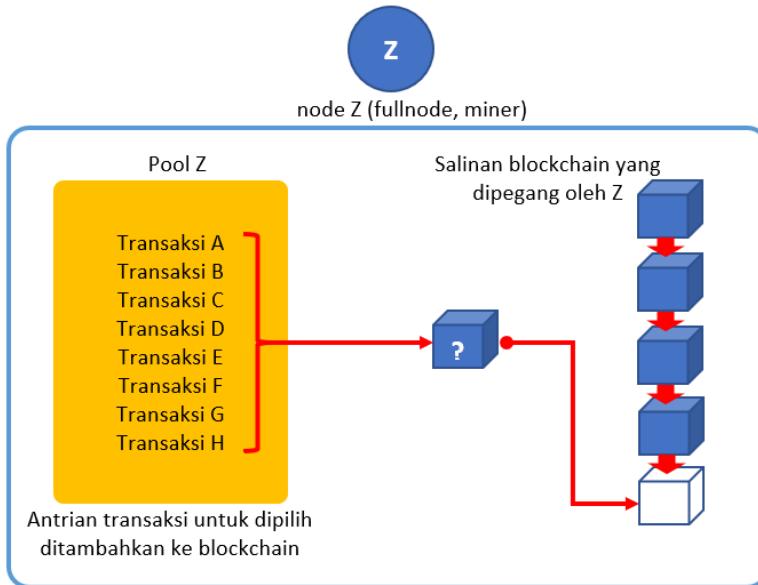
Transaksi A anda dan transaksi B akan dikirim ke semua node dari blockchain, transaksi A dan B ini lalu antri pada setiap *pool* (memori penampung semua transaksi yang datang untuk ditambahkan ke blockchain) dari node di dalam blockchain.

Gambar 27 memberikan ilustrasi bagaimana sebuah node menyimpan semua transaksi yang antri untuk ditambahkan ke blockchain. Untuk suatu arsitektur blockchain tertentu, mungkin disana ada aturan atau protokol yang merangkin derajat kepentingan dari tiap-tiap transaksi kemudian berdasarkan derajat itu, node menambahkan yang paling penting pertama kali.

Protokol untuk memilih transaksi mana yang terpenting boleh jadi mengimplementasikan sistem pendukung keputusan di dalam logikanya sehingga dapat memberi bobot kepada setiap transaksi dan melakukan perangkingan derajat kepentingan.

Setelah pemilihan transaksi dilakukan sedemikain sehingga transaksi A

terpilih dan ditambahkan ke blockchain. Yang berarti ada node yang berhasil menambahkan transaksi A ke blockchain, maka node tersebut akan mengumumkan bahwa keberhasilan itu ke seluruh node, dan menyatakan bahwa transaksi A telah ditambahkan ke blockchain.



Gambar 27. Node Z yang memiliki pool dimana menampung antrian transaksi untuk ditambahkan ke blockchain

Pengumuman Ini menyebabkan semua transaksi A dan semua transaksi yang sama atau identik bitcoin nya dengan transaksi A (yaitu transaksi B) akan dihapus di pool semua node tersebut sebagai dianggap telah selesai ditambahkan.

Ini berarti transaksi B dihapus dari semua *pool* dan ditolak untuk ditambahkan lagi.

Merchant A menerima konfirmasi penambahan transaksi A ke dalam blockchain, lalu menerima pembayaran menggunakan bitcoin tersebut sehingga transaksi A selesai.

Merchant B menerima konfirmasi penolakan atau tidak menerima sama sekali konfirmasi penambahan ke blockchain, sehingga *merchant B* akan menolak transaksi B yang berarti menolak pembayarn bitcoin anda.

Kasus kedua:

Transaksi A anda dan transaksi B akan dikirim ke semua node dari blockchain, transaksi A dan B ini lalu antri pada setiap *pool* (memori penampung semua transaksi yang datang untuk ditambahkan ke blockchain) dari node di dalam blockchain. Lalu kemudian ada dua node yang berhasil menambahkan transaksi A atau B ke blockchain, maka masing-masing node akan menambahkan transaksi A atau B ke blockchain.

Misal node X menambahkan transaksi A ke blockchain, dan node Y menambahkan transaksi B ke blockchain. Akan tetapi node X akan mengumumkan penambahan ini, sehingga node Y juga akan menambahkan transaksi A. Demikian juga sebaliknya, node Y akan mengumumkan penambahan transaksi B ke blockchain. Akan tetapi karena transaksi A dan B sama, maka blockchain menerapkan protokol bahwa dimana diantara kedua transaksi itu mendapatkan konfirmasi valid terbanyak dari populasi semua node di dalam blockchain maka transaksi itulah yang dinyatakan valid untuk ditambahkan ke blockchain. Transaksi yang kurang mencapat konfirmasi atau kurang cepata mendapat konfirmasi terbanyak akan diabaikan dan ditolak untuk ditambahkan ke blockchain.

Dengan demikian, *couble spending* pada kasus terjadinya penambahan secara bersamaan ke dalam blockchain oleh dua node yang berbeda dapat diatasi.

Sumber bacaan tambahan: <https://coinsutra.com/bitcoin-double-spending/>

8. Kemungkinan keberhasilan serangan double attack pada blockchain

Ada dua kemungkinan sehingga serangan double attack dapat berhasil dilakukan. Akan tetapi ini sulit dari sisi banyaknya sumber daya yang harus dikuasai atau besarnya probabilitas gagal.

Serangan pertama disebut sebagai 51% *attack* dan serangan kedua disebut *race attack*.

Kasus 51% *attack*:

Serangan ini dilakukan dengan menjadikan keseluruhan blockchain

menjadi seakan-akan menjadi private blockchain, yaitu dimana penyerang seakan menguasai public blockchain sehingga seluruh node tidak lagi menjadi independen satu sama lain sebagai bawaan sifat terdistribusi.

Ini dilakukan dengan menguasai paling sedikit 51% node (*miner*) yang ada. Akan tetapi ini sama saja sebagai usaha untuk menguasai sumber daya 51% dari jaringan blockchain yang mungkin terdiri dari ribuan atau jutaan node. Sehingga menjadi sulit untuk dilakukan.

Kasus *race attack*:

Pada kasus ini, merchant tidak berhati-hati dengan terlalu cepat menutup transaksi dengan user sebagai transaksi yang telah selesai sebelum menerima konfirmasi dari blockchain bahwa bitcoin yang dibayarkan dapat ditambahkan ke dalam blockchain atau bitcoin itu dikonfirmasi valid.

Misal user mengirim bitcoin yang sama ke merchant A dan B bersamaan. Sehingga mestilah salah satu dari A dan B ada yang menerima konfirmasi penolakan penambahan blok di blockchain atau bahwa bitcoin yang dikirim user adalah tidak valid. Misal A dan B tidak sabar menunggu hasil konfirmasi dari blockchain maka mestilah salah satu dari A dan B tertolak transaksinya di blockchain atau bahwa bitcoin yang mereka terima dianggap tidak valid.

Sumber bacaan tambahan: <https://coinsutra.com/bitcoin-double-spending/>

9. Arti confirmation di dalam blockchain terhadap transaksi yang berhasil ditambahkan sebagai block

Jika sebuah transaksi ditambahkan di dalam atau termasuk ke dalam sebuah blok yang berhasil ditambahkan ke blockchain, maka *confirmation* terhadap keabsahan transaksi ini adalah banyaknya blok baru yang ditambahkan setelahnya.

Misalkan blok yang memuat transaksi A memiliki nomor 302, kemudian dalam proses di blockchain, terjadi penambahan block terhadap salinan blockchain di seluruh node sehingga mencapai blok nomor 509, maka *confirmation* terhadap transaksi A adalah sebanyak $509 - 309 = 200$ konfirmasi.

Ini berarti, transaksi A bernilai 200 konfirmasi keabsahan di atas blockchain. Pada kasus *double spending* sebelumnya, misalkan transaksi A dan transaksi B bersamaan di masukkan di dalam 2 blok berbeda yang berhasil ditambahkan di dalam jaringan blockchain, maka transaksi yang paling valid adalah transaksi memiliki nilai konfirmasi terbanyak. Sumber bacaan tambahan: <https://medium.com/coinmonks/how-a-miner-adds-transactions-to-the-blockchain-in-seven-steps-856053271476>

10. Tentang bagaimana proses penambahan blok di dalam blockchain terjadi

Tata cara penambahan ini memiliki protokol yang berbeda-beda bagi setiap arsitektur blockchain. Arsitektur blockchain awal menggunakan protokol *proof of work* (PoW) tetapi arsitektur blockchain yang lain menggunakan protokol *proof of stake* (PoS) atau arsitektur lain lagi menggunakan *delegate proof of stake* (DPoS) atau mereka menggunakan kombinasinya, dan sebagainya beberapa protokol lain lagi yang mungkin.

Kita meninjau untuk arsitektur blockchain awal yang menggunakan protokol PoW.

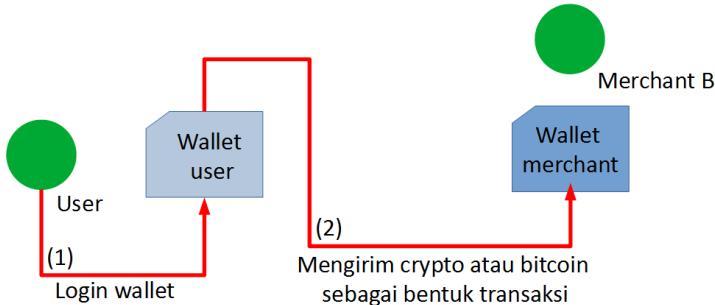
Proses penambahan block juga disebut sebagai proses penambangan atau mining. Karena untuk setiap keberhasilan penambahan blok di dalam jaringan blockchain menyebabkan blockchain mengenerate atau menghasilkan bitcoin secara murni untuk diberikan sebagai imbalan kepada node yang berhasil melakukannya. Langkah-langkah penambahan blok atau penambangan ini dinyatakan orang dalam 7 langkah. Yaitu sebagai berikut:

a) Langkah pertama user menggunakan wallet dan melakukan transaksi crypto atau bitcoin

Pada langkah pertama ini, user masuk ke dalam akun wallet nya. Kemudian melakukan transaksi crypto atau bitcoin. Pada transaksi ini, user mengirim crypto atau bitcoin ke alamat wallet yang lain, atau menerima crypto/bitcoin dari wallet lain.

Transaksi ini mungkin didasari adanya pembelian online atau offline akan tetapi pembayarannya lewat uang crypto. Transaksi bisa

jadi juga didasari adanya penggunaan jasa tepi yang memberikan jasa menerima pembayaran dengan uang crypto. Sehingga pengguna jasa membayarnya dengan mengirim uang crypto dari wallet nya menuju wallet pemberi jasa.

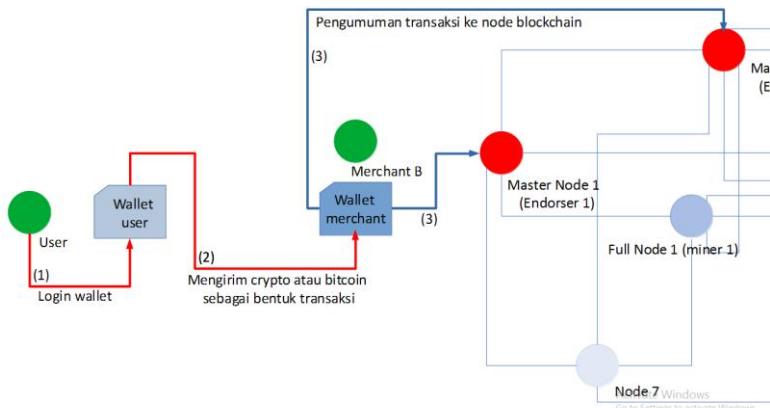


Gambar 28. User menggunakan wallet nya

b) Langkah pengumuman transaksi baru oleh wallet kepada node-node blockchain

Setelah transaksi selesai, wallet mengirimkan pesan ke semua node akan transaksi yang baru saja dia buat agar didaftarkan ke dalam blok yang hendak ditambahkan ke dalam blockchain.

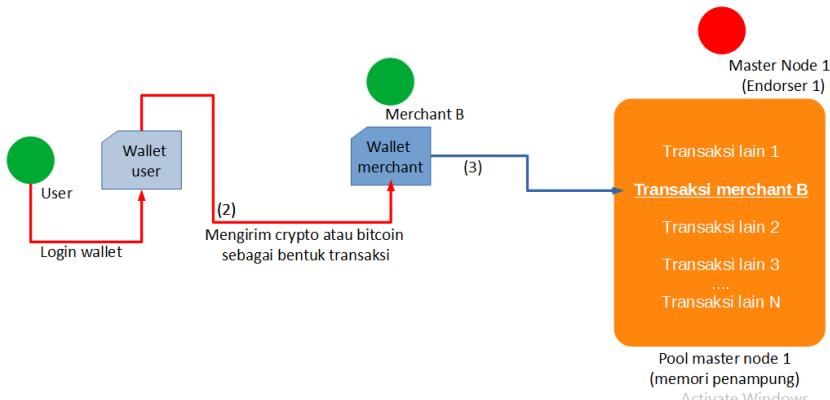
Pengumuman ini kemudian menunggu konfirmasi dari blockchain apakah proses penambahan ke blockchain terkonfirmasi atau tidak. Jika terkonfirmasi maka proses transaksi selesai dan *merchant* bisa memberikan barang yang di beli oleh user, karena crypto atau bitcoin user telah berpindah ke *merchant* secara sah.



Gambar 29. Pengumuman transaksi ke blockchain

c) Langkah seleksi transaksi untuk ditambahkan ke dalam rencana blok

Transaksi yang dikirim wallet kemudian diterima oleh node-node yang ada di dalam blockchain. Transaksi-transaksi itu, jika ada lebih dari 1 wallet yang mengirim, dikumpulkan ke dalam memori khusus dari node (*buffer*) yang disebut *pool*. *Pool* menampung semua transaksi yang dikirim oleh wallet-wallet. Semua transaksi itu antri di *pool*. Penyimpanan ke *pool* ini dapat dilihat ilustrasinya pada gambar 27 dan gambar 30 di bawah.



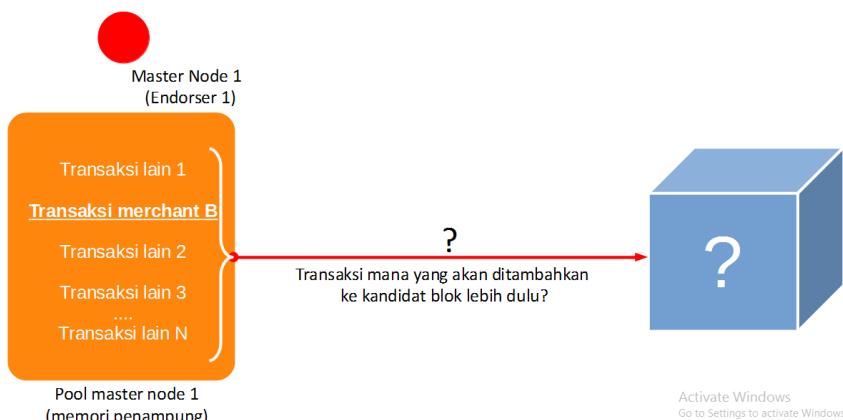
Gambar 30. Transaksi merchant B di simpan di pool node

Kemudian node melakukan seleksi terhadap transaksi-transaksi

mana yang akan ditambahkan ke dalam rencana blok. Pemilihan ini mungkin berdasarkan besarnya *fee* yang bisa didapatkan oleh node jika menambahkan transaksi tersebut lebih dulu ke rencana blok. Ini dengan melihat riwayat transaksi-transaksi sebelumnya dari asal transaksi yang sama apakah dia memiliki simpanan bitcoin yang lebih besar dalam riwayat penyimpanannya dibanding sumber transaksi yang lain.

Yaitu misalkan pemilik wallet A memiliki riwayat penyimpanan transaksi di blockchain yang menyatakan besarnya bitcoin yang dia simpan di dalam blockchain selama ini. Wallet A mungkin memiliki riwayat penyimpanan bitcoin yang lebih besar dari wallet B sehingga semua transaksi dari wallet A yang antri di *pool* didahulukan untuk ditambahkan pada rencana blok, karena memiliki kemungkinan membayar *fee* lebih banyak kepada node tersebut.

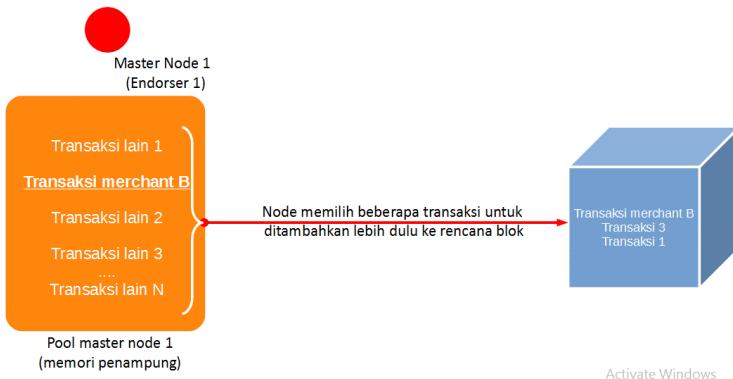
Gambar 31 mengilustrasikan node yang harus memilih diantara beberapa transaksi dari *pool* untuk ditambahkan ke kandidat blok. Kandidat blok karena belum resmi sebagai blok yang diterima atau valid untuk ditambahkan ke blockchain. Semua transaksi yang ada di *pool* berasal dari beberapa orang yang berbeda yang memiliki keperluan yang sama dengan *merchant B*, yaitu menyimpan hasil transaksi ke blockchain.



Gambar 31. Node berusaha memilih transaksi-transaksi yang ditambahkan lebih dulu

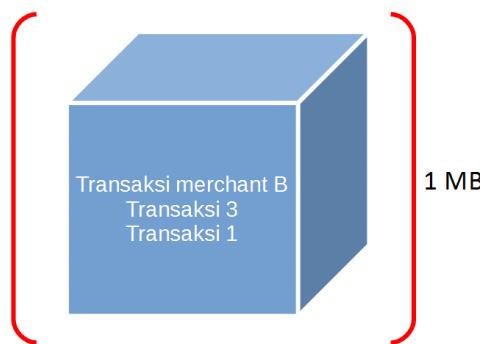
Node bisa saja membuat sistem pendukung keputusan sendiri yang secara otomatis memilih transaksi dari dalam *pool* untuk ditambahkan ke kandidat blok yang rencananya ditambahkan ke blockchain. Gambar 32 mengilustrasikan bagaimana node berhasil memilih beberapa transaksi yang paling penting untuk ditambahkan lebih dulu ke kandidat blok.

Di dalam jaringan blockchain, transaksi dari *merchant B* diumumkan ke semua node, sehingga di node lain, boleh jadi transaksi B kalah bersaing dengan transaksi lain untuk ditambahkan lebih dulu ke kandidat blok node yang lain itu. Akan tetapi ini tidak menjadi masalah karena itu berarti dia berada di blok lain saja yang bukan blok yang dibuat oleh node sebelumnya. Hanya saja mungkin waktu menunggu bagi *merchant B* mungkin menjadi sedikit lebih lama untuk menerima konfirmasi bahwa transaksinya berhasil ditambahkan ke blockchain.



Gambar 32. Node memilih beberapa transaksi untuk ditambahkan ke blok kandidat

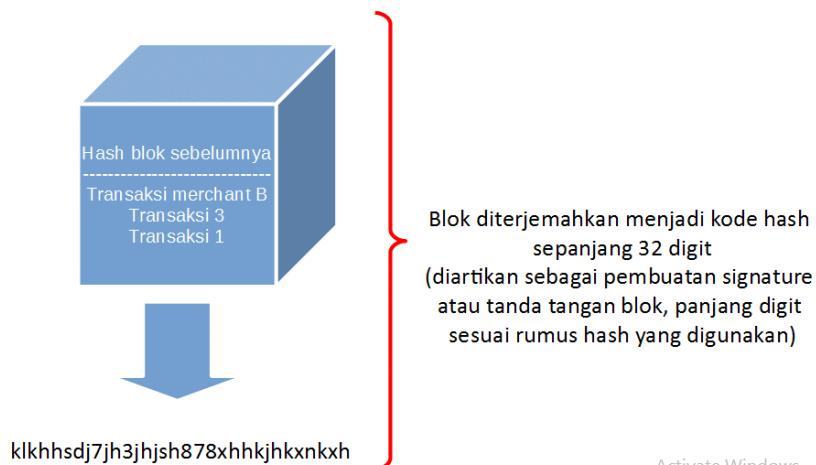
Dalam aturan atau protokolnya, mungkin jaringan blockchain memiliki aturan batas maksimal ukuran blok. Misal maksimal 1 Mb. Ini menyebabkan jumlah transaksi yang ditambahkan ke dalam ukuran 1 blok menjadi terbatas pada 1 Mb itu. Misal hanya ada 5 transaksi yang bisa ditambahkan ke dalam 1 rencana blok.



Gambar 33. Contoh aturan dimana ukuran blok tidak boleh lebih dari 1 MB

d) **Langkah penemuan *signature* blok (hash)**

Pada langkah ini, proses *proof of work* terjadi. Setiap node yang berusaha menambahkan rencana bloknya ke dalam blockchain harus terlebih dulu membuat hash dari bloknya. Sehingga diperoleh 32 digit hash acak. Ini adalah *signature* atau tanda tangan dari rencana blok tersebut.

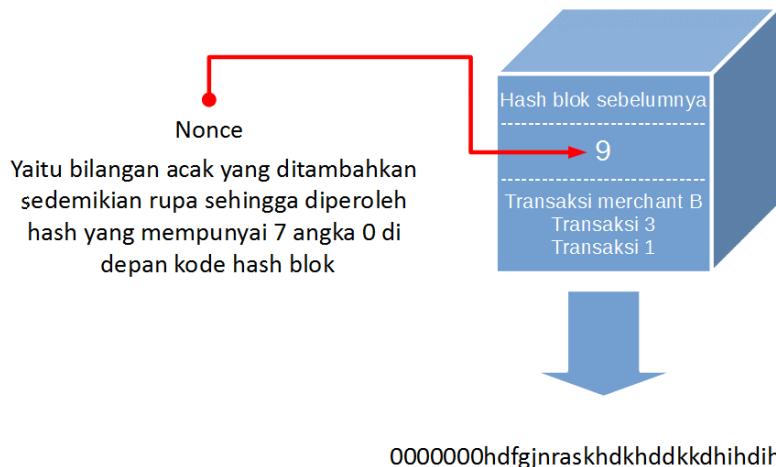


Gambar 34. Node menerjemahkan kandidat blok ke dalam kode hash

Akan tetapi jaringan blockchain memberikan aturan atau konsensus bahwa setiap hash yang dihasilkan oleh node harus memiliki jumlah nol sebanyak 7 digit nol di depan kode acak hashnya. Dalam hal

ini nilai 7 hanya sebuah contoh aturan. Pada dasarnya jaringan blockchain terus menerus merubah setiap 2 minggu sekali aturan jumlah digit nol yang harus ada pada bagian depan kode hash. Bisa lebih besar dari 7, bisa lebih kecil, menurut jumlah node yang bergabung setiap saat di dalam blockchain. Gambar 35 memberikan ilustrasi bagaimana node berusaha keras dengan segala sumberdaya komputasi yang dimilikinya untuk menemukan kode hash yang memiliki 7 digit angka 0 di depan. Ini dilakukan dengan terus mencoba-coba angka acak (*nonce*) sehingga hasil hash blok sesuai yang diinginkan. Yaitu kode hash yang memiliki 7 angka 0 di depan.

Sebagai contoh, node X memperoleh kode hash acak "klkhhsdj7jh3jhjsh878xhhkjhxnxh" maka dia wajib sedemikian rupa merubah kode acak ini menjadi kode acak yang depannya ada 7 digit nol. Misalkan saja dia berhasil memperoleh perubahan hash menjadi "0000000hdfgjnraslkhdhdkddkkdhihdih" yaitu diperoleh kode hash dengan 7 digit 0 di depannya, maka node X dianggap berhasil memecahkan masalah *proof of work* sehingga dia berhak untuk menambahkan rencana bloknya ke blockchain.

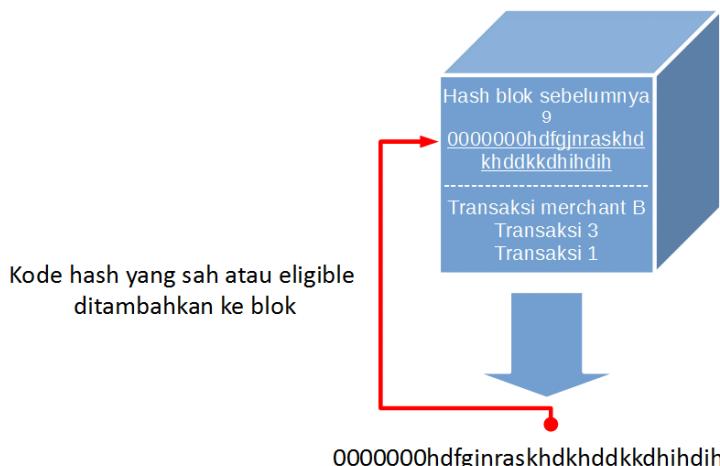


Gambar 35. Penambahan nonce sebagai usaha *proof of work* agar diperoleh kode hash yang memiliki 7 digit didepan kode hash

Merubah kode hash “klkhhsdj7jh3jhjsh878xhhkjhkxnkxh” menjadi kode hash yang memiliki 7 digit angka nol di depannya yaitu “0000000hdfgjnrankhdhddkkdhihdih” dilakukan dengan merubah-rubah **nonce**. *Nonce* adalah sebuah bilangan bulat yang ditambahkan atau disuntikkan ke dalam blok sedemikian rupa sehingga diperoleh hasil hash yang dikehendaki. Komputasi komputer node X adalah dengan mencoba-coba nilai *nonce* yang mungkin sehingga kode hash rencana blok menjadi kode hash dengan 7 digit angka 0 di depannya.

Kode hash yang berhasil memiliki 7 digit angka 0 di depannya disebut sebagai *eligible signature* atau *eligible hash*. Dalam hal ini, kode hash sebelumnya yaitu “klkhhsdj7jh3jhjsh878xhhkjhkxnkxh” dianggap tidak *eligible* sehingga tidak bisa digunakan. Akan tetapi kode hash “0000000hdfgjnrankhdhddkkdhihdih” dianggap *eligible*, atau sah sebagai tanda tangan blok (*signature*). Jika kode hash *eligible* ini lebih dulu ditemukan oleh node, maka ini menyebabkan node tersebut memenangkan hak dari jaringan blockchain untuk menambahkan kandidat bloknya ke dalam blockchain, mendahului node-node yang lain di dalam jaringan blockchain. Sehingga node-node yang lain tidak lagi bisa menambahkan blok baru kecuali menambahkan blok baru di atas blok yang dimenangkan ini, atau melanjutkan blok baru yang lain di atas blok baru ini.

Gambar 36 memberikan ilustrasi pembentukan blok yang sah sesuai dengan *proof of work* yang dikerjakan oleh node sebagai kerja komputasi yang dipersyaratkan sebelum berhak menambahkan blok ke blockchain.



Gambar 36. Kandidat blok ditambahkan kode hash hasil proof of work, yaitu kode hash yang sah atau eligible

Aturan jumlah digit angka 0 adalah tingkat kesulitan daripada kerja poof of work di dalam blockchain. Jaringan blockchain secara berkala akan merubah aturan jumlah angka 0 ini. Konsensus untuk jumlah angka 0 ini adalah “jumlah angka 0 ditentukan sedemikian rupa sehingga total kekuatan komputasi seluruh node di dalam blockchain rata-rata menghasilkan sebuah penambahan blok yang berhasil untuk setiap 10 menit”.

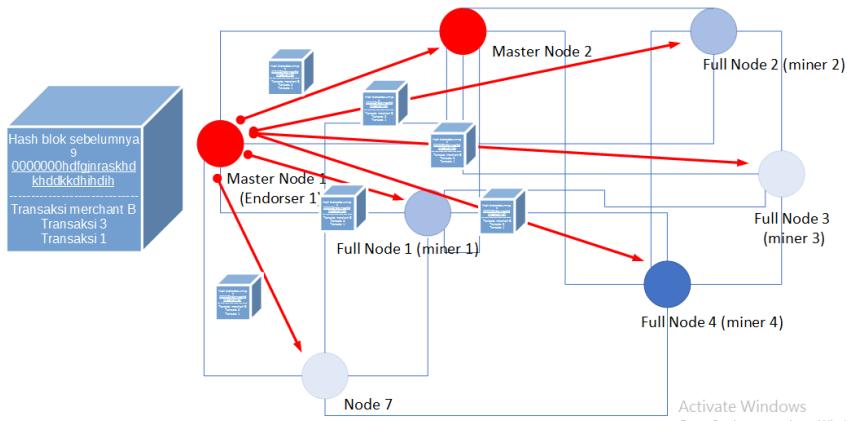
Peninjauan jumlah angka 0 ini adalah setiap 2 minggu sekali (atau aturan lain tentang jumlah hari peninjaun menurut arsitektur blockchainnya). Manakala dengan 7 angka nol, kemudian jumlah orang-orang yang bergabung menjadi node di dalam blockchain bertambah sehingga total kekuatan komputasi di jaringan blockchain bertambah, ini secara total rata-rata akan meningkatkan kecepatan penemuan kode hash yang *eligible*. Sehingga pada gilirannya menambah kecepatan penambahan blok di dalam blockchain.

Penambahan kecepatan ini mungkin berbentuk bahwa penambahan 1 blok bukan lagi total rata-rata secara global adalah 1 blok per 10 menit, tetapi menjadi 1 blok per 8 menit. Ini menyebabkan jaringan blockchain meninjau ulang aturan, lalu menambah aturan jumlah 0 menjadi 9, bukan 7 lagi. Sehingga menambah kesulitan tingkat

komputasi untuk menemukan hash yang *eligible*. Penambahan tingkat kesulitan ini sedemikian rupa sehingga total rata-rata global penambahan blok kembali menjadi 1 blok per 10 menit.

e) Langkah pengumuman (*broadcast*)

Pada langkah ini, node yang berhasil menemukan *eligible hash* wajib mengumumkan blok yang berhasil ditambahkannya ke seluruh node yang lain. Dalam pengumuman ini, dia juga mengumumkan semua transaksi yang ditambahkan ke dalam blok tersebut. sehingga setiap node mengetahui transaksi-transaksi mana saja yang sudah ditambahkan ke dalam blockchain.



Gambar 37. Node 1 yang berhasil mengerjakan *proof of work* lebih dulu, lalu mengumumkan penambahan blok yang telah berhasil dibuatnya kesemua node agar node yang lain ikut menambahkan dan tidak lagi memasukkan transaksi yang sama ke kandidat blok mereka

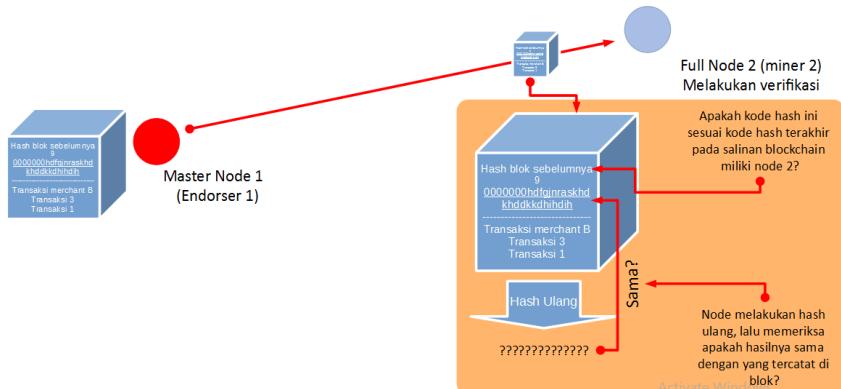
Gambar 37 memberi ilustrasi dimana node 1 telah berhasil lebih dulu mengerjakan *proof of work*, yaitu menemukan nonce sedemikian rupa sehingga hasil hash memiliki 7 angka 0 di depan kode hash blok, yaitu “0000000hdfgjnaskhdkhddkdhidh” di dalam contoh ini.

Node 1 lalu mengumumkan (*broadcast*) ke seluruh node yang lain di dalam jaringan blockchain agar menambahkan blok baru yang dibuatnya itu ke dalam salinan blockchain masing-masing node. Ini juga

mengumumkan agar semua transaksi yang ada di dalam blok baru itu (dalam contoh ini adalah transaksi *merchant B*, transaksi 3 dan transaksi 1) yaitu tidak lagi ditambahkan ke rencana blok baru node-node yang lain. Tindakan ini mencegah pencatatan 2 kali transaksi yang sama sehingga bisa menimbulkan pelanggaran yang disebut sebagai *double spending*.

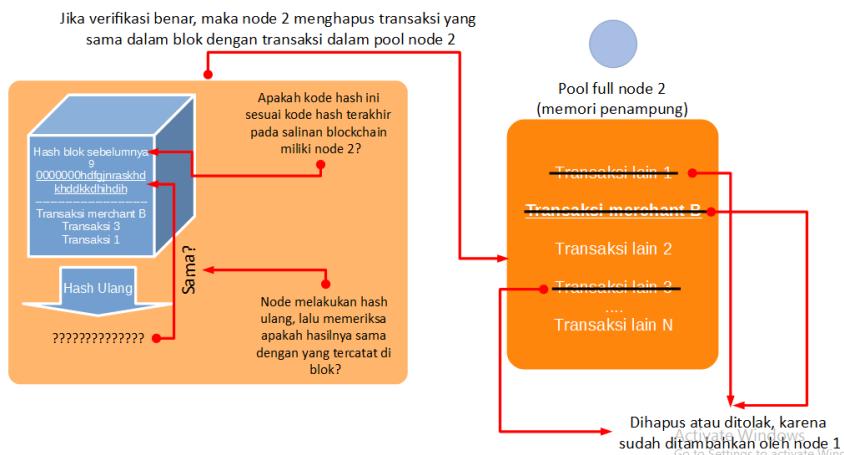
f) Langkah verifikasi

Pada langkah ini, setiap node lalu memeriksa kode hash yang *eligible* tersebut yang datang bersama dengan pengumuman tersebut dengan cara meng-hash-kan ulang isi blok yang diumumkan itu. Jika kode hash benar maka blok terverifikasi benar. Mereka lalu menambahkan juga ke dalam salinan blockchain mereka sendiri.



Gambar 38. Salah satu node yaitu node 2 (dan juga semua node yang lain) melakukan verifikasi terhadap blok yang diumumkan oleh node 1

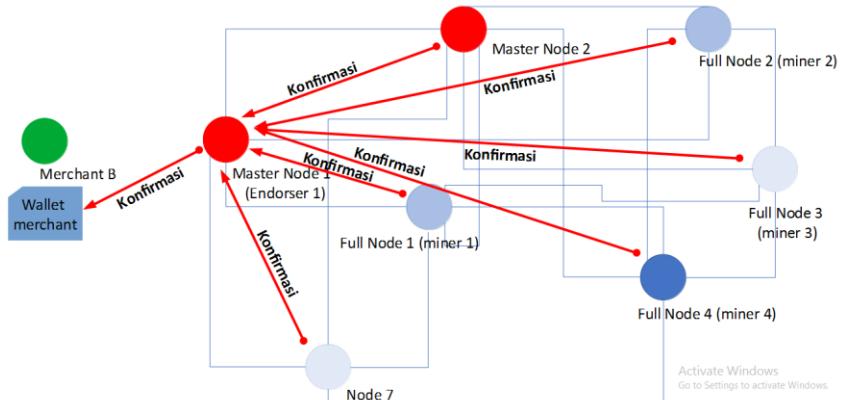
Semua node yang menerima pengumuman tersebut lalu mengetahui semua transaksi-transaksi yang termasuk ke dalam blok baru tersebut. Sehingga mereka dengan segera menghapus transaksi-transaksi yang sama di dalam *pool* nya atau melakukan penolakan penambahan transaksi-transaksi yang sama tersebut untuk rencana blok mereka sendiri.



Gambar 39. Node 2 (termasuk semua node yang lain) menghapus transaksi-transaksi pada memori penampung sementara (pool) jika sudah ditambahkan pada blok baru yang diumumkan node 1

g) Langkah konfirmasi (*confirmation*)

Pada langkah ini, blok yang baru ditambahkan kemudian mendapat konfirmasi. Yaitu sebagai tambahan informasi di dalam berita verifikasi blok di dalam jaringan blockchain.

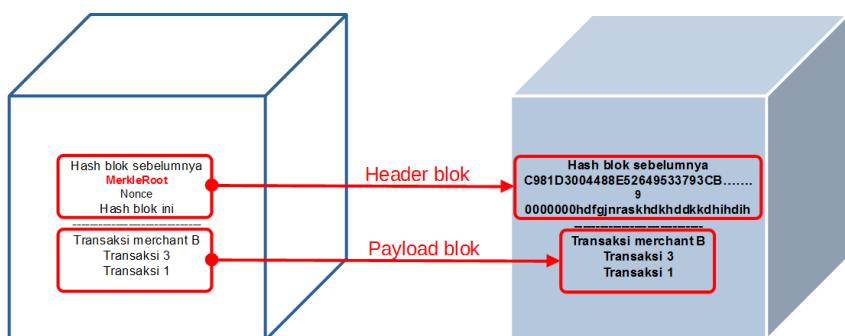


Gambar 40. Proses konfirmasi penambahan transaksi ke dalam blockchain (sampai pada tahap ini proses transaksi merchant B selesai)

Sumber bacaan tambahan : <https://medium.com/coinmonks/how-a-miner-adds-transactions-to-the-blockchain-in-seven-steps-856053271476>

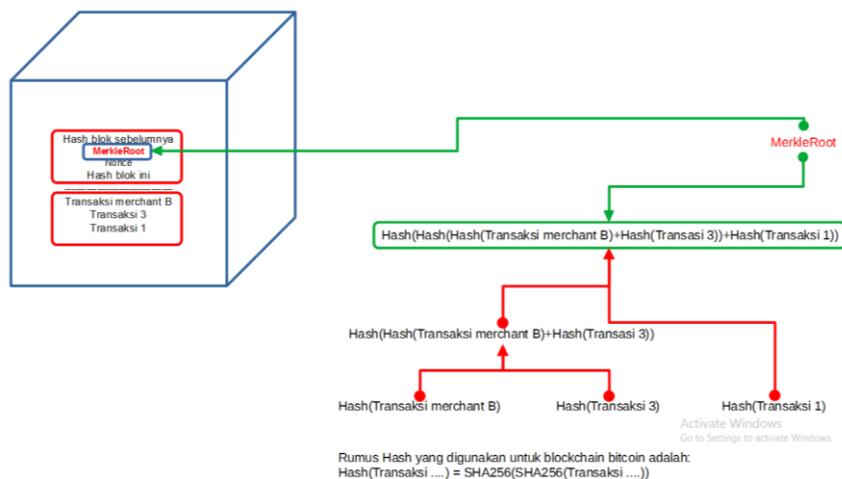
11. Merkle Trees, sebuah cara hashing untuk menjelaskan urutan transaksi

Pada pembuatan blok di dalam blockchain bitcoin. *MerkleRoot* adalah salah satu komponen atau *field* pada *header* blok. *MerkleRoot* dibuat untuk sebagai *signature* (tanda tangan digital) bagi setiap transaksi yang ditambahkan ke dalam blok. Tanda tangan ini memelihara urutan penulisan dari transaksi dan keutuhan individual sebuah transaksi di dalam blok sehingga jika terdapat serangan *hacking* terhadap blok yang mungkin merubah urutan penulisan transaksi dan juga merubah salah satu transaksi secara individual maka ini akan menyebabkan perubahan *signature* berupa berubahnya nilai *merkle tree*.



Gambar 40.1. mengilustrasikan posisi *merkle tree* yang terletak pada *header* blok.

Gambar 40.1. Posisi merkle tree pada blok *MerkleRoot* adalah cara *hash* yang dilakukan secara pohon biner. Dalam cara ini, setiap transaksi diurai menjadi *hash*. Untuk blockchain bitcoin, rumus untuk mengurai transaksi menjadi hash kode adalah SHA256 yang diterapkan 2 kali terhadap data. Gambar 40.2. mengilustrasikan bagaimana data transaksi di dalam blok diurai dalam *hash* SHA256 dalam cara pohon biner *merkle tree*.

**Gambar 40.2. Cara konstruksi merkle tree hash untuk data transaksi**

Berikut ini adalah konstruksi *merkleRoot* menggunakan *javascript*:

```
const tA = 'Transaksi merchant B'
```

```
const tB = 'Transaksi1'
```

```
const tC = 'Transaksi3'
```

```
const sha256 = require('js-sha256').sha256
```

```
// Terapkan dua kali hashing SHA256
```

```
const hA = sha256(sha256(tA))
```

```
const hB = sha256(sha256(tB))
```

```
const hC = sha256(sha256(tC))
```

```
//Hasil
```

```
262F7682F2328820BD1FCBF6DE5FA23635C2F9F36863AA8DED8DC28
E5E420943E4BC574223252158977223AEE4EBDDC0DEEC3EEB7B6D80
141702DA4BA4C7EF4C //ha
BCCA2E7AE39304D7B320B01661E6D9B4E6A27B4286AD140DF66D45
D5A87CDD984A74C6A8F8259C95D4B6362F7AE5B4582072AE372102A
BD8BE7CBC18318290D8 //hB
976F54DF83EFBEECB803521A43798233A76ECE7C18D776C1F9156EAF
70741C14D3C4FAE68434D084C9700D7F02BD097B2B1DFC46F41FD16
39BED61C52186F9DA //hC
```

//Terapkan pada penjumlahan string dari hA dan hB

```
const hAB = sha256(sha256(hA + hB))
```

//Hasil

```
8B1FC9A6372F5A71BB2A08631A6B62308E4839B0D40A452BAE5A8A6  
930541F11A5C1EB0CAC09ADBD934BB3C6C8658617612A82B11DF913  
82BEED7A66CB947F9B //hAB
```

//Terapkan pada penjumlahan string dari hAB dan hC

```
const hABC = sha256(sha256(hAB + hC))
```

//Hasil

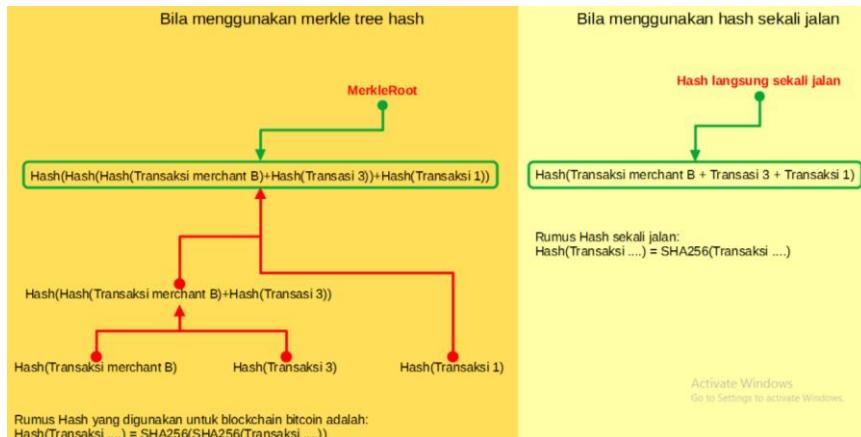
```
C981D3004488E52649533793CB856D8D36116013E50BCF8D5395295  
5FA6DC28335D8FCE77C856A99CD1AD8A102742CF53933D67AA9A8E  
19ABA43F40CAC30DD53 //hABC
```

//Diperoleh *MerkleRoot* yaitu hABC yang kemudian disimpan pada *header* blok

Penggunaan *merkle tree* untuk membangun *hash* dari transaksi tentunya lebih kuat daripada melakukan *hash* secara langsung kepada jumlah string transaksi keseluruhan. Ini karena untuk *hash* transaksi secara keseluruhan tanpa membentuk pohon *hash* menyebabkan hanya terhadap satu rumus *hash* dilakukan usaha pemaksaan (*brute force*) oleh mesin untuk melakukan *tampering* (merubah) data di dalam transaksi. Akan tetapi jika menggunakan pohon *hash* maka ini memerlukan banyak usaha untuk memecahkan lapis-lapis rumus *hash* yang bertingkat.

Gambar 40.3. memberikan ilustrasi bagaimana perbandingan kerumitan *hash* antara keduanya. Pada *hash* sekali jalan, perubahan atau *tampering* pada sebuah data transaksi, misal *hacker* mencoba merubah data transaksi *mechant B*. Maka usaha *hacker* untuk memanipulasi *signature* transaksi (*hash*) adalah hanya satu usaha terhadap rumus *hash*. Akan tetapi jika kita menggunakan *merkle tree*

maka itu memerlukan paling tidak sekitar 10 macam usaha terhadap 10 rumus hash yang bertingkat di dalam *tree*, yaitu usaha terhadap seluruh penggunaan rumus *hash* SHA256 yang disusun bertingkat membentuk sebuah pohon (*tree*).



Gambar 40.3 Perbandingan jika menggunakan hash sekali saja

12. Difficulty dalam blok blockchain bitcoin

Difficulty di dalam blok dari *blockchain bitcoin* adalah sebuah potongan data atau sebuah *field* atau sebuah kolom di dalam *header* blok. Difficulty adalah data tambahan berikut. Difficulty memberikan informasi tentang tingkat kesulitan usaha penambangan untuk menambahkan blok di dalam *blockchain*.

Seperti dijelaskan sebelumnya, proses *node* melakukan penambangan di dalam *blockchain* adalah berbentuk usaha untuk menemukan *hash* blok yang memiliki sejumlah angka 0 (nol) di depannya. Jadi masalah matematika yang harus dipecahkan oleh setiap penambang (*miner*) adalah menemukan *hash* dari blok yang mana *hash* itu memiliki sejumlah angka 0 di depannya. Permasalahan matematika ini mungkin dapat dinyatakan sebagai berikut:

“Temukan 32 digit kode *hash* untuk blok yang memiliki *n* digit angka 0 di depannya”

Sebagai contoh, *blockchain* memberikan permasalahan berupa menemukan kode *hash* blok yang hendak ditambahkan ke dalam

blockchain yang mana kode *hash* itu memiliki 7 digit angka 0 di depannya.

Contoh skenario usaha komputasi untuk memecahkan masalah ini adalah sebagai berikut:

Usaha ke-1:

Hash(blok) = 1gdsjddjksdjsksjklsdjuoiw728hk81

Usaha ke-2:

Hash(blok + nonce-1) = nejskwieodl2n3m471892ksbenw48a

Usaha ke-3:

Hash(blok + nonce-2) =

0lskdjfghgmzk2u333333333633456hw8q

.....dan seterusnya sampai...

Usaha ke-300.000.000:

Hash(blok + nonce ke-299.999.999) =

0000000dhfar637854hdgtw72883h889

Misal pada usaha yang ke-300.000.000 dengan terus menerus merubah nilai *nonce* (bilangan yang dimasukkan/disisipkan ke blok sedemikian rupa agar kode *hash* blok berubah seperti yang diinginkan) diperoleh apa yang disyaratkan oleh permasalahan matematika, yaitu sebuah kode *hash* yang memiliki 7 digit angka 0 di depannya. Karena *node* berhasil menemukan kode *hash* tersebut maka dia berhak menambahkan blok ke dalam *blockchain*. Kode *hash* **0000000dhfar637854hdgtw72883h889** disebut sebagai kode *hash* yang *eligible* atau valid (Proses ini keseluruhan disebut sebagai *proof of work* (PoW)).

Deret angka 0 di depan kode *hash* (yaitu 0000000) disebut sebagai tingkat kesulitan atau *Difficulty*. Akan tetapi ini juga bisa dinyatakan sebagai jumlah digit angka 0 saja, yaitu 7 atau dengan menyebut jumlah keseluruhan *node* di *blockchain* yang berposisi sebagai *miner*, misal ada 1 juta *miner* di *blockchain* maka tingkat kesulitan adalah 1.000.000.

Penentuan *Difficulty* menggunakan jumlah *miner* di dalam keseluruhan *blockchain* karena penentuan berapa digit angka 0 di depan kode *hash* yang wajib ditemukan oleh penambang adalah

berkaitan dengan jumlah penambang (*miner*).

Logikanya sebagai berikut:

Semakin banyak orang yang mendaftar sebagai *miner* ke dalam *blockchain* maka semakin banyak jumlah mesin komputasi (komputer) di dalam jaringan global *blockchain* yang bisa memecahkan permasalahan matematika tersebut secara total. Ini berarti tingkat kesulitan semula yang misalnya hanya menargetkan 6 angka 0 di depan kode *hash* adalah menjadi semakin mudah dikerjakan dan semakin cepat karena semakin bertambahnya jumlah mesin komputasi (komputer) di dalam jaringan *blockchain* yang ikut bergabung sebagai *miner*.

Ini memaksa pengatur *blockchain* global untuk merubah aturan penemukan digit angka 0 di depan kode *hash*. Yang tadinya hanya 6 digit angka 0 maka dinaikkan menjadi 7 digit angka 0 di depan. Perubahan kebijakan ini ditinjau secara periodik dengan patokan bahwa harus tetap dijaga agar banyaknya waktu rata-rata untuk menemukan kode *hash*, dihitung berdasarkan patokan global jaringan, adalah sekitar 10 menit waktu yang dibutuhkan untuk menemukan kode *hash* yang sebenarnya.

Jika ditemukan bahwa nilai rata-rata waktu ini berkangur mengjadi 9 menit atau 8 menit, yang artinya bahwa dalam rata-rata waktu itu, selalu ada salah satu *miner* di seluruh *blockchain* global yang berhasil memecahkan permasalahan matematika maka pengambil kebijakan mesti meninjau kembali aturan angka 0 di depan kode *hash*. Mereka berusaha menaikkan jumlah angka 0 agar waktu rata-rata kembali normal ke sekitaran 10 menit lagi.

Karena itu maka terjadi hubungan yang linier yaitu bahwa jika jumlah *miner* bertambah di dalam *blockchain* maka jumlah angka 0 juga bertambah. Sehingga tentunya masuk akal bahwa jumlah *miner* di dalam *blockchain* juga menyatakan tingkat kesulitan penambangan. Jadi berdasarkan contoh di atas, *Difficulty* bisa dinyatakan sebagai berikut:

Difficulty = 0000000

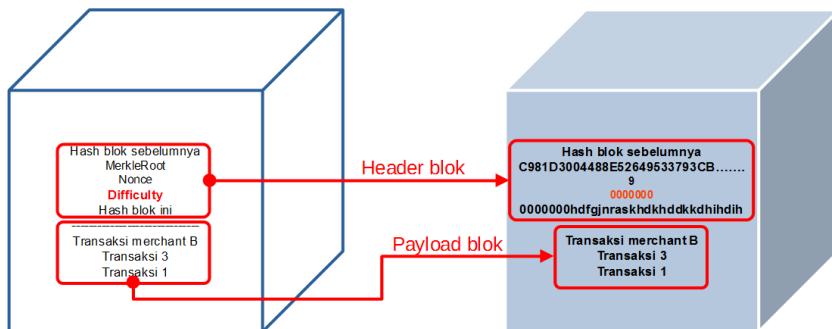
atau:

Difficulty = 7

atau:

Difficulty = 1.000.000

Kesemuanya menyatakan tingkat kesulitan dari penambangan di dalam *blockchain*. Gambar 40.4. mengilustrasikan dimana *Difficulty* diletakkan di dalam blok.



Gambar 40.4. Posisi Difficulty di dalam blok

13. Konstruksi Alamat (Address) Wallet di dalam blockchain bitcoin

Alamat untuk *wallet* di *blockchain* dibuat manakala seseorang mulai mendaftar untuk membuka akun di *blockchain* sebagai pengguna. Setiap pengguna diberi aplikasi antarmuka yang disebut *wallet* lengkap dengan *private key* sebagai identitas *wallet* tersebut.

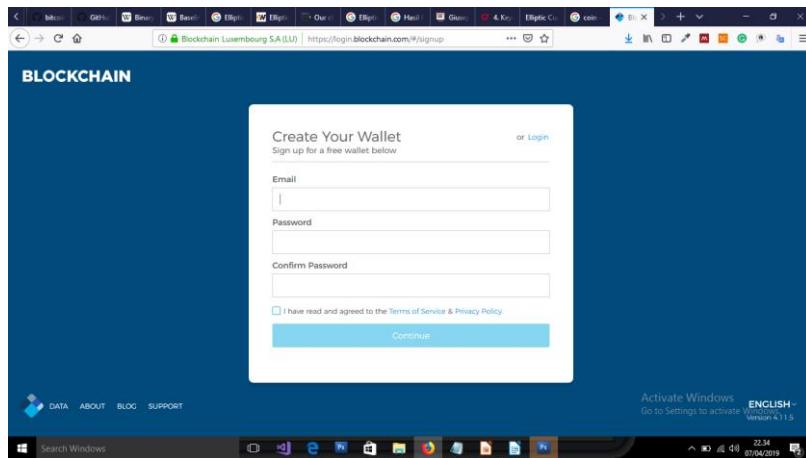
Orang boleh saja memiliki lebih dari 1 *wallet*. Sehingga mereka bisa saja memiliki lebih dari 1 *private key*. *Private key* ini menjadi penting bagi pengguna karena dia digunakan untuk membangkitkan sebuah *publik key*. *Private key* pada dasarnya adalah sebuah bilangan bulat yang ditentukan secara acak. Penentuan bilangan bulat *private key* di dalam *blockchain* adalah dengan cara acak atau random.

Seseorang dapat saja melakukan pemilihan bilangan acak secara manual dengan melemparkan koin sebanyak 256 kali. Sehingga diperoleh sebuah string bit yang panjangnya 256 bit. Bilangan yang dinyatakan oleh 256 bit inilah yang menjadi bilangan random yang menjadi *private key* dimaksud.

Akan tetapi pada sistem *blockchain* sendiri terdapat algoritma atau fungsi yang bisa membangkitkan bilangan random secara

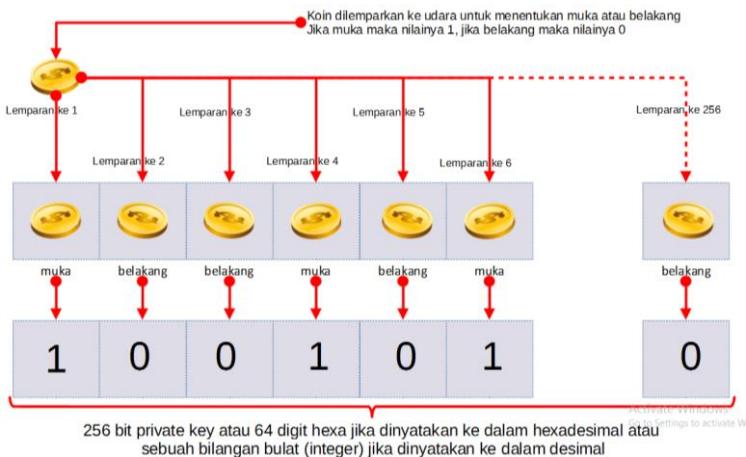
pseudorandom (tidak benar-benar random karena bukan asli dari cara alam/manual menghasilkan bilangan random) tetapi dengan *entropy* (ukuran keacakan) yang cukup tinggi sehingga bisa diterima.

Anda dapat membuat *wallet blockchain* sekarang juga dengan membuka halaman situs *blockchain* pada <https://login.blockchain.com/#/signup> sehingga memperoleh halaman pembuka pertama untuk proses pembuatan *wallet*.



Gambar 40.5. Proses pembuatan *wallet* pada *blockchain bitcoin*

Gambar 40.6 menyajikan tentang bagaimana cara menentukan *private key* dengan sederhana dan manual. Yaitu cukup dengan melemparkan koin sebanyak 256 kali. Akan tetapi cara ini tentunya tidak digunakan manakala mendaftar akun untuk membuat *wallet* di situs *blockchain.com* karena di sana sudah ada algoritma khusus yang digunakan untuk memberikan anda *private key* secara otomatis.



Gambar 40.6. Penentuan private key secara manual

Setelah *private key* selesai dibuat, maka dengan menggunakan algoritma ECDSA (Elliptic Curve Digital Signature Algorithm). Yaitu sebuah algoritma yang digunakan oleh *blockchain* untuk menghasilkan *public key* buat anda. *Public key* ini panjangnya 64 byte + 1 byte extra. Terbagi sebagai 1 byte + 32 byte + 32 byte. 32byte pertama menyatakan bilangan bulat x dan 32 byte kedua menyatakan bilangan bulat y . Sehingga pada dasarnya *public key* menyatakan sebuah pasangan bilangan (x,y) , dengan 1 byte extra sebagai penanda genap atau ganjil. Dari *public key* ini, kemudian *blockchain* menambahkan *network ID* sebagai *prefix*. Sehingga diperoleh kode:

$$\text{Public key} = [1][x][y]$$

Kemudian diterapkan rumus hash SHA256:

$$\text{Hash public key} = \text{SHA256}([1][x][y])$$

Kemudian diterapkan lagi rumus hash RIPEMD160:

$$\text{Hash Hash public key} = \text{RIPEMD160}(\text{SHA256}([1][x][y]))$$

Kemudian dilakukan lagi penambahan *prefix* 1 byte yang menyatakan *networkID*. Sehingga diperoleh:

$$[\text{networkID}][\text{Hash Hash public key}] = [1][\text{RIPEMD160}(\text{SHA256}([1][x][y]))]$$

Kemudian $[1][\text{RIPEMD160}(\text{SHA256}([1][x][y]))]$, diterapkan rumus hash SHA256 secara ganda sehingga diperoleh:



SHA256(SHA256([1][RIPEMD160 (SHA256([1][x][y]))]))

Diperoleh 32 byte kode *hash* yang digunakan untuk sebagai *checksum* (penjaga error). Hanya 4 byte pertama bagian depan dari 32 byte tersebut yang diambil sebagai penjaga error dan ditambahkan ke ekor kode hash semula [1][RIPEMD160 (SHA256([1][x][y]))] yaitu menjadi:

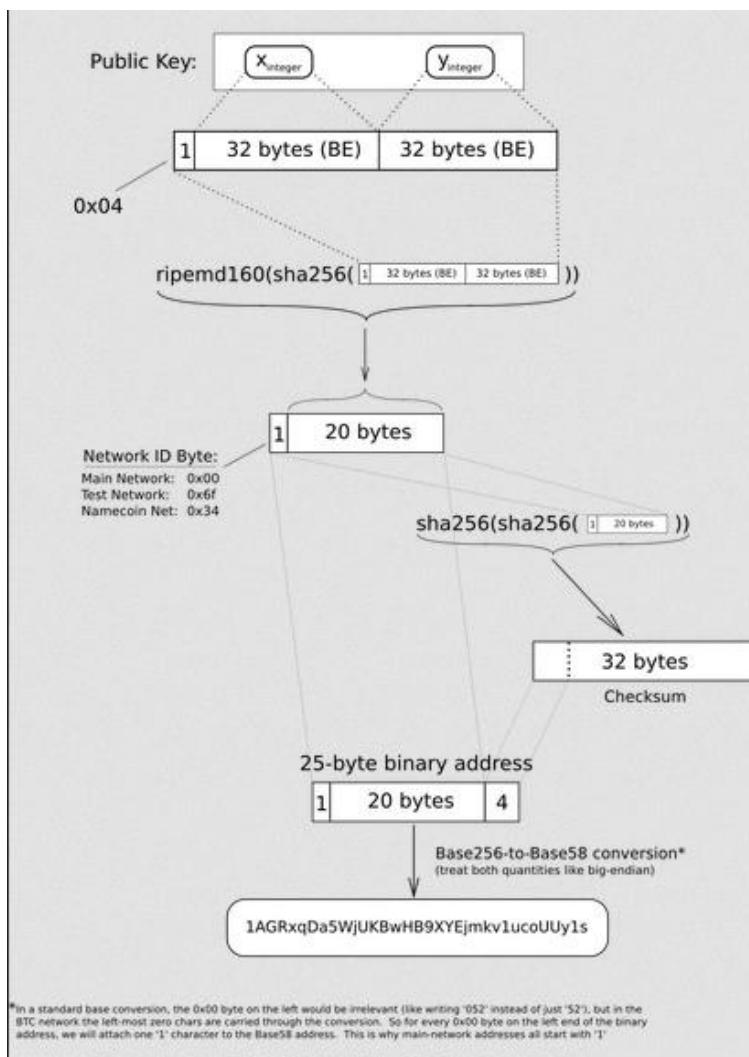
[1][RIPEMD160 (SHA256([1][x][y]))]**[4]**

Kemudian dengan menerapkan proses enkoding (konversi) dari deret bit 0 atau 1 (bentuk biner) ke *alphanumeric* (karakter-karakter huruf dan bilangan) yang mana untuk proses ini *blockchain* menggunakan pemetaan biner ke alphanumerik yang bernama *base58* maka diperoleh alamat bitcoin atau alamat *wallet* yang diinginkan.

Alamat (address) *wallet* = *base58*([1][RIPEMD160 (SHA256([1][x][y]))]**[4]**)

Gambar 40.7. memberikan ilustrasi rinci bagaimana proses pembuatan alamat *wallet* ketika kita pertama kali mendaftar akun pada *blockchain*. Gambar 40.7 diambil berdasarkan lisensi yang memperbolehkan menggunakan gambar tersebut walaupun untuk tujuan komersial ([Creative Commons CC0 1.0 Universal Public Domain Dedication](#))





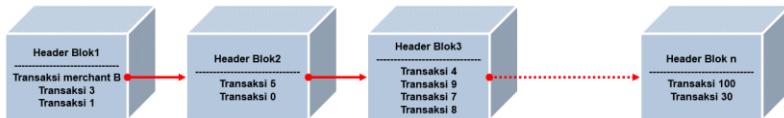
Gambar 40.7. Proses konstruksi alamat wallet bitcoin

(Sumber gambar: <https://en.bitcoin.it/wiki/File:PubKeyToAddr.png>)

14. Anatomi transaksi blockchain bitcoin

Setiap isi dari sebuah blok, terutama adalah transaksi atau himpunan transaksi yang dikemas ke dalam blok sebagai *payload* (isi utama) dari blok. Seluruh blok dapat dilihat saja sebagai kotak-kotak yang menyimpan transaksi-transaksi di dalamnya. Gambar 40.8

mengilustrasikan posisi penting transaksi di dalam blockchain bitcoin. Jika dilihat saja secara sederhana, seluruh rantai blockchain sebenarnya hanyalah rantai transaksi.



Gambar 40.8. Blockchain dengan transaksi-transaksi di dalamnya

Setiap transaksi menyimpan nilai bitcoin dan pemiliknya. Tetapi secara lebih rinci transaksi adalah sebuah struktur data. Dalam bahasa yang sederhana, transaksi adalah sebuah rekord atau baris di dalam tabel excel. Beberapa buah transaksi adalah berarti beberapa baris excel. Hanya demikian.

Akan tetapi, di dalam konstruksinya, struktur data transaksi tidak dinyatakan ke dalam bentuk rekord excel. Struktur data diimplementasikan dalam bentuk JSON atau XML atau berbagai pilihan struktur data lainnya.

Contoh sederhana berikut yang diambil dari (Peyrott, 2017) adalah transaksi yang dinyatakan ke dalam JSON:

```
{  
"previous-transaction-id": "FEDCBA987654321...",  
"owner-pubkey": "123456789ABCDEF...",  
"prev-owner-signature": "AABBCCDDEEFF112233..."  
}
```

Rantai transaksi tidaklah identik dengan rantai blok di dalam blockchain. Di dalam sebuah rantai blok (blockchain) terdapat lebih dari satu rantai transaksi yang menjadi bagian dari rantai blok. Untuk melihat bagaimana perbedaan ini, terlebih dulu hendak diuraikan bagaimana rantai transaksi terbentuk.

Untuk menjelaskan bagaimana rantai transaksi terbentuk di dalam blockchain, kita menggunakan penyederhanaan struktur data transaksi sebagaimana yang digunakan oleh (Peyrott, 2017) yaitu bahwa sebuah struktur data transaksi secara pokok dapat dinyatakan

hanya dalam 4 bagian, yaitu:

- Bagian identitas (ID) transaksi sekarang (*current transaction*)
- Bagian identitas transaksi sebelumnya (*previous-transaction-id*)
- Kunci publik orang yang jadi tujuan pengiriman bitcoin (tujuan transaksi)
- Tanda tangan pengirim bitcoin

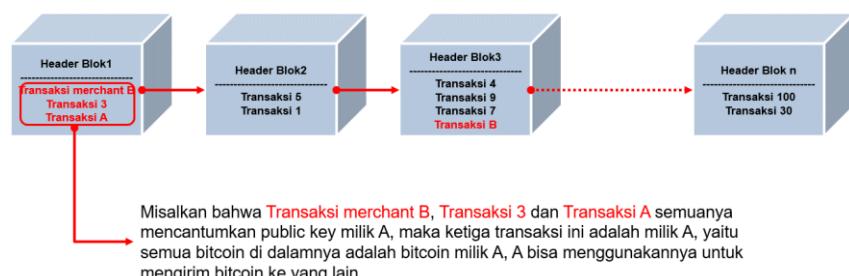
Secara umum, sebuah transaksi di dalam blockchain bitcoin adalah transaksi pengiriman koin bitcoin kepada yang berhak atau penerima. Hanya demikian. Tidak seperti pada anatomi transaksi pada blockchain Ethereum atau blockchain *Hyperledger fabric* atau blockchain lainnya.

Seseorang misal A hendak mengirim 50 BTC kepada B maka apa yang dilakukan oleh A adalah A membuka *walletnya*. Kemudian dari *wallet* itu dia membuat transaksi pengiriman ke 50 BTC kepada B.

Apa yang terjadi pada aplikasi *wallet* ketika A membuat transaksi pengiriman 50 BTC adalah sebagai berikut:

- *Wallet* melihat uang A yang tersedia.
- Uang berbentuk daftar id transaksi yang dia terima dari pengirim bitcoin sebelumnya kepadanya atau dia terima sebagai insentif karena telah berhasil menambang (*coinbase* untuk *miner*) serta mungkin alamatnya pada blok mana dia tersimpan di blockchain. Ini karena setiap transaksi menyimpan bitcoin untuk A di blockchain.

Sebuah transaksi dengan bitcoin di dalamnya adalah milik A jika di dalam transaksi itu tercantum public key nya adalah milik A. Gambar 40.8.1 memberikan ilustrasi tentang kepemilikan ini. Sebenarnya semua transaksi milik A bisa tersebar di beberapa blok, tidak saja di blok 1 seperti contoh gambar 40.81.



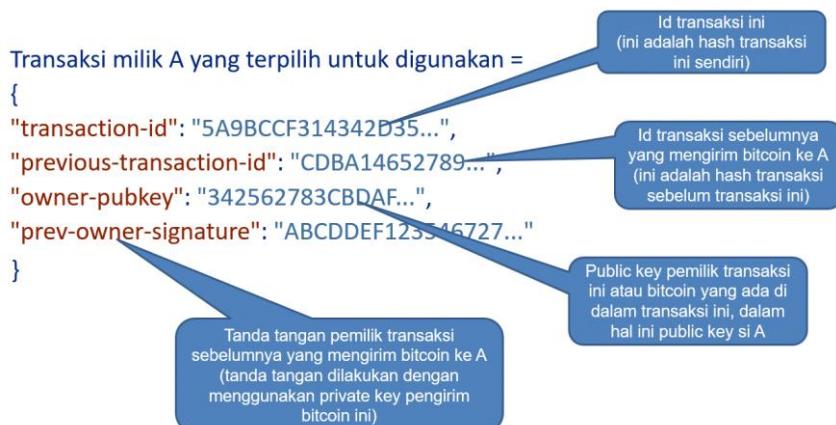
Gambar 40.8.1. Contoh kepemilikan transaksi oleh A di dalam blockchain

Jika terdapat id transaksi yang menyimpan uang 50 BTC atau lebih dari 50 BTC maka *wallet* memilih id transaksi tersebut sebagai id transaksi yang hendak dialihkan nilai bitcoinnya. Misal transaksi yang terpilih itu adalah **transaksi A** pada gambar 40.8.1.

Transaksi milik A yang terpilih untuk digunakan =

```
{
  "transaction-id": "5A9BCCF314342D35...",
  "previous-transaction-id": "CDBA14652789...",
  "owner-pubkey": "342562783CBDAF...",
  "prev-owner-signature": "ABCDDEF123546727..."
}
```

Gambar 40.9 memberikan penjelasan bodi dari transaksi ini sebagai berikut:



Gambar 40.9. Penjelasan bodi transaksi yang terpilih untuk digunakan oleh A

Kemudian *wallet* membuat struktur data transaksi yang baru dengan bentuk struktur sebagai berikut:

Transaksi d =

```
{
}
```

```
"previous-transaction-id": "5A9BCCF314342D35...", (ini adalah  
hash transaksi sebelumnya)  
"owner-pubkey": "123456789ABCDEF...", (ini adalah public key dari  
B sebagai penerima)  
}
```

Setelah struktur data transaksi di atas dibuat, struktur data transaksi itu ditandatangani menggunakan *private key* dari pengirim yaitu *private key* milik A.

Yaitu berbentuk:

```
"prev-owner-signature": "AABBCCDDEEFF112233..."
```

Kemudian tandatangan ini ditambahkan kepada struktur data transaksi menjadi sebagai berikut:

Transaksi f =

```
{  
"previous-transaction-id": "5A9BCCF314342D35...",  
"owner-pubkey": "123456789ABCDEF...",  
"prev-owner-signature": "AABBCCDDEEFF112233..."  
}
```

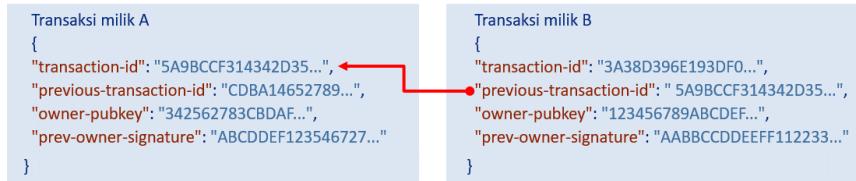
Wallet dapat menambahkan id untuk transaksi baru ini yang bisa merupakan hasil *hash* dari struktur data pada poin f di atas sehingga diperoleh struktur data transaksi sebagai berikut:

Transaksi g =

```
{  
"transaction-id": "3A38D396E193DF0...", (ini adalah hash(Transaksi  
f))  
"previous-transaction-id": "5A9BCCF314342D35...",  
"owner-pubkey": "123456789ABCDEF...",  
"prev-owner-signature": "AABBCCDDEEFF112233..."  
}
```

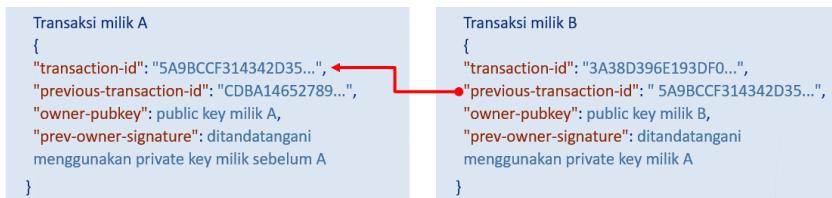
Dengan demikian, diperoleh struktur data transaksi yang lengkap sebagaimana point g. Walaupun di dalam konteks ini dikatakan lengkap tetapi ini bentuk yang disederhanakan. Bentuk ini sederhana karena belum ada tercantum nilai koin 50 BTC di dalamnya dan sebagainya.

Selanjutnya B menerima transaksi ini dan tercatat di blockchain. Transaksi dari A ke B di dalam blockchain adalah membangun rantai transaksi sebagai berikut:



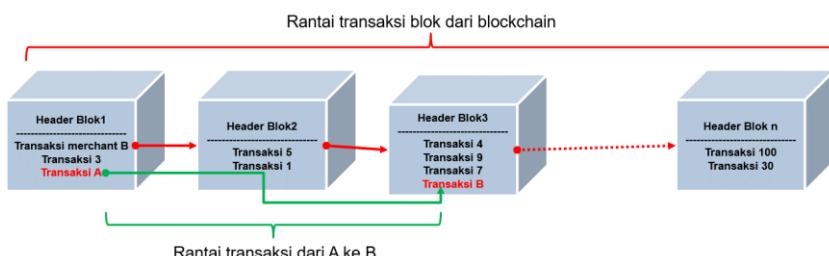
Gambar 40.10. Rantai transaksi dari A ke B

Gambar 40.11 memberikan ilustrasi dengan penjelasan ditiap bagiannya sebagai berikut:



Gambar 40.11. Rantai transaksi dari A ke B

Jika kita melihat kedudukan rantai transaksi dari A ke B di dalam rantai blok, maka terlihat perbedaan yang dapat diilustrasikan oleh gambar 40.12 sebagai berikut:



Gambar 40.12. Perbedaan antara rantai transaksi dan rantai blok

Demikian penjelasan ini untuk contoh anatomi transaksi yang sederhana seperti di atas. Akan tetapi di bawah ini ingin diuraikan dengan lebih rinci tentang anatomi transaksi pada blockchain bitcoin. Anatomi transaksi yang sebenarnya dari blockchain adalah setiap

transaksi terbagi dalam dua bagian. Bagian pertama adalah bagian input dan kedua adalah bagian output. Jika transaksi A dinyatakan dalam bagian input dan output, mungkin terlihat sebagai berikut:

```
{  
/*Bagian Input*/  
"previous-transaction-id": "CDBA14652789...",  
"prev-owner-signature": "ABCDEF123546727..."  
  
/*Bagian Output*/  
"transaction-id": "5A9BCCF314342D35...",  
"owner-pubkey": "342562783CBDAF...",  
}
```

Secara lebih rinci, struktur data transaksi blockchain sebenarnya adalah berbentuk sebagai berikut:

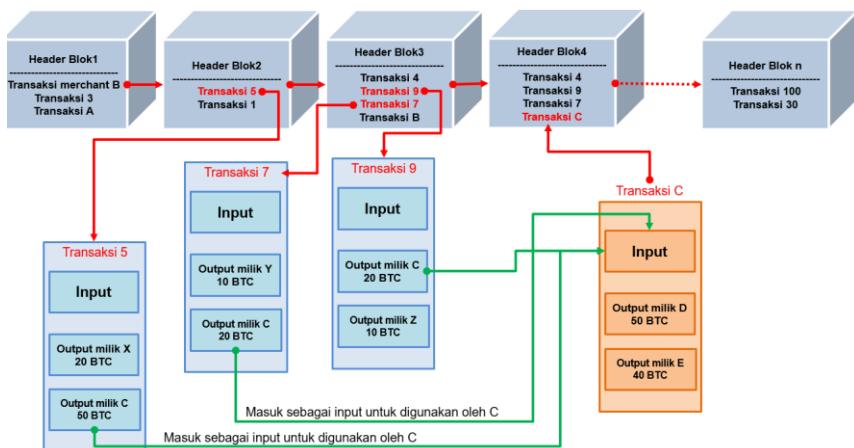
```
{  
"version": .....,  
  
/*Bagian Input*/  
"input-count": .....,  
"input": [ { "TXID": .....,  
           "VOUT": .....,  
           "ScriptSig Size": .....,  
           "ScriptSig": .....,  
           "Sequence": .....  
         },  
  
/*Bagian Output*/  
"Output Count": .....,  
"Output": [ {"Value": .....,  
            "ScriptPubKey Size": .....,  
            "ScriptPubKey": .....,  
          }],  
  
"Locktime": .....
```

}

Struktur data di atas membolehkan lebih dari 1 input dan lebih dari 1 output. Ini memiliki arti bahwa seseorang dapat memiliki sumber bitcoin dari beberapa transaksi di beberapa blok blockchain, dan dapat mengirimkan ke lebih dari 1 orang.

Sebuah contoh sebagai berikut:

Misalkan C hendak mengirim bitcoin ke D dan E, tetapi C memiliki bitcoin yang tersimpan di transaksi 5, transaksi 9 dan transaksi 7 dimana transaksi 5 tercatat di blok 2 dan transaksi 9 dan 7 tercatat bersamaan di blok 3. C ingin menggunakan bitcoin di ketiga transaksi itu untuk dikirim ke D dan E. Setiap transaksi milik C semuanya masing-masing memiliki bagian input-output. Koin bitcoin C terletak pada bagian output. Tidak semua koin adalah milik C pada transaksi tersebut, hanya output yang menyatakan public key C saja yang memang menjadi milik C dan bia dicairkan atau digunakan oleh C untuk membuat transaksi baru yang mengirim bitcoin ke D dan E. Gambar 40.12 mengilustrasikan bagaimana C mengkonstruksikan transaksi baru yaitu transaksi C yang kemudian disimpan di blok 4 blockchain.



Gambar 40.13. Pembuatan transaksi C oleh C untuk mengirim bitcoin ke D dan E

Ini menjelaskan bahwa tidak semua koin pada transaksi yang nantinya dibuat C adalah milik D, hanya pada output dimana *public key*

D dinyatakan. Pada output yang lain, bitcoin adalah milik E karena pada output itu dinyatakan *public key* E.

Semua output pada transaksi yang tidak sedang digunakan untuk dikirim ke orang lain disebut sebagai *Unspent Transaction Output* (UTXO). Pada contoh di gambar 40.13, Ouput milik X, milik Y dan Z adalah UTXO. Kecuali jika X atau Y atau Z menggunakan sebagaimana C menggunakannya.

Berikut ini adalah bagaimana C mengkonstruksikan sebuah transaksi baru yaitu transaksi C. Konstruksi transaksi C dilakukan dengan membuat struktur data sebagai berikut:

```
{  
    "version": "versi struktur data transaksi, bisa berubah versi",  
    /*Bagian Input*/  
    "input-count": 3, (menyatakan jumlah input yang digunakan, ada 3  
    transaksi digunakan)  
    "input": [ { "TXID-5": "id transaksi 5 di blok 2",  
                "VOUT": 2, (ini karena dia menggunakan ouput kedua  
                pada transaksi 5),  
                "ScriptSig Size":.....,  
                "ScriptSig":.....,  
                "Sequence":.....  
            },  
            {"TXID-9": "id transaksi 9 di blok 3",  
                "VOUT": 1, (ini karena dia menggunakan output  
                pertama pada transaksi 9)  
                "ScriptSig Size":.....,  
                "ScriptSig":.....,  
                "Sequence":.....  
            },  
            {"TXID-7": "id transaksi 7 di blok 3",  
                "VOUT": 2, (ini karena dia menggunakan output kedua  
                pada transaksi 7)  
                "ScriptSig Size":.....,  
                "ScriptSig":.....,
```

```
"Sequence":.....  
}  
],  
/*Bagian Output*/  
"Output Count": 2, (jumlah output, yaitu untuk output untuk D dan  
untuk C)  
"Output": [{"Value": 50 BTC, (nilai bitcoin yang dikirim ke D,  
tetapi aslinya dinyatakan dalam satuan  
Satoshi, bukan BTC)  
"ScriptPubKey Size":.....,  
"ScriptPubKey": "skrip public key D"  
},  
{"Value": 40 BTC, (nilai bitcoin yang dikirim ke D,  
tetapi aslinya dinyatakan dalam satuan  
Satoshi, bukan BTC)  
"ScriptPubKey Size":.....,  
"ScriptPubKey": "skrip public key E"  
}],  
"Locktime": 000000 (diisi 0 saja sementara, nanti oleh miner yang  
mengisikan waktu sebenarnya yaitu waktu  
saat transaksi ini valid tersimpan di block  
blockchain)  
}
```

Pembayaran berdasarkan public key ini disebut sistem P2PK (*pay to public key*). Yaitu pembayaran dengan menyebut *public key* penerima pada bagian output transaksi.

Akan tetapi ada banyak cara pembayaran selain menyebut *public key* penerima. Ada juga pembayaran yang tidak menyebut *public key* penerima tetapi hanya menyebut *hash public key* penerima pada bagian output transaksi. Pembayaran jenis ini disebut *pay to public key hash* atau disingkat P2PKH.

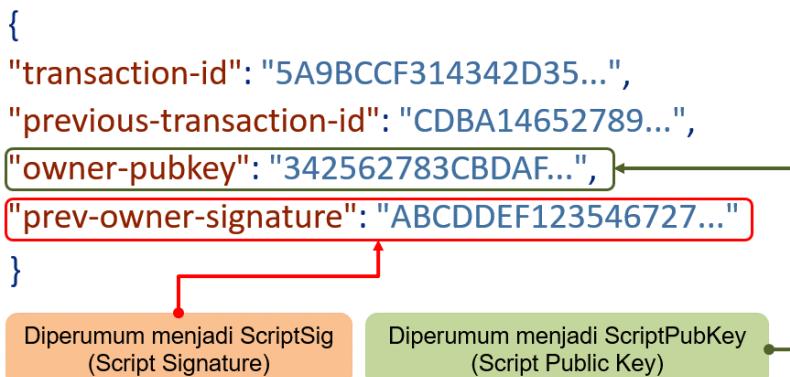
Cara pembayaran lain adalah P2MS (*pay to multisig* atau *pay to multisignature*), P2SH (*pay to script hash*), NULL_DATA. Berikut ini ingin dijelaskan tentang bagaimana orang membuat atau mengkonstruksikan

skrip P2PK, P2PKH, P2MS, P2SH dan NULL_DATA.

Secara umum P2PK, P2PKH, P2MS, P2SH dan NULL_DATA dinyatakan di dalam transaksi sebagai ScriptSig dan ScriptPubKey.

1. Penjelasan tentang ScriptSig dan ScriptPubKey

ScriptSig dan ScriptPubKey adalah perumuman dari *signature (prev-owner-signature)* dan *public key (owner-pubkey)* yang tertera sebagai suatu bagian dalam struktur data transaksi. Sebagaimana struktur data transaksi yang sederhana seperti telah dibahas sebelumnya seperti pada gambar 40.9.



Gambar 40.14. Posisi ScriptSig dan ScriptPubKey sebagai bentuk umum sebelumnya

Pada contoh struktur data transaksi sederhana sebelumnya seperti bentuk transaksi pada gambar 40.9, agar penerima transaksi yaitu B dapat menerima dan menggunakan bitcoin yang dikirim oleh pengirim A maka hal mendasar yang dilakukan A adalah menandatangani transaksi yang menyatakan bahwa benar A yang mengirim bitcoin tersebut dan itu tidak dapat disangkal oleh A, dan *public key* menegaskan bahwa hanya B tidak ada yang lain yang dapat menerima transaksi yang dikirim oleh A.

Penerima B dapat menggunakan bitcoin pada bentuk sederhana transaksi gambar 40.14 jika memenuhi bahwa pasangan *signature* dan *public key* adalah benar atau TRUE. Pasangan tersebut

adalah true manakala B mendekrip *signature* menggunakan *public key* B maka transaksi diperoleh kembali dan di dalamnya terdapat catatan *public key* yang sama dengan *public key* yang digunakan untuk mendekrip *signature*.

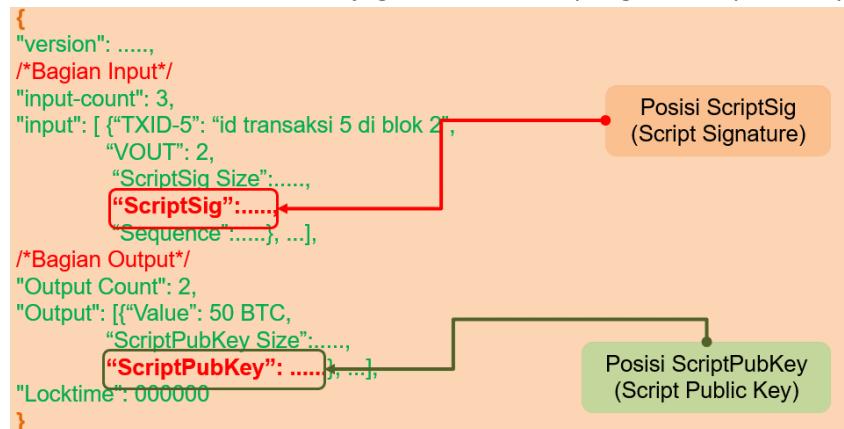
Demikian juga berlaku untuk pasangan ScriptSig dan ScriptPubKey. ScriptPubKey adalah skrip program yang menerima input berupa ScriptSig. Jika program di dalam ScriptPubKey dijalankan dan hasilnya TRUE maka penerima B dapat menggunakan bitcoin yang ada di dalam output transaksi yang ditujukan atau dikirim untuk B.

ScriptPubKey(ScriptSig) = TRUE

Atau ditulis secara sebenarnya sesuai cara baca mesin virtual blockchain dari kiri ke kanan

[ScriptSig] ScriptPubKey = TRUE

Gambar 40.15 menunjuk kepada posisi ScriptSig dan ScriptPubKey pada transaksi yang dibuat oleh pengirim. Ketika pengirim membuat transaksi maka dia juga membuat ScriptSig dan ScriptPubKey.



Gambar 40.15. Posisi ScriptSig dan ScriptPubKey pada struktur data transaksi

Berikut ini hendak dikemukakan penjelasan cara untuk mengkonstruksikan ScriptSig dan ScriptPubKey menurut jenis pembayaran bitcoin yaitu P2PK, P2PKH, P2MS, P2SH dan NULL_DATA.

2. Pembuatan ScriptSig dan ScriptPubKey untuk P2PK

Sebagaimana bahwa kedua kata tersebut mengandung kata *Script*, ini menunjukkan bahwa keduanya adalah sebuah skrip. Skrip adalah sebuah untai kode atau program kecil yang sengaja ditulis oleh pembuat transaksi agar nantinya orang yang menerima transaksi dapat melakukan validasi sekaligus validasi bagi yang bersangkutan untuk bisa menggunakan uang atau bitcoin yang ada di dalam transaksi.

ScriptSig artinya skrip yang memuat *digital signature* di dalamnya. Yaitu *signature* yang dibuat oleh pembuat transaksi atau pengirim.

ScriptPubKey artinya skrip yang memuat *public key* penerima di dalamnya atau hash nya atau skrip *public key* di dalam skrip *public key* dalam hal ini bisa menjadi lebih rumit menjadi skrip di dalam skrip.

Skrip tersebut adalah program yang ditulis di dalam bahasa yang dimengerti oleh mesin virtual *blockchain* sehingga bisa dieksekusi oleh mesin virtual untuk melakukan validasi transaksi.

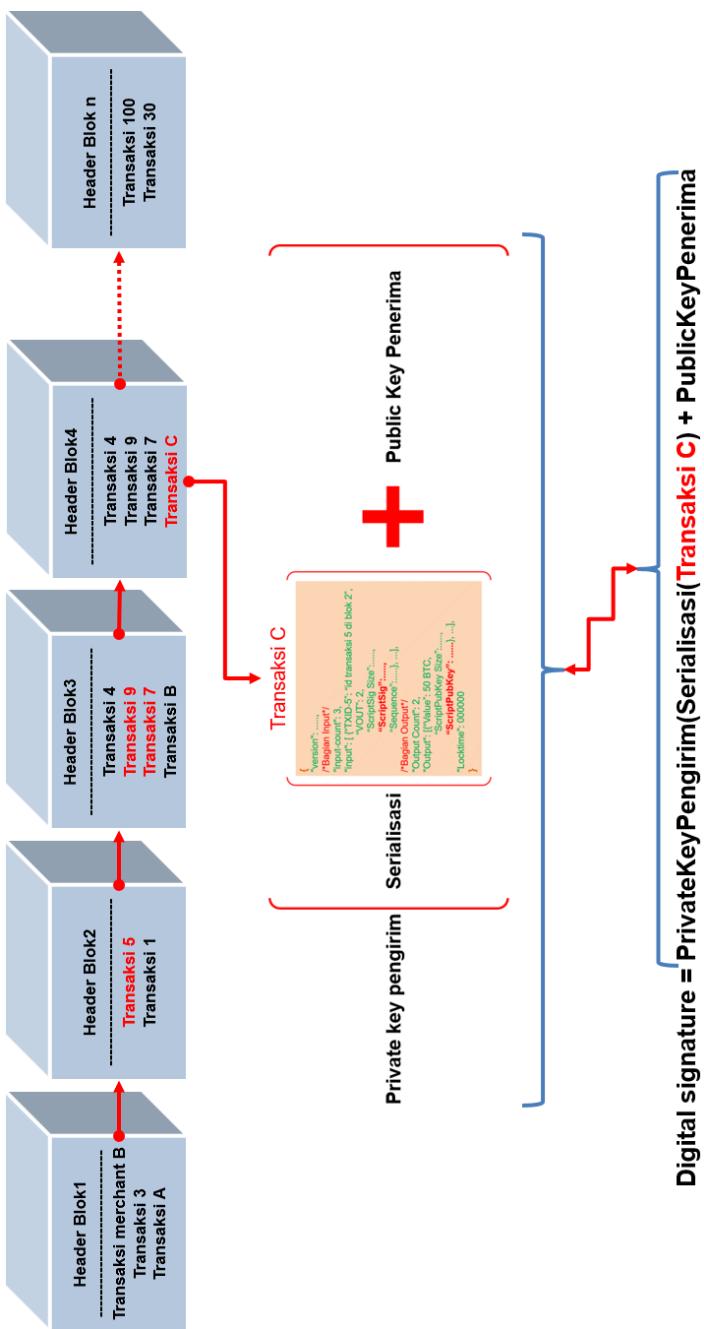
Skrip adalah bahasa khusus yang mengambil bentuk serupa dengan bahasa assembler. Yaitu bahasa di dalam bentuk op code atau *mnemonic* (kata singkatan yang mengandung arti perintah).

Contoh penulisan atau pembuatan skrip ini untuk P2PK:

ScripSig = <*signature*>

ScripPubKey = <*public key*> CHECKSIG

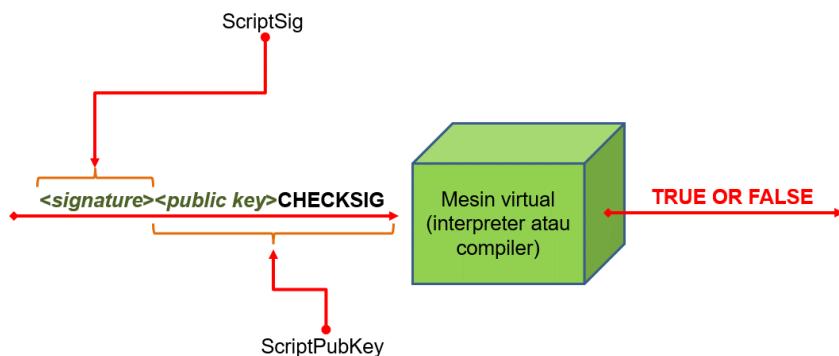
Dimana <*signature*> adalah data langsung dalam hal ini adalah *signature* itu sendiri dan <*public key*> adalah juga data yang merupakan *public key* itu sendiri.



Gambar 40.15.1. Cara membuat <signature> dalam hal ini adalah *digital signature*

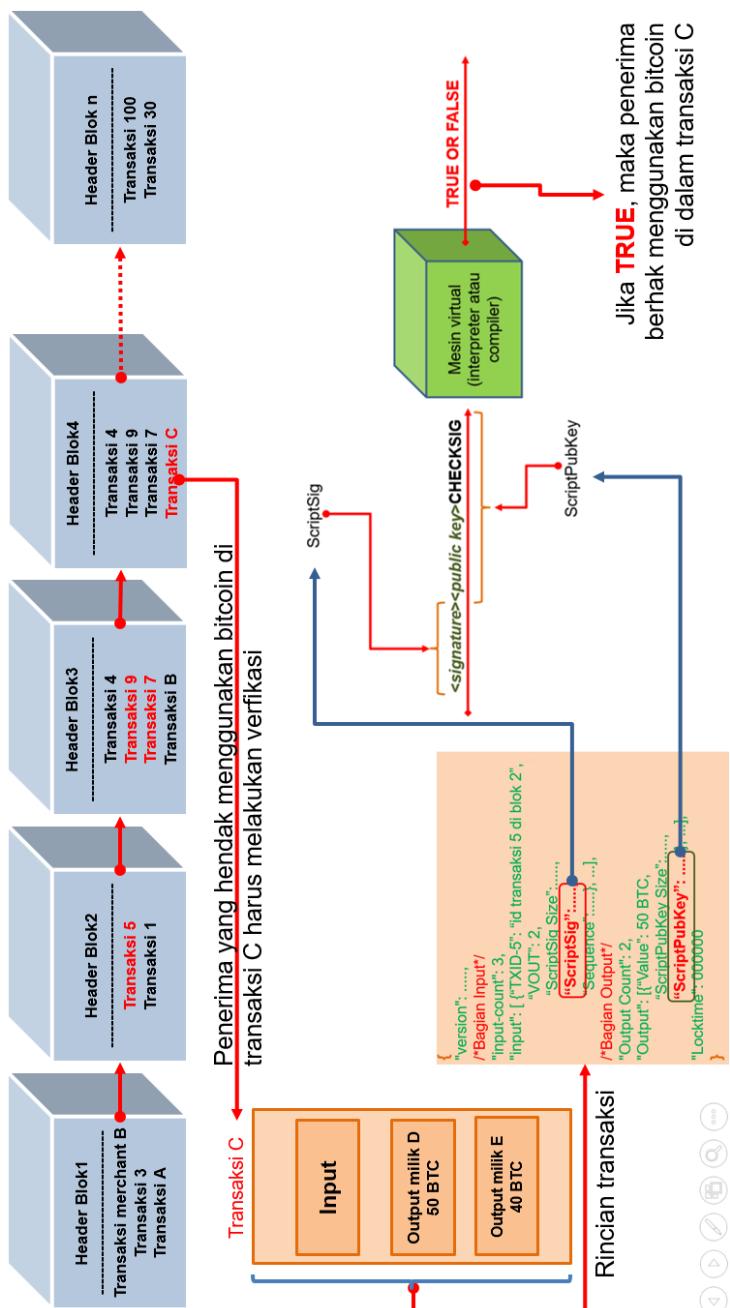
Adapun CHECKSIG adalah *mnemonic* yang berisi perintah untuk memeriksa apakah *signature* adalah benar sesuai *public key*.

Jika kedua skrip ini hendak dieksekusi maka keduanya diinputkan ke dalam mesin virtual menjadi sebagaimana diilustrasikan oleh gambar 40.16.



Gambar 40.16. Mesin virtual blockchain mengeksekusi ScriptSig-ScriptPubKey

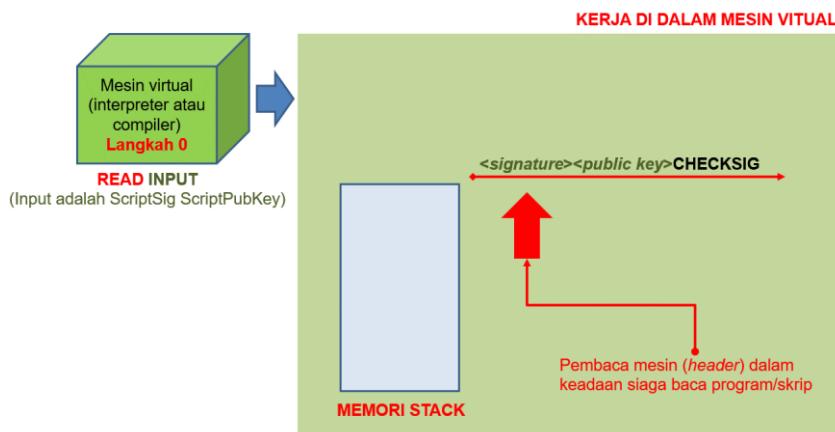
Gambar 40.17 mengilustrasikan bagaimana penerima D atau E hendak menggunakan uang bitcoin yang ada pada output transaksi C. Pada *wallet* D atau E atau pada *miner* terdapat mesin virtual *blockchain* yang melakukan proses pemeriksaan identitas penerima dengan memeriksa keabsahana *ScriptSig* *ScriptPubKey* pada transaksi.



Gambar 40.17. Rincian bagaimana mesin virtual bekerja terhadap transaksi

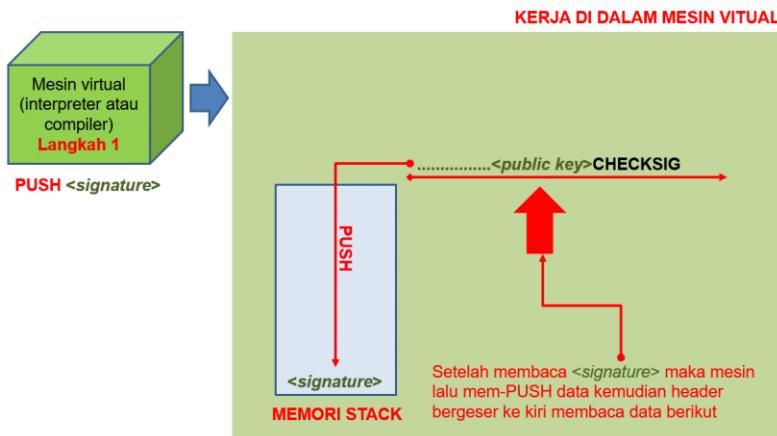
Bahasa ini memanfaatkan memory khusus yang bernama *stack*. Sebuah ruang memori biasa saja akan tetapi memiliki aturan pembacaan dan penulisan yang disebut *last in first out* atau LIFO. Sepertinya disana tidak ada penggunaan variabel seperti umumnya bahasa pemrograman. Akan tetapi murni hanya secara aktif menggunakan stack untuk setiap skrip yang hendak dieksekusi. Berikut ini langkah-langkah eksekusi skrip menggunakan stack oleh mesin virtual sebagai berikut:

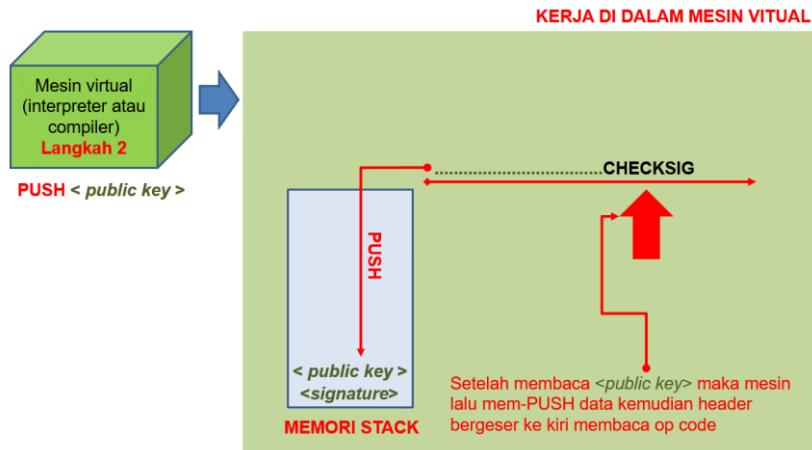
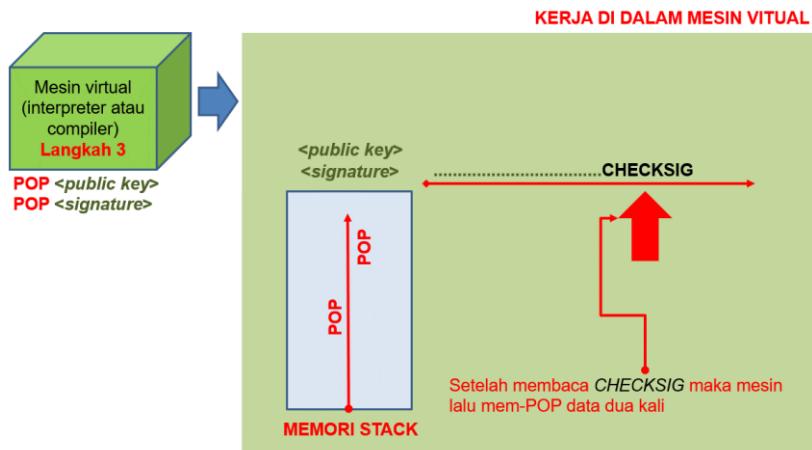
Langkah 0:

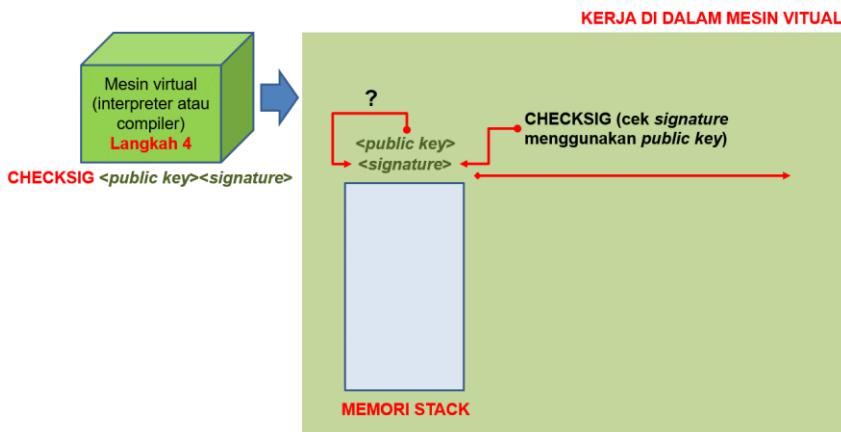


Gambar 40.18. Mesin menerima input berupa pasangan ScriptSig dan ScriptPubKey

Langkah 1:

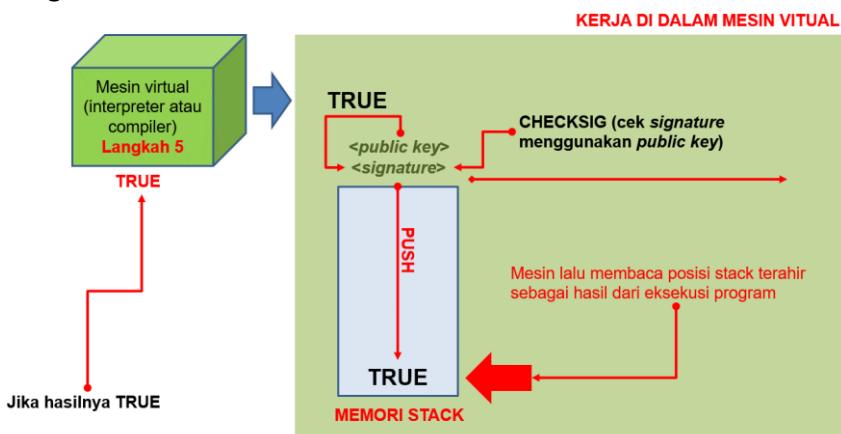


Gambar 40.19. Mesin mem-PUSH data <signature> ke stack**Langkah 2:****Gambar 40.19. Mesin mem-PUSH data <public key> ke stack****Langkah 3:****Gambar 40.19. Mesin mem-PUSH data <public key> ke stack****Langkah 4:**



Gambar 40.20. Perintah CHECKSIG memeriksa keabsahan *digital signature* menggunakan *public key*

Langkah 5:



Gambar 40.21. Jika hasil pemeriksaan adalah TRUE maka penerima dapat menggunakan bitcoin dalam output transaksi (untuk contoh ini yaitu D atau E dapat menggunakan bitcoin yang tertera pada bagian output transaksi C)

Metode pembayaran menggunakan *public key* ini (P2PK) secara umum hanya terjadi pada masa awal berjalannya *bitcoin* dan *blockchain*. Akan tetapi orang mulai merasa khawatir bahwa di masa depan sebuah komputer kuantum dapat dengan mudah menebak *private key* penerima hanya dengan melihat *public key* penerima yang

tertera secara telanjang pada bagian output transaksi. Misal bahwa *public key* D dan E tertera pada bagian output transaksi C, sehingga di masa depan dengan menggunakan komputer kuantum, orang bisa menebak *private key* D dan E hanya dengan melihat *public key* D dan E yang tertera pada bagian output transaksi C.

Ini berarti peretas (*hacker*) setelah *hacker* dapat memperoleh *private key* D dan E, maka *hacker* bisa membuat transaksi baru, misal transaksi D yang dapat secara ilegal mengakses bitcoin pada output transaksi C untuk dikirim ke dirinya sendiri.

3. Pembuatan ScriptSig dan ScriptPubKey untuk P2PKH

ScriptSig dan ScriptPubKey untuk pembayaran bitcoin menggunakan *hash public key* memiliki bentuk skrip sebagai berikut:

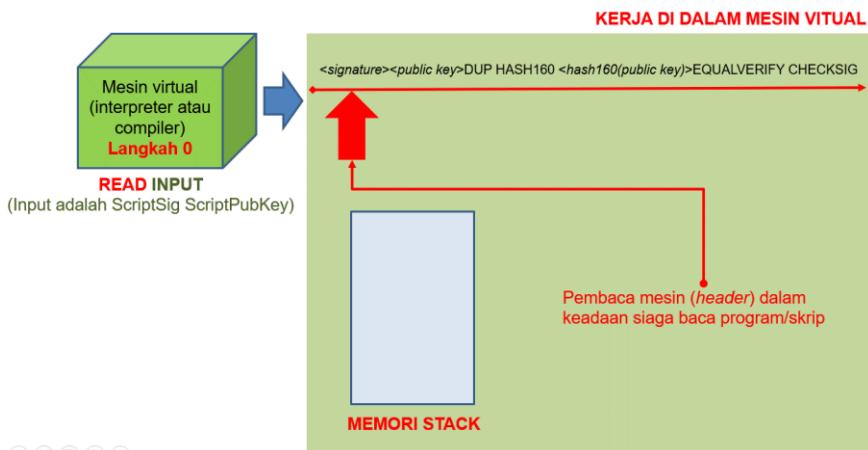
ScriptSig = <*signature*><*public key*>

ScriptPubKey = DUP HASH160 <*hash160(public key)*>EQUALVERIFY
CHECKSIG

Kode skrip ini adalah bentuk standar. Tentang mengapa menggunakan *hash public key* sebenarnya bertujuan agar *public key* tidak terlihat agar dia menjadi aman dari serangan komputer yang bisa meretas dan mendapatkan *private key*. Akan tetapi, ide mereka belum sempurna karena masih terdapat *public key* yaitu <*public key*> pada bagian ScriptSig dari transaksi.

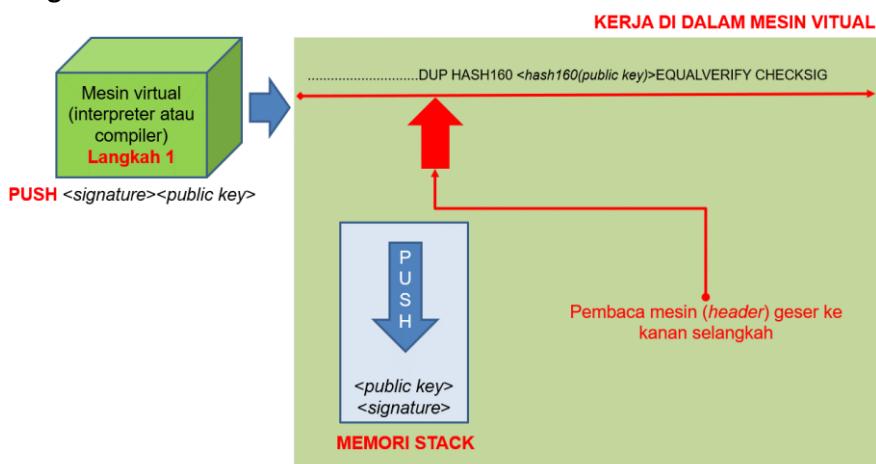
Berikut ini adalah demonstrasi bagaimana mesin virtual mengesekusi ScriptSig ScriptPubKey untuk memvalidasi transaksi. Yaitu sebagai berikut:

Langkah 0:



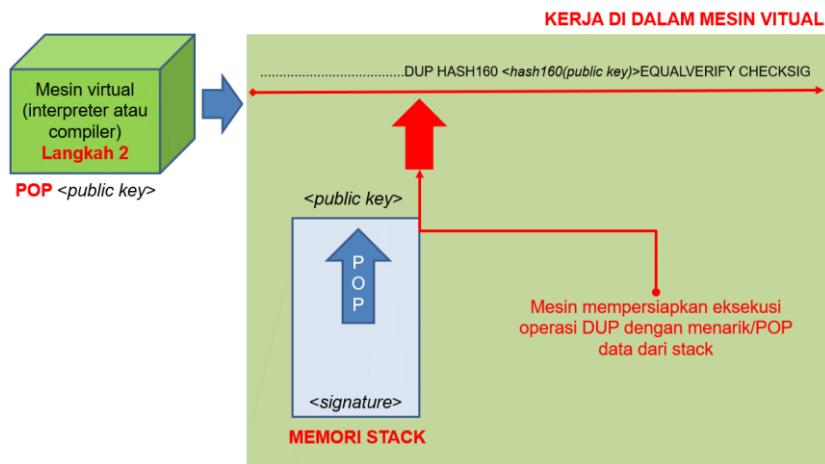
Gambar 40.22. Persiapan pembacaan input oleh mesin virtual

Langkah 1:



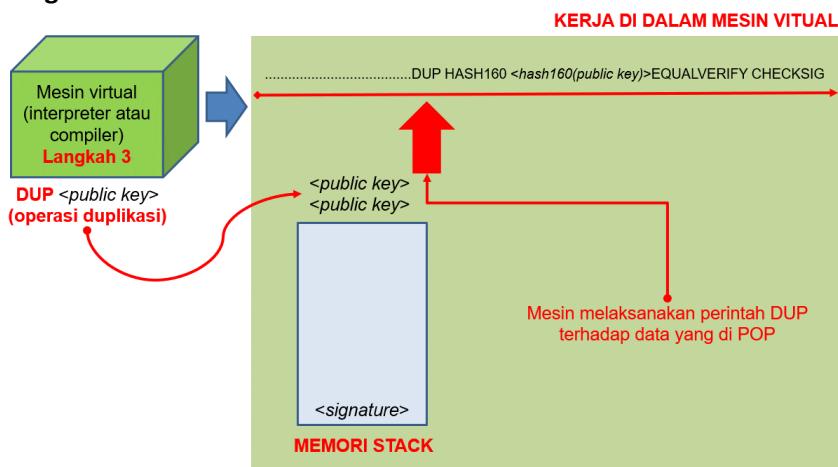
Gambar 40.23. Pembacaan data *<signature>* dan *<public key>* lalu PUSH data ke stack

Langkah 2:



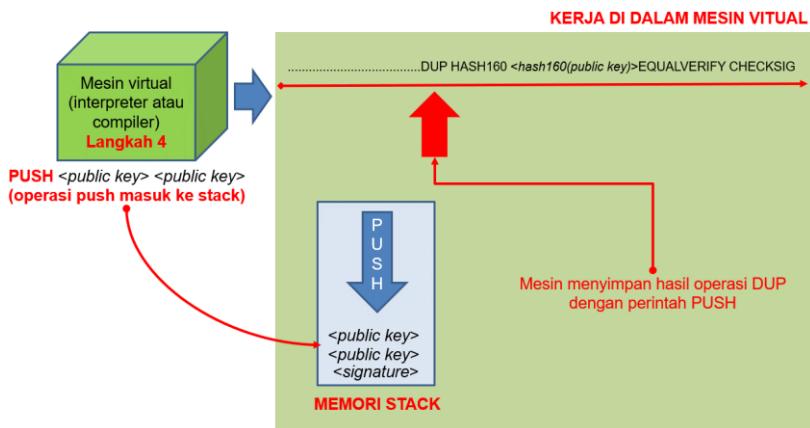
Gambar 40.24. Mesin membaca perintah DUP lalu persiapan data yang di POP dari stack

Langkah 3:

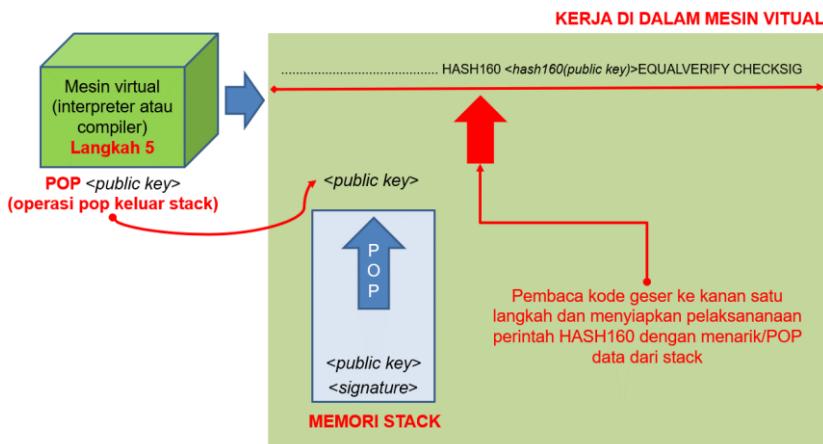


Gambar 40.25. Mesin melaksanakan perintah DUP pada skrip

Langkah 4:

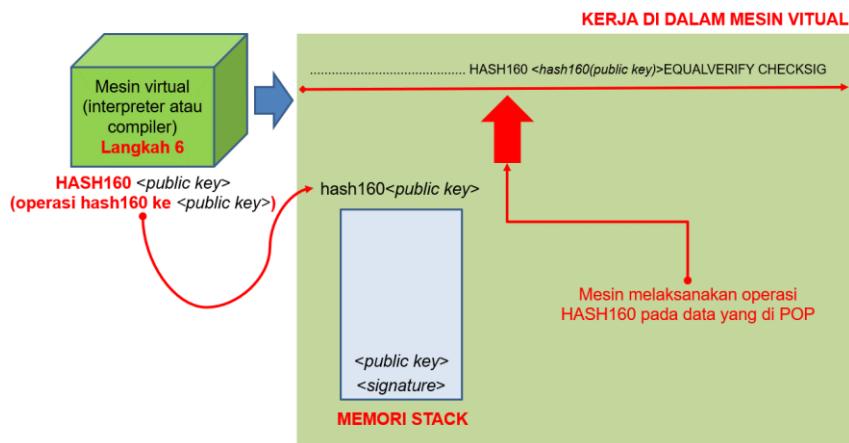


Gambar 40.26. Mesin menyimpan hasil pelaksanaan perintah DUP Langkah 5:



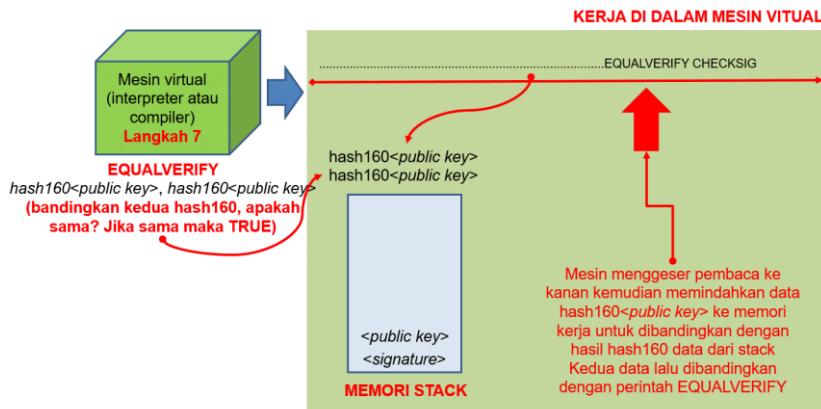
Gambar 40.27. Mesin membaca perintah Hash160 lalu menyiapkan pelaksanaan perintah dengan menarik data dari stack menggunakan perintah POP

Langkah 6:



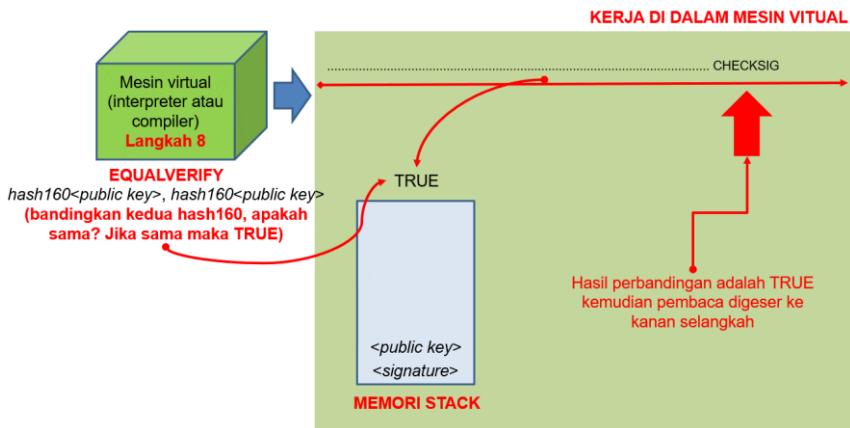
Gambar 40.28. Perintah hash160 dilaksanakan pada data yang telah di POP sebelumnya

Langkah 7:



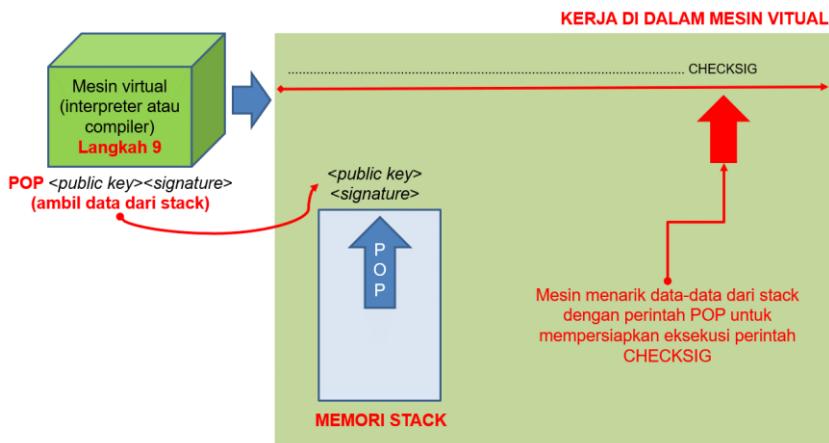
Gambar 40.29. Mesin membaca data hash160<publik key> lalu memindahkan ke memori kerja untuk persiapan dibandingkan dengan hasil hash160 data dari stack, kemudian membaca perintah EQUALVERIFY untuk membandingkan

Langkah 8:



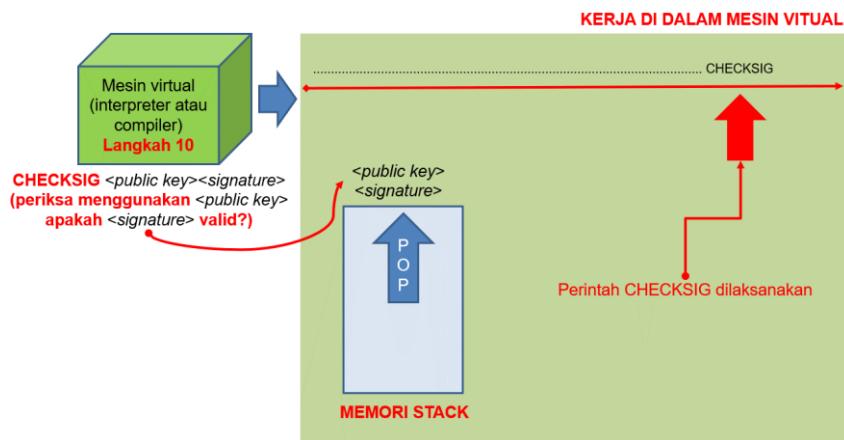
Gambar 40.30. Setelah dibandingkan diperoleh TRUE (kedua data ternyata sama), mesin lalu membacaperintah berikut

Langkah 9:



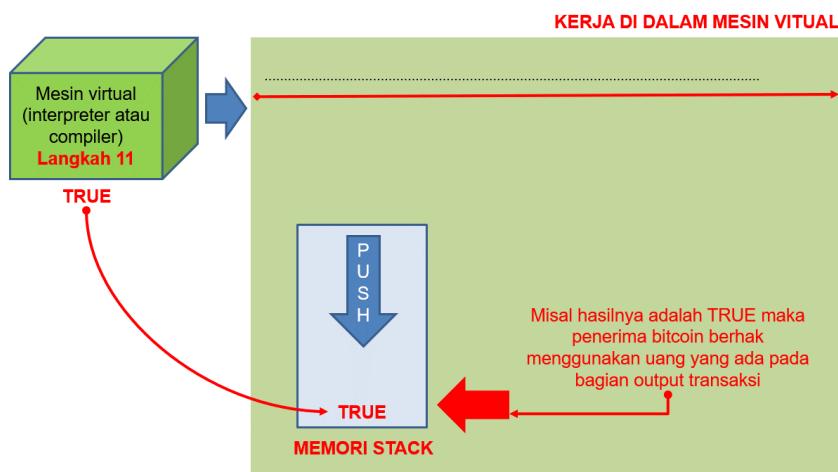
Gambar 40.31. Persiapan pelaksanaan perintah CHECKSIG dengan menarik data dari stack menggunakan perintah POP

Langkah 10:



Gambar 40.32. Perintah CHECKSIG dilaksanakan yaitu memeriksa apakah *<signature>* adalah benar dengan menggunakan data *<public key>*

Langkah 11:



Gambar 40.33. Hasil pelaksanaan perintah adalah TRUE, ini berarti verifikasi penggunaan bitcoin berhasil

Demikianlah langkah-langkah verifikasi skrip agar penerima dapat menggunakan bitcoin yang ada di dalam bagian output dari transaksi.

Tetapi pada dasarnya P2PKH bukanlah skrip yang wajib diterapkan di

dalam transaksi, dia hanya skrip yang standar. Karena pada prinsipnya skrip dibuat dengan tujuan umum atau bebas. Anda dapat membuat skrip apa saja menggunakan *opcode* atau *mnemonic* yang disediakan. Bahkan tanpa mengandung *public key* atau *hash-nya* sekalipun juga bisa digunakan sebagai skrip di dalam ScriptPubKey.

Kita dapat mengarang sebuah skrip yang berbeda sebagai berikut:

Signature = PrivateKeyPengirim(Hash(Serialisasi(Transaksi)) + Hash160(*public key*))

ScripSig = <*signature*>

ScripPubKey = <hash160(*public key*)> CHECKSIG

Skrip buatan di atas benar-benar menyembunyikan *public key* karena tidak terdapat pada sisi *ScriptSig* dan pada sisi *ScriptPubKey* kecuali *hash-nya* saja.

Ada banyak sekali kemungkinan skrip yang bisa anda buat. Akan tetapi perlu diingat bahwa blockchain mungkin akan selalu lebih dulu mendahului menghitung transaksi yang memiliki susunan skrip standar. Susunan skrip standar yaitu P2PK, P2PKH, PSMS, P2SH dan NULL_DATA. Ini karena kebanyakan node mungkin lebih mengutamakan kecepatan dan keamanan, sebab boleh jadi bentuk skrip yang tidak umum dapat mengandung *malware* yang sengaja disusun secara logis oleh pembuat skrip untuk melakukan *hacking* terhadap sistem. Sehingga mungkin transaksi yang anda buat menggunakan skrip sendiri yang tidak standar bisa jadi lama menunggu di *pool* data transaksi karena tidak ada *miner* yang berminat untuk mengambilnya kecuali jika sudah tidak ada lagi transaksi yang skripnya standar.

Di bawah ini adalah koleksi perintah atau kata yang bisa anda gunakan untuk membangun skrip, kumpulan kata yang lengkap untuk membangun skrip adalah dapat dibaca pada <https://en.bitcoin.it/wiki/Script>

Word
OP_NOP
OP_IF
OP_NOTIF

OP_ELSE
OP_ENDIF
OP_VERIFY
OP_RETURN
OP_0, OP_FALSE
N/A
OP_PUSHDATA1
OP_PUSHDATA2
OP_PUSHDATA4
OP_1NEGATE
OP_1, OP_TRUE
OP_2-OP_16
OP_TOALTSTACK
OP_FROMALTSTACK
OP_IFDUP
OP_DEPTH
OP_DROP
OP_DUP
OP_NIP
OP_OVER
OP_PICK
OP_ROLL
OP_ROT
OP_SWAP
OP_TUCK
OP_2DROP
OP_2DUP
OP_3DUP
OP_2OVER
OP_2ROT
OP_2SWAP
OP_CAT
OP_SUBSTR

OP_LEFT
OP_RIGHT
OP_SIZE

Semua transaksi yang ditampilkan di atas hanyalah model yang disederhanakan. Akan tetapi pada kenyataanya, transaksi yang tertulis di blockchain adalah transaksi yang sebenarnya dan mungkin dapat kita bedah untuk disesuaikan dengan diskusi sebelumnya tentang transaksi. Berikut ini adalah sebuah contoh struktur data dari transaksi sebenarnya di dalam blockchain.

(diambil dari <https://learnmeabitcoin.com/glossary/transaction-data>):

Transaksi:

```
c1b4e695098210a31fe02abffe9005cffc051bbe86ff33e173155bcbdc58  
21e3  
0100000017967a5185e907a25225574544c31f7b059c1a191d65b53d  
cc1554d339c4f9efc010000006a47304402206a2eb16b7b92051d0fa38  
c133e67684ed064effada1d7f925c842da401d4f22702201f196b10e6e4  
b4a9fff948e5c5d71ec5da53e90529c8dbd122bff2b1d21dc8a90121039  
b7bcd0824b9a9164f7ba098408e63e5b7e3cf90835cce19868f54f8961  
a825ffffffff014baf2100000000001976a914db4d1141d0048b1ed15839  
d0b7a4c488cd368b0e88ac00000000
```

Jika kita membedah transaksi ini maka kita dapat menyatakannya dalam potongan-potongan data yang memiliki arti sebagai berikut:

Transaksi:

```
c1b4e695098210a31fe02abffe9005cffc051bbe86ff33e173155bcbdc58  
21e3  
0100000017967a5185e907a25225574544c31f7b059c1a191d65b53d  
cc1554d339c4f9efc010000006a47304402206a2eb16b7b92051d0fa38  
c133e67684ed064effada1d7f925c842da401d4f22702201f196b10e6e4  
b4a9fff948e5c5d71ec5da53e90529c8dbd122bff2b1d21dc8a90121039  
b7bcd0824b9a9164f7ba098408e63e5b7e3cf90835cce19868f54f8961  
a825ffffffff014baf2100000000001976a914db4d1141d0048b1ed15839  
d0b7a4c488cd368b0e88ac00000000
```

Penjelasannya adalah sebagai berikut:

4. Kode yang menyatakan TXID (transaksi id) dari transaksi ini, merupakan hash dari transaksi ini sendiri
**c1b4e695098210a31fe02abffe9005cffc051bbe86ff33e173155bcbd
c5821e3**
5. Kode yang menyatakan versi dari struktur data yang digunakan transaksi ini (versi menyatakan bentuk struktur data standar atau template struktur data yang digunakan):
01000000
6. Kode yang menyatakan transaksi sebelumnya yang digunakan, ini bisa lebih dari satu transaksi yang digunakan, tetapi dalam contoh ini hanya 1 maka dikodekan 01:
01
7. Kode yang menyatakan rincian transaksi sebelumnya yang digunakan untuk transaksi ini:

**7967a5185e907a25225574544c31f7b059c1a191d65b53dcc1554d
339c4f9efc010000006a47304402206a2eb16b7b92051d0fa38c133
e67684ed064effada1d7f925c842da401d4f22702201f196b10e6e4
b4a9fff948e5c5d71ec5da53e90529c8dbd122bff2b1d21dc8a90121
039b7bcd0824b9a9164f7ba098408e63e5b7e3cf90835ccebf19868f
54f8961a825**

Kode ini dapat dipecah lagi menjadi:

**7967a5185e907a25225574544c31f7b059c1a191d65b53dcc1554d
339c4f9efc**010000006a47304402206a2eb16b7b92051d0fa38c133
e67684ed064effada1d7f925c842da401d4f22702201f196b10e6e4
b4a9fff948e5c5d71ec5da53e90529c8dbd122bff2b1d21dc8a90121
039b7bcd0824b9a9164f7ba098408e63e5b7e3cf90835ccebf19868f
54f8961a825****

Yaitu dibaca sebagai berikut:

- a. Kode yang menyatakan TXID (transaksi id) transaksi sebelumnya yang ingin digunakan bitcoinnya
**7967a5185e907a25225574544c31f7b059c1a191d65b53dcc15
54d339c4f9efc**

- b. Transaksi sebelumnya ini juga memiliki daftar output. Pilih nomor index output dari transaksi ini yang ingin digunakan, dalam contoh ini dipilih nomor index:

01000000

- c. Kode ini menyatakan panjang ukuran kode tandatangan untuk output yang dipilih:

6a

- d. Kode ini menyatakan kode atau skrip cara menandatangani output ini sebelumnya:

47304402206a2eb16b7b92051d0fa38c133e67684ed064effad
a1d7f925c842da401d4f22702201f196b10e6e4b4a9fff948e5c5
d71ec5da53e90529c8dbd122bff2b1d21dc8a90121039b7bcd0
824b9a9164f7ba098408e63e5b7e3cf90835cce19868f54f896
1a825

8. Kode yang menyatakan *sequence* dari transaksi ini, biasanya diberi kode seperti ini:

ffffffff

9. Kode yang menyatakan jumlah alamat yang dituju oleh pengiriman bitcoin, dalam contoh ini hanya ada 1 sehingga ditulis 01:

01

10. Kode yang menyatakan rincian output dari transaksi:

4baf2100000000001976a914db4d1141d0048b1ed15839d0b7a4c4
88cd368b0e88ac

Kode ini dapat dipecah lagi dalam urutan arti sebagai berikut:

4baf2100000000001976a914db4d1141d0048b1ed15839d0b7a4c4
88cd368b0e88ac

- a. Kode ini menyatakan nilai bitcoin dalam satuan Satoshi yang hendak dikirim:

4baf2100000000000

- b. Kode ini menyatakan panjang kode public key orang yang dituju:

19

- c. Kode ini menyatakan kode atau skrip public key orang yang dituju
76a914db4d1141d0048b1ed15839d0b7a4c488cd368b0e88ad
11. Kode yang menyatakan kapan transaksi ini dinyatakan valid, biasanya dibiarkan 0, mungkin nantinya dirubah oleh node (*miner*) yang memverifikasi transaksi.
00000000

Struktur data transaksi di atas dapat kita kembalikan ke dalam bentuk JSON sebagaimana sebelumnya pada contoh yang sederhana, yaitu sebagai berikut:

```
{  
  "transaction-id":  
    "c1b4e695098210a31fe02abffe9005cffc051bbe86ff33e173155bcbdc58  
    21e3",  
  "previous-transaction-id":  
    "7967a5185e907a25225574544c31f7b059c1a191d65b53dcc1554d339  
    c4f9efc",  
  "owner-pubkey": "123456789ABCDEF...",  
  "prev-owner-signature": "AABBCCDDEEFF112233..."  
}
```

BAB 2

Penerapan Blockchain

Bidang Farmasi dan Medis

Industri farmasi adalah salah satu industri terbesar dan paling kritis yang diandalkan masyarakat. Namun, terlepas dari ketergantungan mendasar farmasi pada produk dan perawatan yang diberikan, farmasi masih menghadapi tantangan di seluruh rantai pasokan mulai dari penelitian awal obat-obatan hingga konsumen akhir, demikian juga pada pemeliharaan data uji klinis yang konsisten, riset obat-obatan, penyimpanan obat, kerjasama atau kontrak perdagangan antara usaha farmasi dan pedagang besar atau unit sarana layanan.

Permasalahan pemalsuan obat yang masuk di dalam rantai distribusi obat dapat memberi kerugian yang besar bagi industri farmasi dan dampak yang buruk bagi kesehatan masyarakat.

Demikian juga pada sisi dunia kesehatan yang berkait erat dengan industri farmasi, dalam dunia perawatan kesehatan yang sangat diatur oleh regulasi pemerintah, data sering kali dapat didiamkan dan kurang dimanfaatkan. Bahkan di antara rumah sakit dalam sistem yang sama, sulit untuk melacak catatan kesehatan pasien tertentu, apalagi mengenakan biaya perawatan yang tepat. Memelihara data penyedia biaya sekitar \$ 2,1 miliar setiap tahun di seluruh sistem perawatan kesehatan. Namun, teknologi Blockchain menawarkan solusi yang menjanjikan untuk menyelesaikan tantangan ini dan menurunkan biaya.

Buku besar ledger dan blockchain yang terdesentralisasi dapat menawarkan solusi keamanan siber yang kuat untuk data layanan kesehatan, mengamankan rantai pasokan peralatan medis dan farmasi, dan menghapus pekerjaan administratif duplikat untuk perusahaan asuransi.

Sudah banyak perusahaan luar negeri yang mengambil langkah pertama mereka ke dalam proyek-proyek berbasis blockchain dengan

bergabung dengan konsorsium tertutup kecil yang menggunakan sistem buku besar terdistribusi atau blockchains diizinkan untuk mengatur dan mengamankan data bersama.

Sebagai definisi dasar, blockchain adalah buku besar yang didistribusikan. Beberapa pemangku kepentingan membentuk bagian dari jaringan blockchain yang menyimpan catatan digital. Rekaman digital ini sinkron di semua bagian jaringan dan dihubungkan dengan algoritma konsensus, yang diatur melalui matematika dan kode. Catatan disimpan secara permanen oleh semua pihak, di mana informasi hanya dapat dibuat atau dibaca (tidak seperti database tradisional di mana biasanya CRUD - buat, baca, perbarui, hapus).

Pada buku ini hendak dikemukakan bagaimana teknologi blockchain memecahkan berbagai masalah di dalam industri farmasi. Penjelasan ini dinyatakan dalam bentuk use case, tantangan dan pemesahan blockchain.

Blockchain untuk Manufaktur dan Distribusi Obat

Saat ini, rantai pasokan farmasi melibatkan sejumlah perantara dari produksi awal hingga distribusi akhir obat-obatan kepada pasien. Solusi blockchain akan menetapkan asal produk di seluruh bagian rantai pasokan ini, dengan pembaruan di setiap perantara sampai penjualan terdaftar dan produk diserahkan kepada pasien.



Gambar 2.1. Rantai pasokan (supply chain) obat

Keuntungan dari berbagai solusi blockchain dieksplorasi lebih lanjut di seluruh kategori lintasan dan lacak, penetapan harga, pembayaran, diskon, potongan harga dan pelacakan pengembalian dana, tata kelola, risiko, peraturan dan kredensial, dan mengurangi

penipuan dan obat-obatan palsu.

Use case di dalam masalah manufaktur dan distribusi ini adalah sebagai berikut:

Use Case pada rantai pasokan (*Supply Chain*):

1. Pencegahan pemalsuan obat dan peralatan medis.
2. Verifikasi keaslian obat yang dikembalikan

Tantangan yang ada (*Current Challenge*):

Mitra manufaktur dan distribusi farmasi saat ini sering menjangkau berbagai negara dan wilayah hukum, yang membuatnya sulit untuk melacak setiap obat atau perangkat medis yang didistribusikan. Karena kekurangan pelacakan yang bersifat real-time, catatan relatif mudah untuk diubah, dan kurangnya transparansi antara perantara menghasilkan sejumlah besar obat-obatan palsu dan peralatan medis yang dimasukkan ke dalam rantai pasokan.

Rantai pasokan global menjadi semakin kompleks, dengan vendor alternatif yang tidak sah berkontribusi terhadap pasar gelap farmasi. Baik pasien akhir dan produsen terkena dampak negatif, menimbulkan kerugian pada pendapatan dan perawatan kesehatan yang aman dan andal.

PwC memperkirakan bahwa \$ 163 miliar hingga \$ 217 miliar obat palsu dijual di seluruh dunia pada tahun 2015, di mana sekitar 1 persen dari semua obat yang beredar di pasar Barat (dan rata-rata sekitar 10 persen secara global) diyakini palsu. Organisasi Kesehatan Dunia (WHO) memperkirakan bahwa 8 persen perangkat medis yang beredar saat ini adalah tiruan palsu.

Untuk mencoba mengatasi masalah penipuan dan barang palsu ini, AS (Undang-Undang Keamanan Rantai Pasokan Obat atau DSCSA) dan UE (Falsified Medicine Directive atau FMD) telah memperkenalkan undang-undang yang mengharuskan sistem elektronik untuk melacak dan mengotentikasi obat saat bergerak melalui jaringan distribusi.

Demikian juga dalam wilayah hukum Indonesia sendiri, pemerintah telah berusaha mencegah hal ini dengan menerapkan aturan-aturan pembuatan obat yang baik (CPOB) dan aturan

pendistribusian obat yang baik (CDOB).

Selain itu daripada itu, melacak pengembalian barang-barang farmasi yang dikembalikan merupakan tantangan tersendiri juga. Masalah pengembalian ini rawan akan penyalahgunaan obat-obat atau peralatan medis untuk berbagai tujuan.

Peluang teknologi blockchain sebagai solusi:

Blockchain dapat digunakan sebagai satu sumber keabsahan dalam melacak obat-obatan, produk, dan perangkat medis di semua titik dalam rantai pasokan. Pembagian nomor identifikasi unik dapat digunakan untuk tidak hanya melacak asal dan memberikan keaslian, tetapi juga dapat mengatasi banyak persyaratan dari Badan pengawasan obat dan makanan (BPOM) dan berbagai peraturan lain terkait pembuatan dan pendistribusian obat yang dibuat oleh pemerintah.

Kemampuan untuk membangun privasi data dari data sensitif, sambil tetap memberikan verifikasi yang diperlukan dengan solusi blockchain, memastikan keamanan yang lebih besar antara perantara. Solusi blockchain menjadikannya lebih sulit untuk memasukkan obat-obatan palsu dan alat kesehatan ke dalam rantai pasokan dan bahkan menyediakan metode bagi pelanggan akhir untuk memeriksa keaslian produk.

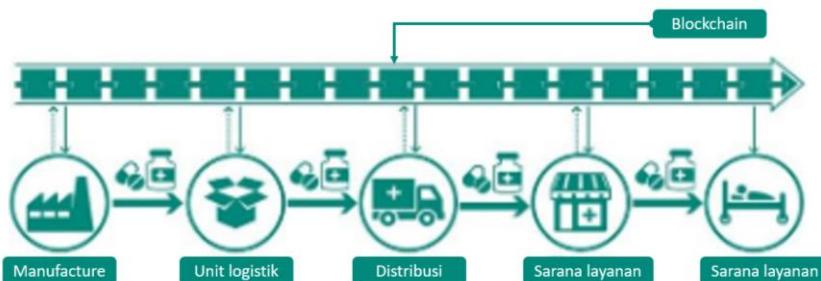
(Haq and Esuka, 2018) memberikan deskripsi dan langkah-langkah kerja blockchain pada rantai distribusi obat sehingga blockchain dapat menjadi rantai penyimpanan data perjalanan obat yang sulit untuk dipalsukan. Deskripsi dan langkah-langkah itu adalah sebagai berikut:

Cara kerja blockchain pada rantai distribusi (*supply chain*) :

1. Ketika sebuah pabrik menghasilkan produk baru, mereka akan membuat hash yang unik dan menempatkannya pada produk tersebut.
2. Produk akan terdaftar di blockchain menggunakan hash (ID unik).
3. Produk akan dianggap sebagai aset digital di jaringan blockchain,

dan hash-nya akan digunakan untuk melacaknya kapan saja di jaringan.

4. Informasi tambahan apa pun dari produk dapat disimpan secara off-chain (lepas dari blockchain) atau on-chain (terkait ke dalam blockchain) tergantung pada pilihan pabrikan.
5. Data off-chain akan digabungkan dengan data on-chain dengan menggunakan beberapa jenis pengidentifikasi. Secara konvensional, dalam sebagian besar aplikasi berbasis blockchain, sebuah hash-digest (mis. SHA-256) dari semua data off-chain dihasilkan dan dihubungkan ke data on-chain. Namun pendekatan terbaik adalah menyimpan file-file besar (mis. Gambar) off-chain dan data teks on-chain.
6. Setelah produk didaftarkan ke blockchain oleh pabrikan, kepemilikannya akan dengan mudah ditransfer ke orang lain menggunakan aplikasi seluler yang mudah digunakan.
7. Katakanlah pedagang grosir ingin membeli obat dari produsen, produsen akan secara fisik mentransfer obat ke pedagang grosir dan transaksi transfer akan didaftarkan ke blockchain secara bersamaan. Pedagang grosir akan mengulangi proses yang sama untuk mentransfer obat ke distributor, dan distributor akan melakukan bisnis yang sama dengan farmasi. Dalam Gambar 2 struktur dasar manajemen rantai phrmmacutical berbasis blockchain diberikan.

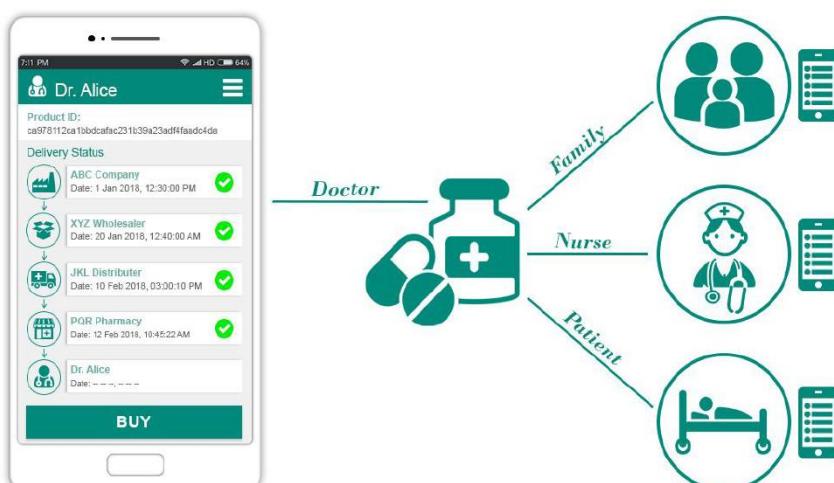


Gambar 2.2. Blockchain sebagai tulang punggung rantai distribusi obat. Sumber gambar (Haq and Esuka, 2018)

Skenario cara kerja blockchain ini dapat digambarkan sebagai

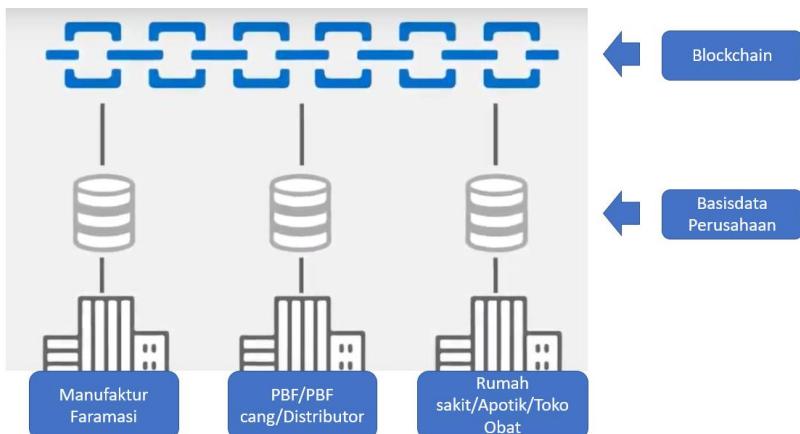
berikut:

- Misalkan Dokter Alice memerlukan obat-obatan dan dia ingin membeli jika dari apotek.
- Dengan menggunakan aplikasi seluler, Dr. Alice akan terlebih dahulu menanyakan ID obat, untuk mengonfirmasi semua perjalanananya dari produsen ke apotek.
- Jika produk tersebut asli, aplikasi seluler akan menunjukkan semua riwayatnya dan jika obat itu palsu tidak ada catatan yang akan ditampilkan.
- Setelah Dr. Alice yakin tentang orisinalitas obat-obatan itu, ia kemudian akan membelinya.
- Sama seperti dokter, pengguna lain (mis. Perawat, Keluarga dan Pasien, dll.) juga akan dapat melacak perjalanan obat-obatan tersebut. Tata letak sederhana dari antarmuka sistem ditunjukkan pada Gambar 3.



Gambar 2.3 Skenario penggunaan aplikasi blockchain
Sumber gambar (Haq and Esuka, 2018)

Arsitektur blockchain sesuai dengan skema distribusi di Indonesia sesuai CDOB diterapkan sebagaimana gambar 4.



Gambar 2.4. Skema blockchain dalam alur CDOB

Use cases pada masalah keuangan farmasi (*Financial facilitation*):

1. Penentuan harga
2. Pembayaran
3. Diskon
4. Potongan harga
5. Pelacakan pengembalian uang

Tantangan yang ada (*Current Challenge*):

Mengelola kontrak dan penetapan harga (yang harus mencakup catatan terkini dan historis dari ketentuan perjanjian perdagangan, pembayaran, diskon, potongan harga, dan pengembalian uang) dari berbagai pemangku kepentingan adalah relatif kompleks. Kontrak berbasis kertas dan kurangnya interoperabilitas antara pemasok sering menimbulkan perselisihan.

Peluang teknologi blockchain sebagai solusi:

Platform blockchain dapat digunakan untuk memfasilitasi pembayaran, penggantian, dan transaksi lainnya dengan aman. Ini dapat dicapai melalui interoperabilitas dengan sistem pembayaran yang ada atau menggunakan model berbasis token. Token dapat digunakan untuk mewakili penyimpan nilai, yang dapat ditukar antara semua

perantara pada konversi yang disepakati. Ada opsi untuk akhirnya mentransfer ini kembali ke mata uang pemerintah, jika diperlukan. Ini akan sangat berguna untuk pembayaran lintas batas antar negara untuk transaksi yang lebih cepat dan pada tingkat yang lebih murah dibandingkan dengan sistem pertukaran mata uang yang ada.

Solusi blockchain terbuka (*public blockchain*) juga dapat membantu pemain yang lebih kecil memasuki rantai produksi dan distribusi. Hambatan untuk masuk akan jauh lebih rendah dengan keamanan tambahan, berpotensi menurunkan biaya di seluruh manajemen rantai pasokan dan mekanisme pembayaran, serta transparansi di antara peserta jaringan. Ini dapat memberikan bentuk baru hubungan tepercaya antara bisnis farmasi besar yang bersejarah dan pendatang baru, yang memungkinkan akses ke pendanaan melalui model berbasis token. Pada akhirnya, ini dapat berdampak positif bagi pasien di area pasar yang kurang terlayani dalam mendapatkan akses ke produk farmasi.

Use cases:

1. Kepatuhan terhadap peraturan (CPOB, CDOB dan aturan-aturan lain pemerintah)
2. Kredensial
3. Penilaian risiko

Tantangan yang ada (*Current Challenge*):

Produk farmasi tertentu harus mematuhi persyaratan penyimpanan dan penanganan yang ketat ketika diangkut di sepanjang rantai pasokan. Persyaratan ini dapat mencakup tingkat suhu, bahan penyimpanan, penanganan, kelembaban, dan kualitas udara. Persyaratan ini diberlakukan untuk memastikan bahwa kualitas dan efektivitas obat tidak terdegradasi.

Namun, ada masalah dalam memastikan bahwa peraturan yang sama mematuhi seluruh rantai pasokan. Setiap perantara dari kelompok riset, manufaktur, dan distribusi memiliki catatan sendiri tentang metrik keselamatan ini selama transit. Seringkali dalam kasus di mana perangkat pintar tidak digunakan dan ditautkan ke database

pusat, catatan dilacak secara manual yang mengarah pada masalah lebih lanjut yang membuat audit sebelumnya hampir tidak mungkin.

Peluang teknologi blockchain sebagai solusi:

Solusi blockchain dapat digunakan untuk memberikan cara yang lebih aman dan transparan untuk melaporkan perubahan kondisi operasi. Semua perantara dan otoritas eksternal dalam rantai pasokan akan dapat menggunakan ini untuk memastikan kepatuhan yang lebih baik.

Kontrak pintar (*smart contract*) juga dapat digunakan untuk memicu peristiwa tertentu di mana kondisi ini belum terpenuhi. Salah satu contohnya adalah hukuman kepada perusahaan logistik terkait yang bertanggung jawab untuk memastikan produk diangkut dalam kondisi yang tepat. Ini akan menciptakan hubungan yang lebih tepercaya antara mitra rantai pasokan, dan konsumen akhir dengan peningkatan transparansi pada kualitas produk farmasi. Membangun platform semacam itu dapat meningkatkan operasi rantai pasokan, yang mengarah pada efisiensi yang lebih besar dan peningkatan kualitas produk.

Blockchain bagi Konsumen dan Pasien

Use cases:

1. Perawatan berbasis nilai
2. Rekam medis
3. Pengumpulan dan interoperabilitas data
4. Agregasi dan monetisasi
5. Pembagian resep obat

Tantangan yang ada (*Current Challenge*):

Data pasien biasanya benar-benar dipisahkan antara penyedia layanan kesehatan. Intervensi manual diperlukan dalam keadaan tertentu ketika berbagi informasi pasien secara langsung diperlukan.

Peluang teknologi blockchain sebagai solusi:

Berbagi resep dapat dikelola dengan lebih baik, dengan stempel waktu yang tidak dapat dilampirkan pada catatan. Ini meningkatkan kualitas data dan menyediakan layanan kesehatan yang lebih baik. Ini menyebabkan kegiatan yang biasanya pasien membawa resep obat fisik yang ditandatangani dari kunjungan rumah sakit ke apotek akan berakhir, dan ini akan mengurangi jumlah resep obat palsu.

Visi masa depan jangka panjang kemudian dapat diperluas ke salinan digital yang dapat diverifikasi dalam catatan sakit. Dikombinasikan dengan data besar dan kecerdasan buatan (AI), solusi blockchain akan dapat memprediksi dan mengelola epidemi, mengarah pada penelitian yang lebih lengkap dan mempercepat upaya kami menuju layanan kesehatan yang lebih baik.

Data yang diberikan oleh pasien atau konsumen juga dapat dimonetisasi. Ini dapat mencakup catatan medis, tes kesehatan rutin di rumah sakit, pemeriksaan gigi, tes kebugaran, dan bahkan data yang dikumpulkan dari barang yang dapat dikenakan. Ini dapat dicapai melalui platform pasar melalui penjualan dan pembelian data tanpa memaparkan informasi identitas pribadi.

Ini akan memungkinkan pengguna untuk memilih kapan data mereka digunakan oleh orang lain dan oleh siapa, memberikan insentif kepada mereka untuk melakukannya, sementara masih memiliki manfaat bersih dari memiliki data yang dapat menginformasikan pengambilan keputusan. Data yang lebih andal akan dipertukarkan sebagai akibat dari insentif untuk memberikan informasi yang berkualitas.

Terdapat banyak hal lain di dalam industri farmasi dimana blockchain dapat diterapkan selain daripada di atas.

Use cases:

1. Manajemen identitas
2. Manajemen persetujuan

Tantangan yang ada (*Current Challenge*):

Di era di mana hampir semuanya memiliki representasi digital, manajemen identitas dan persetujuan telah jauh tertinggal. Proses pengisian formulir secara manual, penandatanganan dokumen fisik, dan pelacakan data pasien telah menyebabkan banyak inefisiensi dan kehilangan peluang dalam meningkatkan layanan kesehatan pasien. Selain itu, kurangnya keamanan di sekitar data kesehatan pasien menciptakan kerentanan yang dapat mengakibatkan pelanggaran data.

Ketika populasi dunia bertambah, jumlah pasien dan, karenanya, data mereka juga meningkat secara signifikan. Ini, pada gilirannya, memberi tekanan pada proses manual yang ada dan selanjutnya dapat menghambat upaya koordinasi antara penyedia layanan kesehatan untuk mendistribusikan data pasien, yang seringkali dapat menyebabkan hilangnya informasi penting.

Peluang teknologi blockchain sebagai solusi:

Solusi Blockchain untuk manajemen identitas dan persetujuan, ditambah dengan teknologi biometrik, dapat digunakan untuk membuktikan identitas pasien sambil memberikan akses ke informasi perawatan kesehatan masa lalu.

Bidang Pendidikan dan akademik

Sebagai sasaran pertama, sistem informasi untuk nilai mahasiswa adalah area untuk menguji coba teknologi blockchain. Ada banyak kasus tentang mahasiswa yang membobol server kampus hanya untuk merubah nilai ujiannya agar mendapat nilai yang diinginkannya. Beberapa contoh kasus di bawah ini memberikan fakta-fakta bahwa ini adalah sebuah realitas yang patut di antisipasi oleh pihak kampus.

Gambar 2.5 dan gambar 2.7 memberikan berita tentang mahasiswa yang melakukan pembobolan ke server kampus di suatu kota di indonesia, Gambar 2.6 menceritakan kasus yang sama tetapi

terjadi di universitas terkenal di luar negeri.

Kesemua peristiwa ini memberikan sebuah permasalahan yang kuat bahwa server atau basisdata yang menyimpan data-data nilai mahasiswa adalah rawan untuk diretas oleh mahasiswa. Karena itu sistem blockchain diharapkan dapat menjadi solusi bagi penyimpanan data nilai mahasiswa di kampus.



Gambar 2.5. Contoh kasus mahasiswa di Manado



Gambar 2.6. Contoh kasus mahasiswa di California



Gambar 2.7. Contoh kasus mahasiswa di Bengkulu

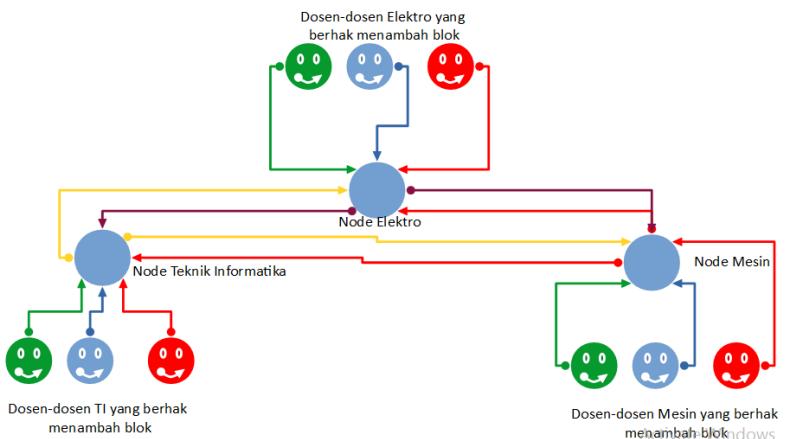
Bagaimana memecahkan masalah kerawanan data nilai mahasiswa terhadap peretasan baik dari luar maupun dari dalam menggunakan blockchain? Bagaimana mengimplementasikan teknologi blockchain menjadi fondasi bagi sistem informasi kampus?

Perguruan tinggi dapat secara sendirian membangun sebuah private blockchain dimana node-node dari sistem privat ini adalah setiap jurusan atau program studi di seluruh fakultas. Beberapa protokol dibuat sendiri dan mungkin berbeda dengan protokol-protokol blockchain standar. Dalam hal ini, misal protokol konsensus tidak dibuat berdasarkan proof of work atau proof of stake. Tetapi semata penugasan pembuatan blok terhadap dosen yang diberi otoritas untuk melakukan input data transaksi di blockchain. Tak ada imbalan kerja komputasi ataupun imbalan asset digital.

Pembuatan *blockchain* dimulai dengan mengumpulkan informasi tentang denah proses bisnis pengisian nilai dosen, lalu metadata tabel nilai mahasiswa. Dari informasi-informasi itu, dibuat meta data ledger *blockchain*. Dalam hal ini konstruksi kolom-kolom untuk pembuatan signature (kode *hash*) dan kolom transaksi. Dari sini diperoleh model data *blockchain*.

Selanjutnya pembuatan protokol untuk penambahan blok

kepada *blockchain*. Oleh karena sistem *blockchain* dibuat sebagai private *blockchain*, maka penambahan blok berbentuk penugasan dan keabsahan autentikasi. Yaitu ada sejumlah dosen yang diberikan otoritas untuk menambah blok. Walaupun demikian, setiap dosen tidak menyimpan salinan *blockchain* keseluruhan, dia hanya menyimpan salinan blok yang dibuatnya. Komputer jurusanlah yang menyimpan salinan *blockchain* secara keseluruhan. Dalam hal ini, komputer jurusanlah yang berkedudukan sebagai simpul resmi di dalam *blockchain*.



Gambar 3.2. Arsitektur private blockchain yang direncanakan

Tiap-tiap dosen diberi hak menambah sebuah blok. Sebuah blok tidak saja terdiri dari 1 transaksi akan tetapi boleh lebih dari 1 transaksi. Sehingga untuk sekali konstruksi blok, dapat terdiri dari n buah transaksi. Misalnya untuk 1 blok adalah berisi semua data nilai sebuah kelas belajar untuk 1 mata kuliah. Proses hash dilakukan pada sisi server jurusan. Setiap dosen hanya membuat kandidat blok. Setiap kandidat blok ditampung di memori penampung server (*pool*).

Selanjutnya adalah pembuatan protokol untuk melakukan verifikasi salinan blockchain secara berkala. Dalam rencana ini, tiap-tiap node saling memverifikasi file meta yang berupa rantai hash semata. Kemudian pembuatan file genesis, yaitu file pertama di tiap-tiap node.

Langkah terakhir adalah pembuatan aplikasi server di setiap

node dan aplikasi wallet di setiap dosen. Setiap dosen memiliki aplikasi wallet yang mengirim usulan penambahan blok di blockchain.

Sumber-sumber bacaan untuk bab ini:

<https://medium.com/linum-labs/revolutionizing-pharma-one-blockchain-use-case-at-a-time-e7922c8aa9a3>

<https://pharmaphorum.com/news/60-of-pharma-companies-using-or-trying-blockchain-survey/>

<https://pharmaphorum.com/views-and-analysis/five-use-cases-for-blockchain-in-pharma/>

<https://www.thepharmaletter.com/article/how-will-blockchain-impact-on-pharma>

<https://hackernoon.com/top-5-use-cases-of-blockchain-in-pharma-and-healthcare-that-you-should-know-about-77ccdd76369b>

<https://pharmaphorum.com/views-and-analysis/pharmas-manufacturing-supply-chain-needs-blockchain-innovation/>



BAB 3

Contoh Sederhana Konstruksi Blockchain

Pada bab 3 ini, dikemukakan sebuah contoh konstruksi sederhana sebuah blockchain. pada dasarnya sebuah blockchain hanyalah sebuah file blockchain yang tersimpan secara terdistribusi dan terdesentralisasi di seluruh komputer node. Akan tetapi setiap node, baik itu master node atau bukan, mestilah memiliki aplikasi yang mengelola file blockchain tersebut dimana setiap user node melakukan penambahan blok, verifikasi blok dan melaksanakan seluruh aturan dan protokol *blockchain*.

Bab 3 ini mengemukakan contoh aplikasi untuk mengelola file *blockchain* yang berupa aplikasi dengan arsitektur MVC. Aplikasi dibuat dengan bahasa pemrograman PHP dan javascript serta file blockchain ditulis dalam file teks sql atau file json.

Pembuatan *blockchain* dilakukan seperti langkah-langkah di bawah. Ini hanyalah versi penulis buku tentang bagaimana membangun *blockchain* menggunakan PHP dan javascript. Pembaca dapat memikirkan cara lain atau strategi lain yang berbeda dengan ini. Untuk versi buku ini, langkah-langkah hanya sampai pada pembuatan kode untuk membaca file *blockchain* dan hash kuncinya. Belum masuk pada kode untuk melakukan verifikasi rantai kunci, *proof of work* atau mungkin juga *proof of stake* atau jenis protokol lain untuk melakukan konsensus. Juga belum masuk untuk membuat kode *chaincode* atau *smart contract*, yang mana setiap node atau juga *master node* perlu memiliki mesin virtual untuk mengeksekusi *smart contract*. Akan tetapi jika kita menggunakan javascript sebagai bahasa penulisan *smart contract* maka itu menjadi lebih mudah karena setiap web server atau browser memiliki *compiler* (mesin virtual) untuk mengeksekusi javascript.

Langkah-langkah itu adalah sebagai berikut.



1. Pembuatan Spesifikasi blockchain
2. Pembuatan Spesifikasi node dari blockchain
3. Pembuatan Spesifikasi protokol autentikasi penambahan blok
4. Arsitektur MVC dari aplikasi node (dengan asumsi bahwa pada contoh ini, aplikasi pendukung blockchain dibuat menggunakan PHP dan setiap node di posisikan sebagai server web. Dalam hal ini, node mungkin adalah sebuah perusahaan dalam sebuah blockchain konsorsium atau blockchain *private*).
5. Kode untuk akses basisdata pada aplikasi node
6. Kode untuk login pada aplikasi node
7. File blockchain dalam versi json
8. Kode untuk menambah blok dan membaca hash

Rincian langkah-langkah itu adalah sebagai berikut:

3.1. Spesifikasi Blockchain

Blockchain dibuat sesederhana mungkin untuk memperoleh kecepatan dan keamanan yang lebih baik. Karena itu, blockchain yang dimaksud di dalam implementasi ini hanyalah sebuah file sql yang bisa di eksekusi di sebarang server basisdata, tetapi ditekankan pada server MySQL atau MariaDB bawaan XAMPP.

Spesifikasi blockchain adalah sebagai berikut:

1. Hanyalah sebuah tabel dalam sebuah basisdata
2. Tabel terdiri dari 2 kolom (*field*)
3. Kolom pertama adalah kolom nomor index dari blok
4. Kolom kedua adalah kolom blok yang ter-enkoding
5. Setiap kolom blok terdiri atas sejumlah *subfield* dan sebuah *payload-block*.
6. Setiap *payload-block* terdiri atas sejumlah transaksi.
7. Setiap transaksi terdiri atas sejumlah *subsubfield* dan *payload-transaksi*.

Rincian format file blockchain dalam format file sql yang telah dibuat adalah sebagai berikut:

```
| -- phpMyAdmin SQL Dump
```

```
-- version 4.9.0.1
-- https://www.phpmyadmin.net/
--
-- Host: 127.0.0.1
-- Waktu pembuatan: 21 Agu 2019 pada 11.38
-- Versi server: 10.4.6-MariaDB
-- Versi PHP: 7.3.8

SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET AUTOCOMMIT = 0;
START TRANSACTION;
SET time_zone = "+00:00";

/*!40101                                     SET
@OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101                                     SET
@OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */
*/;
/*!40101                                     SET
@OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;

--
-- Database: `blockchain`
--

-----
-----

-- Struktur dari tabel `blockchain`
--

CREATE TABLE `blockchain` (
  `idblok` int(11) NOT NULL,
  `blok` text NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Indexes for dumped tables
--

-- Indeks untuk tabel `blockchain`
--
```

```
ALTER TABLE `blockchain`  
    ADD PRIMARY KEY (`idblok`);  
  
--  
-- AUTO_INCREMENT untuk tabel yang dibuang  
--  
  
--  
-- AUTO_INCREMENT untuk tabel `blockchain`  
--  
ALTER TABLE `blockchain`  
    MODIFY `idblok` int(11) NOT NULL AUTO_INCREMENT;  
COMMIT;  
  
/*!40101                                     SET  
CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;  
/*!40101                                     SET  
CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;  
/*!40101                                     SET  
COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

Rincian format file blockchain dalam format file json yang sepadan adalah sebagai berikut:

```
[  
{"type": "header", "version": "4.9.0.1", "comment": "Export to JSON plugin for PHPMyAdmin"}, {"type": "database", "name": "blockchain"}, {"type": "table", "name": "blockchain", "database": "blockchain", "data": []}]
```

Rincian format file blockchain dalam format file xml yang sepadan adalah sebagai berikut:

```
<?xml version="1.0" encoding="utf-8"?>  
<!--  
- phpMyAdmin XML Dump  
- version 4.9.0.1  
- https://www.phpmyadmin.net  
-  
- Host: 127.0.0.1  
- Waktu pembuatan: 21 Agu 2019 pada 11.38  
- Versi server: 10.4.6-MariaDB  
- Versi PHP: 7.3.8  
-->
```

```
<pma_xml_export version="1.0"
xmlns:pma="https://www.phpmyadmin.net/some_doc_url/"
>
<!--
- Structure schemas
-->
<pma:structure_schemas>
    <pma:database name="blockchain"
collation="latin1_swedish_ci" charset="latin1">
        <pma:table name="blockchain">
            CREATE TABLE `blockchain` (
                `idblok` int(11) NOT NULL
                AUTO_INCREMENT,
                `blok` text NOT NULL,
                PRIMARY KEY (`idblok`)
            ) ENGINE=InnoDB DEFAULT
CHARSET=latin1;
        </pma:table>
    </pma:database>
</pma:structure_schemas>

<!--
- Database: 'blockchain'
-->
<database name="blockchain">
    <!-- Tabel blockchain -->
</database>
</pma_xml_export>
```

Rincian *field* dan sub *field* blok dan transaksi dinyatakan dalam spesifikasi berikut:

1. Index blok:
Menyatakan nomor blok. Merupakan *primary key* yang bersifat otomatis dan *autoincrement*.
2. Blok
Blok terdiri dari enkoding ke dalam teks beberapa *field* seperti:
 - Hash dari blok sebelumnya
 - Kode dari versi blockchain
 - Kode id node pembuatnya
 - Panjang dari ukuran *payload-block*

- Merkle tree hash dari semua transaksi yang dimasukkan ke dalam blok
 - *Payload-block* berupa transaksi-transaksi yang dimuat oleh blok
 - Hash dari blok sekarang
3. Transaksi
- Kode idtransaksi, bisa merupakan hash dari transaksi itu sendiri
 - Panjang dari transaksi ini

3.2. Spesifikasi Node Blockchain

Pada arsitektur *private blockchain* ini, node yang berfungsi sebagai titik di dalam blockchain yang bisa menambahkan blok ke dalam blockchain dibuat sebagai sebuah server web tersendiri. Sehingga secara umum jika terdapat 3 buah node yang berpartisipasi di dalam *private blockchain* maka terdapat 3 buah web server yang berdiri sendiri.

Di dalam arsitektur yang dibuat ini, node diimplementasikan sebagai sebuah server web XAMPP dengan basisdata MySQL, sehingga disana ada 3 node yang dibuat maka terdapat 3 buah server XAMPP yang masing-masing berdiri sendiri. Akan tetapi seluruh server itu berbagi file basisdata blockchain yang sama. Manakala sebuah node melakukan penambahan blok di dalam basisdata MySQL, maka dia mengumumkan (*broadcast*) ke seluruh node yang lain sehingga node yang lain juga melakukan penambahan blok yang sama. Tentunya dengan aturan-aturan penambahan yang terverifikasi sebagai sebuah protokol blockchain.

Secara teknis, rincian spesifikasi node adalah sebagai berikut:

1. Setiap node diimplementasikan sebagai sebuah server web XAMPP yang telah mencakup server basisdata MySQL (MariaDB). Pilihan implementasi ini karena *private blockchain* masih dalam tahapan pengembangan. Sebagai konsekuensi, pilihan ini tidak terlalu aman jika memasuki tahap implementasi yang sebenarnya di atas internet, sehingga dapat diganti dengan alternatif lain seperti

- WAMP, MAMP atau LAMP yang bisa diposting di *cloud*, atau jenis server web lain yang bukan keluarga Apache.
2. Setiap node diimplementasikan sebagai sebuah aplikasi web dengan arsitektur MVC.
 3. Setiap node memiliki dua sisi aplikasi.
 - a. Sisi pertama aplikasi node adalah aplikasi *system admin* dari node dimana dia bisa menambahkan blok dan melakukan verifikasi blok. Juga untuk melakukan autentikasi user dan node lain di dalam lingkungan blockchain. Fungsi aplikasi admin dari node adalah sebagai berikut:
 - Untuk melakukan penambahan blok
 - Untuk melakukan verifikasi manual selain verifikasi otomatis
 - Untuk melakukan otentikasi manual selain otentikasi otomatis
 - Node bisa menghidupkan dan mematikan fungsi verifikasi otomatis
 - Node bisa melihat semua list node yang terdaftar di blockchain serta rincian yang boleh terlihat atau diijinkan terlihat oleh super admin.
 - Node bisa melihat fork jika ada dan menghapus fork secara manual atau otomatis.
 - b. Sisi kedua aplikasi node adalah aplikasi user, dimana sebarang user yang diberi otoritas dapat menggunakananya untuk membaca blockchain.
- Aplikasi node memiliki fungsi sebagai berikut:
- Sebagai halaman untuk melakukan *searching* blok, baik per blok atau sejumlah blok (*range* blok).
 - Sebagai halaman untuk melakukan dekoding blok, yaitu menerjemahkan blok ke dalam tampilan sejumlah *field* yang mudah di baca oleh manusia.
 - Sebagai halaman untuk mencetak sejumlah blok

- Sebagai halaman untuk mengekspor sejumlah data blok ke tipe file lain misal excel, xml, json, csv atau file sql.
 - Sebagai halaman untuk mengunduh blockchain bagi user yang berminat melakukan verifikasi manual.
4. Setiap orang dapat mendaftar sebagai node di dalam *private blockchain* dengan terlebih dulu mendapat persetujuan dari 51% atau lebih dari seluruh anggota blockchain (node).
 5. Setiap orang dapat mendaftar sebagai user bukan node di dalam *private blockchain* dengan terlebih dulu mendapat persetujuan node dimana user mendaftar.
 6. Setiap node memiliki hak untuk menambahkan blok di dalam *blockchain*.
 7. Setiap node memiliki kewajiban melakukan verifikasi blok untuk setiap penambahan blok yang terjadi di dalam ekosistem *blockchain*.
 8. Setiap node memiliki hak untuk menyetujui atau menolak seseorang untuk menjadi node.
 9. Setiap node harus menginstal XAMPP atau setaranya agar dapat menginstal aplikasi *blockchain*.
 10. Setiap node memiliki basisdata:
 - a. Basisdata blockchain (inti)
 - b. Basisdata pendukung
 - Terdiri atas tabel autentikasi user dan node
 - Tabel transaksi yang belum atau mau dimasukkan ke blok. Tabel ini berfungsi sebagai buffer transaksi yang datang dari user atau admin node sendiri.

3.3. Spesifikasi Protokol Autentikasi Penambahan Blok

1. Arsitektur MVC dari aplikasi node
2. Koding untuk akses basisdata pada aplikasi node (basisdata utamanya terdiri dari tabel yang menyatakan blockchain dan bersifat tidak boleh diedit dan dihapus serta tabel yang menyatakan *pool* atau penampung transaksi yang bersiap hendak dimasukkan ke



dalam blockchain. Tabel *pool* transaksi ini bisa diedit, bisa dihapus. Juga beberapa tabel lain seperti tabel admin dan sebagainya yang bisa diedit dan dihapus).

3. Koding untuk login pada aplikasi node

Protokol pendaftaran sebagai node:

- User mendaftar pada sistem, memberi username dan password.
- Sistem membuat hash dinamis dari username dan password itu (hashuser)
- Sistem hanya mengingat hash itu dan memnghapus username dan password.
- Sistem lalu membuat tabel hashuser dengan arsitektur:

iduser | hashuser | idnodepenadantangandigital | digitsignatur

Tabel ini langsung terisi sampai semua iduser node yang ada di sistem, tetapi hanya sistem yang digital signurnya terisi untuk pertama kali.

- Sistem lalu membuat digital signatur untuk hashuser, sistem sebagai pembuat digital signatur pertama.
- Sistem lalu membroadcast tabel hashuser keseluruhan node dan meminta tanda tangan mereka.
- Setiap node lalu menandatangani secara dinamis sesuai idnodenya, (untuk memecahkan masalah jika suatu saat node lupa password sehingga dia merubah password, ini tetap tidak berubah, untuk verifikasi, orang hanya perlu melihat hash terakhir)
- Tandatangan node = tanda tangan node sebelumnya (ini ga usah, biar ga perlu ada urutan) yang dihash secara dinamis menggunakan password node sekarang.
- Tandatangan node = hash dinamis menggunakan password node terhadap hashuser.
- Node lalu menyimpan salinan tabel hashuser untuk dirinya sendiri sebagai bahan untuk verifikasi tanda tangan di masa depan.

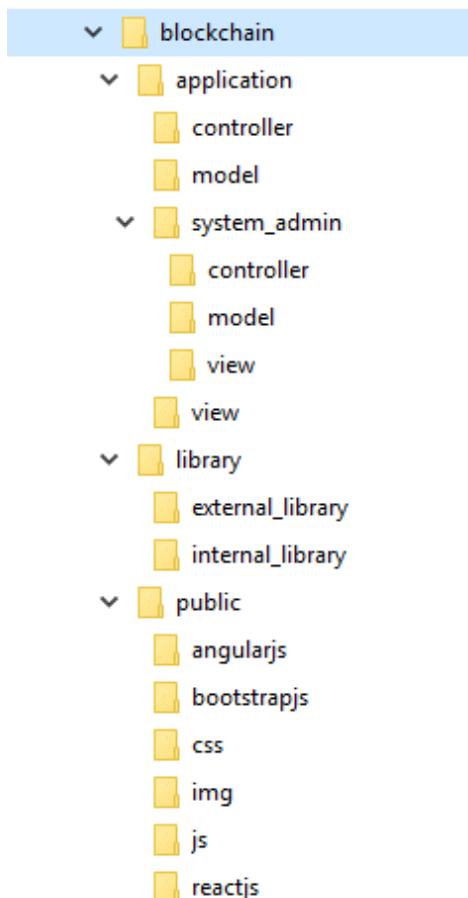
Protokol verifikasi node penambah blok:



- Saat user hendak melakukan penambahan blok, user memasukkan username dan password ke sistem. lalu sistem melakukan hash dinamis sehingga terbentuk hashuser.
- Sistem lalu menandatangani hashuser menggunakan passwordnya dan mengecek/membandingkan dengan tanda tangan sebelumnya pada tabel hashuser yang dipegang lalu menceklist jika absah.
- Sistem lalu mengirim hashuser kesemua node untuk meminta verifikasi node-node lain, jika 51% atau lebih node menyatakan OK maka user boleh menambahkan (atau login?) blok.
- Jika kurang dari 51% maka sistem meminta user untuk mendaftar ulang (buat username baru dan password baru dan semua file blockchain yang dia simpan ditimpa oleh yang baru), lalu proses PROTOKOL PENDAFTARAN dimulai yang baru.
- Bagaimana jika ada node yang memverifikasi ada merubah passwordnya sehingga tidak sepakat lagi dengan tandatangan lama yang dia simpan? ini dipecahan oleh konsep 51%. dimana user tinggal daftar ulang dan proses PROTOKOL PENDAFTARAN dimulai yang baru. atau bahwa ketika node merubah password maka dia juga merubah tandatangan hashuser di tabel hasuser yang dia pegang.

Arsitektur MVC dari aplikasi node:

Arsitektur MVC dari aplikasi node dinyatakan dalam struktur folder sebagai berikut:



Gambar 4.1. Arsitektur MVC aplikasi node

Kode untuk akses basisdata pada aplikasi node

Untuk sementara koding yang dihasilkan untuk mengakses basisdata adalah sebagai berikut:

```
<?php
/*
-----+
Akses ke basisdata

Source By: Aslan Alwi ->
Version: 1.0
Copyright: Bebas untuk didistribusikan dan
dimodifikasi selama signature ini diikutkan.
Versi PHP: 5.2.0.
```

Warnings: Anda harus merubah beberapa fungsi yang depreceated jika menjalankannya di versi PHP 7 ke atas.

```
-----*/  
  
function koneksi_server($host,$user,$pass,$database)  
{  
$koneksi=mysql_connect($host,$user,$pass);  
if (! $koneksi){echo "Gagal Koneksi..! (fungsi  
model.php-->koneksi_server)".mysql_error();} //else  
{echo "KONEKSI OKE";}  
mysql_select_db($database) or die ("Database Tidak Ada  
(fungsi model.php-->koneksi_server)".mysql_error());  
}  
  
page_row_Recordset1($pageNum_Recordset1,$maxRows_Recordset1,$tabel,$database) {  
$startRow_Recordset1 = $pageNum_Recordset1 *  
$maxRows_Recordset1;  
mysql_select_db($database) or die ("Database Tidak Ada  
(fungsi model.php-->page)".mysql_error());  
$query_Recordset1 = "SELECT * FROM $tabel";  
$query_limit_Recordset1 = sprintf("%s LIMIT %d, %d",  
$query_Recordset1, $startRow_Recordset1,  
$maxRows_Recordset1);  
$Recordset1 = mysql_query($query_limit_Recordset1) or  
die(mysql_error());  
$row_Recordset1 = mysql_fetch_assoc($Recordset1);  
return $row_Recordset1;  
}  
  
function  
page_Recordset1($pageNum_Recordset1,$maxRows_Records  
et1,$tabel,$database) {  
$field=array();  
$field=penarik_key_model($tabel,$database);  
$startRow_Recordset1 = $pageNum_Recordset1 *  
$maxRows_Recordset1;  
mysql_select_db($database) or die ("Database Tidak Ada  
(fungsi model.php-->page)".mysql_error());  
$query_Recordset1 = "SELECT * FROM $tabel ORDER BY  
$field[0] DESC";  
$query_limit_Recordset1 = sprintf("%s LIMIT %d, %d",  
$query_Recordset1, $startRow_Recordset1,  
$maxRows_Recordset1);  
$Recordset1 = mysql_query($query_limit_Recordset1) or  
die(mysql_error());
```

```
return $Recordset1;
}

function
page_Recordset1_byquery ($pageNum_Recordset1,$maxRows
_Recordset1,$query_Recordset1,$database) {
$startRow_Recordset1      =      $pageNum_Recordset1      *
$maxRows_Recordset1;
mysql_select_db($database) or die ("Database Tidak Ada
(fungsi model.php-->page)".mysql_error());
// $query_Recordset1 = "SELECT * FROM $tabel ORDER BY
$field[0] DESC";
$query_limit_Recordset1 = sprintf("%s LIMIT %d, %d",
$query_Recordset1,                      $startRow_Recordset1,
$maxRows_Recordset1);
$Recordset1 = mysql_query($query_limit_Recordset1) or
die(mysql_error());
return $Recordset1;
}

function
page_Recordset1_search ($pageNum_Recordset1,$maxRows_
Recordset1,$tabel,$database,$kolom_cari,$key_cari) {
$startRow_Recordset1      =      $pageNum_Recordset1      *
$maxRows_Recordset1;
mysql_select_db($database) or die ("Database Tidak Ada
(fungsi model.php-->page)".mysql_error());
$query_Recordset1 = "SELECT * FROM $tabel WHERE
$kolom_cari LIKE '%$key_cari%'";
$query_limit_Recordset1 = sprintf("%s LIMIT %d, %d",
$query_Recordset1,                      $startRow_Recordset1,
$maxRows_Recordset1);
$Recordset1 = mysql_query($query_limit_Recordset1) or
die(mysql_error());
return $Recordset1;
}

//Fungsi penghitung jumlah rekord
function jumlah_rekord ($tabel,$database) {
mysql_select_db($database) or die ("Database Tidak Ada
(fungsi model.php-->jumlah_rekord)".mysql_error());
$query_Recordset1 = "SELECT * FROM $tabel";
$all_Recordset1 = mysql_query($query_Recordset1);
$totalRows_Recordset1           =
mysql_num_rows($all_Recordset1);
```

```
return $totalRows_Recordset1;
}

//Fungsi penghitung jumlah rekord
function jumlah_rekord_query ($query,$database) {
mysql_select_db($database) or die ("Database Tidak Ada
(fungsi model.php-->jumlah_rekord)".mysql_error());
$all_Recordset1 = mysql_query($query);
$totalRows_Recordset1 =
mysql_num_rows($all_Recordset1);
return $totalRows_Recordset1;
}

function
total_halaman($maxRows_Recordset1,$tabel,$database) {
$TotalPages_Recordset1 = ceil(jumlah_rekord
($tabel,$database) /$maxRows_Recordset1)-1;
return $totalPages_Recordset1;
}

//Fungsi penarik nama-nama key dari tabel
function penarik_key_model($tabel,$database)
{

$kolom=array();
$fields = mysql_list_fields($database,$tabel);
$columns = mysql_num_fields($fields);
for ($i = 0; $i < $columns; $i++) {
    $kolom[$i]=mysql_field_name($fields, $i);
}
//echo "MASUK $database $tabel ";
//print_r($kolom);
return $kolom;
}

//Fungsi penarik nama-nama kolom dari tabel:
function
penarik_kolom_model($kolom_value,$kolom_label,$tabel
,$database)
{
mysql_select_db($database) or die ("Database Tidak Ada
(fungsi
model.php-->penarik_kolom_model)".mysql_error());
$query_Recordset1 = "SELECT DISTINCT $kolom_value,
$kolom_label FROM $tabel";
$Recordset1 = mysql_query($query_Recordset1) or
die(mysql_error());
```

```
//$row_Recordset1 = mysql_fetch_assoc($Recordset1);
return $Recordset1;
}

//Fungsi penarik dengan query user defined
function user_defined_query_model($query,$database)
{
mysql_select_db($database) or die ("Database Tidak Ada
(fungsi      model.php-->user_defined      _query_model
)".mysql_error());
$query_user=array();
$query_user=          mysql_query($query)           or
die(mysql_error());
return $query_user;
}

//Fungsi insersi universal:
function
general_insertion_model($kiriman,$tabel,$database) {
$field=penarik_key_model($tabel,$database);
$query_insert = "INSERT INTO $tabel(";
$query_insert .="$field[1]";
for ($i=2;$i<count($field);$i++)   {$query_insert
.=",$field[$i]";}
/*foreach ($field as $isi) {if(!$field[0])  &&
!($field[1])){$query_insert .=",$isi";} */ 
$query_insert .=") VALUES (";
$query_insert .="'$kiriman[1]'";
for ($i=2;$i<count($field);$i++)   {$query_insert
.=",'$kiriman[$i]'";}
/*foreach ($kiriman as $isi) {if(!$isi==$kiriman[0])
&& !($isi==$kiriman[1]))$query_insert .=",'$isi'";} */
$query_insert .=")";
mysql_select_db($database) or die ("Database Tidak Ada
(fungsi      model.php-->general_insertion_model
)".mysql_error());
mysql_query($query_insert) or die(mysql_error());
//echo "Info: Query Sukses";
}

function penafsir_NULL($kiriman) {
foreach ($kiriman as $isi) {if ($isi=="") {$isi=NULL;}}
return $kiriman;
}

function
```

```
general_update_model($kiriman,$tabel,$database) {
mysql_select_db($database) or die ("Database Tidak Ada
(fungsi           model.php-->general_update_model
)."mysql_error());
$field=penarik_key_model($tabel,$database);
foreach ($kiriman as $isi) {if ($isi=="") {$isi=NULL;}}
for ($i=1;$i<count($kiriman);$i++) {
if($kiriman[$i]==NULL) {} else {mysql_query("UPDATE
$tabel      SET      $field[$i]='$kiriman[$i]'      WHERE
$field[0]=$kiriman[0]" )  or die("UPDATE  $tabel  SET
$field[$i]='$kiriman[$i]'          WHERE
$field[0]=$kiriman[0]" );
}
//echo "Info: Perubahan Sukses      ";
}

function
general_update_model_cart($kiriman,$tabel,$database)
{
mysql_select_db($database) or die ("Database Tidak Ada
(fungsi           model.php-->general_update_model
)."mysql_error());
//$field=penarik_key_model($tabel,$database);
//foreach ($kiriman as $isi) {if ($isi=="")
{$isi=NULL;}}
//for ($i=1;$i<count($kiriman);$i++) {
//if($kiriman[$i]==NULL) {} else {mysql_query("UPDATE
$tabel      SET      $field[$i]='$kiriman[$i]'      WHERE
$field[0]=$kiriman[0]" )  or die("UPDATE  $tabel  SET
$field[$i]='$kiriman[$i]'          WHERE
$field[0]=$kiriman[0]" );
mysql_query("UPDATE          $tabel          SET
$field[$i]='$kiriman[$i]'          WHERE
$field[0]=$kiriman[0]" )  or die("UPDATE  $tabel  SET
$field[$i]='$kiriman[$i]'          WHERE
$field[0]=$kiriman[0]" );
}
//echo "Info: Perubahan Sukses      ";
}

function hapus_rekord ($tabel,$database) {
mysql_select_db($database) or die ("Database Tidak Ada
(fungsi model.php-->hapus_rekord ).mysql_error());
$field=penarik_key_model($tabel,$database);
if ((isset($_GET['id'])) && ($_GET['id'] != ""))
{
$id=$_GET['id'];
$deleteSQL      =      "DELETE      FROM      $tabel      WHERE
```

```

$field[0] = $id";
mysql_select_db($database);
$Result1      =      mysql_query($deleteSQL)      or
die(mysql_error());
echo "Info: Penghapusan Sukses      ";
}
}

function hapus_rekord_cart ($id,$tabel,$database) {
mysql_select_db($database) or die ("Database Tidak Ada
(fungsi model.php-->hapus_rekord )".mysql_error());
$field=penarik_key_model($tabel,$database);
if ((isset($_GET['id'])) && ($_GET['id'] != ""))
{
$id=$_GET['id'];
$deleteSQL      =      "DELETE      FROM      $tabel      WHERE
$field[0] = $id";
mysql_select_db($database);
$Result1      =      mysql_query($deleteSQL)      or
die(mysql_error());
//echo "Info: Penghapusan Sukses      ";
}
}

?>

```

3.4. Kode untuk login pada aplikasi node:

```

/*
-----*
Login ke akun blockchain

Source By: Aslan Alwi ->
Version: 1.0
Copyright: Bebas untuk didistribusikan dan
dimodifikasi selama signature ini diikutkan.
Versi PHP: 5.2.0.
Warnings: Anda harus merubah beberapa fungsi yang
depreceated jika menjalankannya di versi PHP 7 ke
atas.
-----*/
function Logout($logoutGoTo) {
// $logoutGoTo = "home_admin.php";
session_start();
unset($_SESSION['MM_Username']);

```

```
unset($_SESSION['MM_UserGroup']);
if      ($logoutGoTo    !=      "")      {header("Location:
$logoutGoTo");}
session_unregister('MM_Username');
session_unregister('MM_UserGroup');

exit;
}

//Fungsi pemeriksa entry kosong dalam array
function notvoid($var)
{
    return((($var)));
}

//Fungsi untuk melakukan enkripsi campuran, contoh
$chiper="md5.md5.md5.sha1.md5"
        dengan
$plaintext="aku" adalah chiper dengan susunan
md5(md5(md5(md5("aku"))));
function enkrip($onechiper,$plaintext) {
    $arraychiper=explode(".",$onechiper);
    $hasil=$plaintext;
    foreach ($arraychiper as $k) {

        if($hasil==$plaintext){$hasil=implode("",array
($k,"","","$hasil","\""));}else{$hasil=implode("",arr
ay($k,"",$hasil,"\"));}
    }
    $oke=implode("",array("\$hasil= ",$hasil,";"));

    eval($oke);
    return $hasil;
}

function symbol_to_chiper($symbol,$alphabet,$chiper)
{
// $alphabet=array(aa,a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p
// ,q,r,s,t,u,v,w,x,y,z,A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P
// ,Q,R,S,T,U,V,W,X,Y,Z);
if(in_array($symbol,$alphabet)){
    $key=array_search($symbol,$alphabet);
    $x=ceil($key/2)-($key/2);
    if($x==0) {
        for ($i = 1; $i <= $key; $i++) {

            if($hasil){$hasil=implode(".",array($chiper[0]
,$hasil));}else{$hasil=$chiper[0];}
        }//end $i
    }
}
```

```
// end $x
else {
    for ($i = 1; $i <= $key; $i++) {

        if($hasil){$hasil=implode(".",array($chiper[1]
,$hasil));}else{$hasil=$chiper[0];}
        //end $i
    }
}//end in_array
else {return FALSE;}
return $hasil;
}

//Fungsi pendukung konstruksi chiper:
function string_to_array($string)
{//,$alphabet,$chiper_genap,$chiper_ganjil) {
$string=explode(" ",$string);
$string=array_filter($string,notvoid);
$string=implode("",$string);
$string=str_split($string);
return $string;
}
```

3.5. Kode untuk file blockchain versi json lain (sebagai perbandingan)

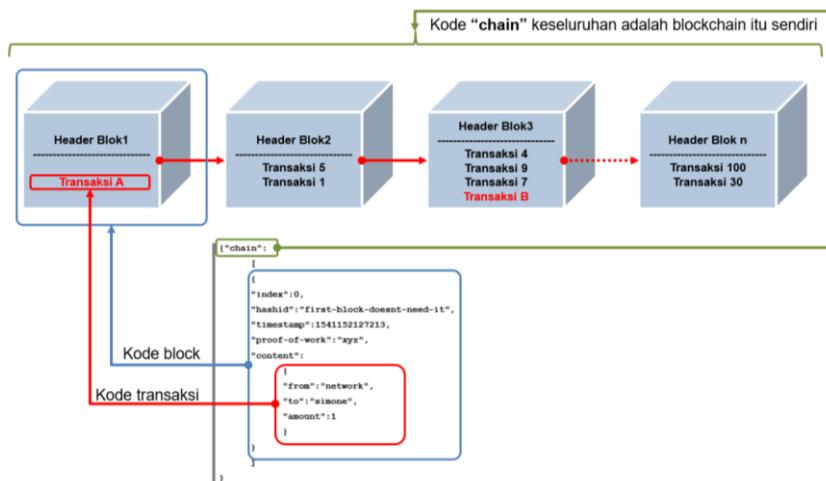
File blockchain dalam sistem ini dibuat berbentuk file json. File inilah yang menjadi blockchain yang sesungguhnya dan didistribusikan secara sempurna (tidak terfragmentasi) ke seluruh node di dalam jejala sistem blockchain. Kode contoh untuk struktur data blockchain ini diambil dari <https://medium.com/@simogol/understanding-how-a-blockchain-works-with-php-and-json-ee9305d7d2c1>. Akan tetapi anda dapat membuat struktur data sendiri yang berbeda dengan konsep yang lain. Ini hanyalah contoh konstruksi selain cara yang dikemukakan sebelumnya, tetapi bisa digunakan langsung atau dipilih sebagai struktur data untuk sistem blockchain yang ingin dibangun.

Struktur isi atau struktur data dari file blockchain adalah sebagai berikut:

```
{"chain":
```

```
[ {
  "index":0,
  "hashid":"first-block-doesnt-need-it",
  "timestamp":1541152127213,
  "proof-of-work":"xyz",
  "content":
  {
    "from":"network",
    "to":"simone",
    "amount":1
  }
} ]
```

Kode ini adalah kode block di dalam blockchain dan kode di dalam transaksi. Kedudukan kode ini di dalam blockchain adalah dalam ilustrasi berikut:



Gambar 4.2 Kedudukan kode file blockchain di dalam blockchain

Penjelasan kode adalah sebagai berikut:

- Index: nomor dari blok yang bersifat unik

- Hashid: kode hash blok ini.
- Timestamp: menyatakan waktu pembuatan blok.
- Proof of work: protokol yang digunakan untuk memverifikasi rantai hash blok ini.
- Content: transaksi yang dikandung oleh blok ini. Transaksi boleh lebih dari satu.
- Pada bagian “Content”, isian “form” menyatakan asal dari transaksi.
- “to” menyatakan transaksi diberikan kesiapa.
- “amount” menyatakan jumlah atau nilai transaksi yang diberikan.

3.6. Kode untuk membaca blok dan membaca hash serta membuat hash baru untuk blok berikut

Kode ini juga diambil dari situs yang memberi contoh kode membaca file blockchain dan cara membaca hashnya. <https://medium.com/@simogol/understanding-how-a-blockchain-works-with-php-and-json-ee9305d7d2c1>. Kode ini memuat fungsi minimal untuk membaca dan memvalidasi blockchain. Kita dapat memperluas kode ini untuk sistem blockchain yang sesuai.

```
<?php
class DAO {
    function read_all() {
        try {
            $jsondata =
file_get_contents(dirname(dirname(__FILE__))."/chain
.json");
            $arr_data = json_decode($jsondata, true);
            return $arr_data;
        }
        catch(Exception $e) {
            echo "Error: " . $e->getMessage();
            exit();
        }
    }
    function get_previous_hashid($chain) {
        $lastEl = array_values(array_slice($chain, -
1))[0];
    }
}
```

```
        return $lastEl["hashid"];
    }
    function
get_new_hashid($previous_hashid,$index,$timestamp,$c
ontent) {
    $full_string =
$previous_hashid.$index.$timestamp.$content;
    $hash = hash('sha256',$full_string);
    return $hash;
}
function read_content($content) {
    $arr_content = json_decode($content);
    return $arr_content;
}
?>
```

3.7. Contoh kode PHP lain yang mengimplementasikan blok dan blockchain secara sederhana dan telah menerapkan *proof of work* pada kodennya.

Contoh kode ini ini dibuat atas nama akun github haydenyoung. Kode bisa diunduh pada alamat:

<https://github.com/knowledgearcdotorg/phpblockchain/commit/4e44ff4db8f0f89ededc5dd0fe1d467a4875223a>

Secara umum, blockchain sederhana yang dibuat terdiri atas 2 kelas. Yaitu kelas block dan kelas blockchain. Kelas block nantinya digunakan untuk membuat *instance* blok-blok di dalam blockchain. Kelas blockchain sendiri memiliki struktur data *chain* yang berupa *array* dimana setiap *item* adalah blok yang dibuat menggunakan *instance* kelas blok.

Kelas blok memiliki metode utama yaitu calculateHash() yang berfungsi untuk menghitung hash dari blok itu sendiri, menggunakan. Secara standar kelas ini menggunakan hash SHA256. Metode ini ditulis secara sederhana sebagai:

```
public function calculateHash()
{
    return hash("sha256", $this->index.$this-
>previousHash.$this->timestamp.((string)$this-
```

```
| >data) . $this->nonce);  
| }
```

Keseluruhan kode untuk kelas blok adalah sebagai berikut:

```
<?php  
class Block  
{  
    public $nonce;  
  
    public function __construct($index, $timestamp,  
    $data, $previousHash = null)  
    {  
        $this->index = $index;  
        $this->timestamp = $timestamp;  
        $this->data = $data;  
        $this->previousHash = $previousHash;  
        $this->hash = $this->calculateHash();  
        $this->nonce = 0;  
    }  
  
    public function calculateHash()  
    {  
        return hash("sha256", $this->index . $this->previousHash . $this->timestamp . ((string)$this->data) . $this->nonce);  
    }  
}  
?>
```

Selanjutnya kelas blok ini disimpan sebagai block.php dan digunakan pada kelas blockchain. Pembuatan kelas blockchain adalah sebagai berikut:

```
<?php  
require_once("./block.php");  
  
/**  
 * A simple blockchain class with proof-of-work  
(mining).  
 */  
class BlockChain  
{  
    /**  
     * Instantiates a new Blockchain.
```

```
 */
public function __construct()
{
    $this->chain = [$this->createGenesisBlock()];
    $this->difficulty = 4;
}

/**
 * Creates the genesis block.
 */
private function createGenesisBlock()
{
    return new Block(0, strtotime("2017-01-01"),
"Genesis Block");
}

/**
 * Gets the last block of the chain.
 */
public function getLastBlock()
{
    return $this->chain[count($this->chain)-1];
}

/**
 * Pushes a new block onto the chain.
 */
public function push($block)
{
    $block->previousHash = $this->getLastBlock()-
>hash;
    $this->mine($block);
    array_push($this->chain, $block);
}

/**
 * Mines a block.
 */
public function mine($block)
{
    while      (substr($block->hash,      0,      $this-
>difficulty) != str_repeat("0", $this->difficulty))
    {
        $block->nonce++;
        $block->hash = $block->calculateHash();
    }
}
```

```
        echo "Block mined: ".$block->hash."\n";
    }

    /**
     * Validates the blockchain's integrity. True if
     the blockchain is valid, false otherwise.
    */
    public function isValid()
    {
        for ($i = 1; $i < count($this->chain); $i++)
{
    $currentBlock = $this->chain[$i];
    $previousBlock = $this->chain[$i-1];

        if ($currentBlock->hash != $currentBlock-
>calculateHash()) {
            return false;
        }

        if      ($currentBlock->previousHash      !=
$previousBlock->hash) {
            return false;
        }
    }

    return true;
}
?>
```

Blok pertama pada blockchain dinyatakan sebagai *Genesis block*. Pembuatan *genesis block* pada kode di atas diimplementasikan pada potongan kode:

```
private function createGenesisBlock()
{
    return new Block(0, strtotime("2017-01-01"), "Genesis
    Block");
}
```

Blok ini diberi indeks 0 dan memiliki hash sebelumnya (*previous hash*) bernilai null. Ini karena tidak ada blok sebelumnya.

Pembuatan blok baru dimplementasikan di dalam kelas blockchain ini dalam bentuk fungsi sebagai berikut:

```
| public function push($block)
```

```
{  
$block->previousHash = $this->getLastBlock()->hash;  
$this->mine($block);  
array_push($this->chain, $block);  
}
```

Proses penambangan (*mining*) blok diimplementasikan dalam bentuk fungsi seperti di bawah dan ini adalah kode yang menerapkan *proof of work*.

```
public function mine($block)  
{  
while (substr($block->hash, 0, $this->difficulty) !==  
str_repeat("0", $this->difficulty)) {  
$block->nonce++;  
$block->hash = $block->calculateHash();  
}  
echo "Block mined: ".$block->hash."\n";  
}
```

Proof of work yaitu pembuatan hash yang harus memenuhi syarat tingkat kesulitan yang ditetapkan. Dalam contoh ini, tingkat kesulitan dinyatakan dalam variabel *difficulty*. Tingkat kesulitan dalam contoh kode ini diletakkan pada bagian konstruktor kelas blockchain sebagai berikut:

```
public function __construct()  
{  
    $this->chain = [$this->createGenesisBlock()];  
    $this->difficulty = 4;  
}
```

Tingkat kesulitan bernilai 4 artinya bahwa hash blok yang hendak ditambahkan ke blockchain harus sedemikian rupa memiliki 4 digit karakter 0 pada sub string awal dari hash.

Misal kode menemukan hash:

789sdhjksddd98dsdhkjdj\$kjhkjdkgjh

Maka pencarian hash harus diulang sedemikian rupa sehingga diperoleh hash berbentuk:

000078yhd8whsjhahlkdhshjsdhkdx\$

Hash yang memiliki 4 digit karakter 0 didepannya:

000078yhd8whsjhahlkdhshjsdhkdx\$

Hash memenuhi tingkat kesulitan 4, karena ada 4 digit karakter 0 didepannya

Gambar 4.4. Ilustrasi hash dengan tingkat kesulitan 4

Percobaan untuk menemukan hash dengan 4 digit 0 didepannya adalah dengan menambahkan *nonce* pada akhir bentuk string dari blok. Nonce terus menerus dinaikkan nilainya agar hash menghasilkan 4 digit 0 di depannya.

Nonce didefinisikan sebagai variabel di kelas block:

```
public $nonce;

    public function __construct($index, $timestamp,
$data, $previousHash = null)
{
    $this->index = $index;
    $this->timestamp = $timestamp;
    $this->data = $data;
    $this->previousHash = $previousHash;
    $this->hash = $this->calculateHash();
    $this->nonce = 0;
}
```

Kemudian ditambahkan pada perhitungan hash blok (kode pada kelas block) dengan cara sebagai berikut:

```
public function calculateHash()
{return hash("sha256", $this->index.$this->previousHash.$this-
>timestamp.((string)$this->data).$this->nonce);}}
```

Nonce ditambahkan pada akhir string blok (konkatenasi string semua nilai komponen blok) lalu dinaikkan nilainya terus menerus sampai nilai hash dengan 4 digit 0 didepannya ditemukan

Gambar 4.5 Nonce ditambahkan pada blok yang sudah dibuat string

Kode lengkap proses menaikkan nonce ini ini dapat dilihat pada fungsi mine yang ada pada kelas blockchain.

```
public function mine($block)
{
    while (substr($block->hash, 0, $this->difficulty) !==
str_repeat("0", $this->difficulty)) {

        //Nilai nonce dinaikkan terus menerus sampai hash
        //dengan 4 digit 0 didepannya ditemukan.
        $block->nonce++;
        $block->hash = $block->calculateHash();
    }
    echo "Block mined: ".$block->hash."\n";
}
```

Selanjutnya, kode untuk melakukan validasi apakah sebuah blok memiliki hash yang konsisten hash yang tertera dalam dirinya dan apakah hash blok sebelumnya (*previous hash*) yang tertera dalam dirinya adalah konsisten dengan hash blok sebelumnya adalah dilaksanakan dengan potongan kode sebagai berikut sebagaimana ditulis dalam potongan kode kelas blockchain.

```
public function isValid()
{
    for ($i = 1; $i < count($this->chain); $i++) {
        $currentBlock = $this->chain[$i];
        $previousBlock = $this->chain[$i-1];
        if      ($currentBlock->hash      !=      $currentBlock-
>calculateHash()) {return false;}
        if      ($currentBlock->previousHash != $previousBlock-
>hash) {return false;}
    }
    return true;
}
```

Contoh penggunaan kode blockchain ini adalah sebagai berikut:

```
<?php
require_once(__DIR__.'/../blockchain.php');

/*
Set up a simple chain and mine two blocks.
*/

//Inisiasi kelas blockchain untuk keperluan penambahan
//blok didalam blockchain.
```

```
$testCoin = new BlockChain();

//Proses penambahan blok
echo "mining block 1...\n";
$testCoin->push(new      Block(1,      strtotime("now"),
"amount: 4"));

//Proses penambahan blok
echo "mining block 2...\n";
$testCoin->push(new      Block(2,      strtotime("now"),
"amount: 10"));

//Menulis hasil penambahan blockchain dalam format
json
echo json_encode($testCoin, JSON_PRETTY_PRINT);

?>
```

Contoh pembuatan blockchain tetapi kemudian dilakukan perubahan secara ilegal atau *hacking* terhadap salah satu blok, kemudian proses validasi dijalankan untuk mengidentifikasi block yang dihack.

```
<?php
require_once(__DIR__.'/../blockchain.php');

/*
Hack the chain, changing values in the first block.
*/

//Inisisasi kelas blockchain untuk keperluan
penembahan blok baru di dalam blockchain
$testCoin = new BlockChain();

//Pembuatan blok baru
echo "mining block 1...\n";
$testCoin->push(new      Block(1,      strtotime("now"),
"amount: 4"));

//Pembuatan blok baru
echo "mining block 2...\n";
$testCoin->push(new      Block(2,      strtotime("now"),
"amount: 10"));

//Uji validasi yang menguji konsistensi rantai hash
echo "Chain valid: ".($testCoin->isValid() ? "true" :
```

```
"false")."\n";  
  
//Hacking terhadap salah satu blok dengan merubah  
nilainya  
echo "Changing second block..."\n";  
$testCoin->chain[1]->data = "amount: 1000";  
$testCoin->chain[1]->hash      =      $testCoin->chain[1]-  
>calculateHash();  
  
//Uji validasi untuk mendeteksi rusaknya rantai hash  
karena proses hacking di atas.  
echo "Chain valid: ".($testCoin->isValid() ? "true" :  
"false")."\n";  
  
?>
```

3.8. Implementasi blockchain pada sisi client menggunakan javascript (dapat juga berarti pada sisi server dengan menggunakan nodejs)

Berikut ini adalah contoh kode implementasi blockchain menggunakan javascript. Sumber asli kode dapat dilihat pada <https://medium.com/swlh/building-a-bitcoin-like-blockchain-in-javascript-e7b4d922ae3>. Dalam buku ini hanya dibuat sangat sedikit perubahan agar kode-kode itu berjalan di sisi client yang mengakses server, tidak hanya sebagai uji coba yang berjalan di konsol.

Untuk versi javascript tidak diberikan penjelasan tentang bagaimana kode-kode ini berkerja karena sama saja penjelasannya pada bagian kode PHP, lagi pula ditulis dengan sintaks yang mirip pada kode sumber untuk PHP, sehingga mudah untuk menganalogikan penjelasannya.

Sebagaimana contoh kode blockchain yang menggunakan PHP dan berjalan di sisi server. Kode untuk blockchain pada sisi client secara mendasar juga terbagi dua, yaitu kode yang membangun block yang dirangkum dalam kelas block dan kode yang membangun blockchain keseluruhan yang ditulis dalam kelas blockchain.

Kelas block untuk javascript yang ditulis terpisah dalam satu file sebagai block.js adalah sebagai berikut:

```
class Block {  
    constructor(index, previousHash, timestamp, data) {
```

```
this.index = index;
this.previousHash = previousHash;
this.timestamp = timestamp;
this.data = data;
this.nonce = 0;
this.hash = calculateHash(this); // defined later
}
}

Block.prototype.mineBlock = function(difficulty) {
    this.nonce = 0;
    var zeros = "0".repeat(difficulty);

    while (this.hash.substring(0, difficulty) != zeros) {
        this.nonce++;
        this.hash = calculateHash(this);
    }

}
```

Kemudian buat sebuah fungsi untuk menghitung hash secara terpisah dalam satu file, misal diberi nama calculateHash.js, sebagai berikut:

```
function calculateHash(block) {
    return sha256(block.index + block.previousHash +
        block.timestamp + block.data +
        block.nonce);
}
```

Kemudian buat kelas blockchain dalam satu file yang diberi nama Blockchain.js. Rincian kode kelas ini adalah sebagai berikut:

```
class Blockchain {
constructor(difficulty) {
this.difficulty = difficulty;
this.blocks = [];
// Add Genesis Block
var genesisBlock = new Block(0, null, Date.now(), "Genesis block");
genesisBlock.mineBlock(this.difficulty);
this.blocks.push(genesisBlock);
}
}
```

```
Blockchain.prototype.newBlock = function (data) {
var latestBlock = this.blocks[this.blocks.length - 1];
];
return new Block(latestBlock.index + 1, latestBlock.
hash, Date.now(), data);
}

Blockchain.prototype.addBlock = function(block) {
block.mineBlock(this.difficulty);
this.blocks.push(block);
}

Blockchain.prototype.isFirstBlockValid = function()
{
var firstBlock = this.blocks[0];
if (firstBlock.index != 0)
return false;

if (firstBlock.previousHash != null)
return false;

if (firstBlock.hash == null ||
calculateHash(firstBlock) != firstBlock.hash)
return false;

return true;
}

Blockchain.prototype.isValidBlock = function (block,
previousBlock) {
if (previousBlock.index + 1 != block.index)
return false;

if (block.previousHash == null || 
block.previousHash != previousBlock.hash)
return false;

if (block.hash == null ||
calculateHash(block) != block.hash)
return false;

return true;
}

Blockchain.prototype.isBlockchainValid = function()
{
if (!this.isFirstBlockValid())
```

```
return false;
for (var i = 1; i < this.blocks.length; i++) {
var block = this.blocks[i];
var previousBlock = this.blocks[i - 1];
if (!this.isValidBlock(block, previousBlock))
return false;
}
return true;
}

Blockchain.prototype.display = function() {
for (var i = 0; i < this.blocks.length; i++) {
var block = this.blocks[i];
var str = "Block #" + block.index + " ["
+ "previousHash: " + block.previousHash + ", " +
"timestamp: " + block.timestamp + ", " +
"data: " + block.data + ", " +
"hash: " + block.hash + "]";
//console.log(str);
return str;
}
}
```

Selanjutnya sebuah antarmuka pada sisi client (misal diberi nama antarmuka.html dapat dibuat sebagai berikut:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Eksperimen blockchain sisi client</title>
<script src="Block.js"></script>
<script src="Blockchain.js"></script>
<script src="calculateHash.js"></script>

<!--Tambahkan pustaka fungsi hash sha256-->
<script src="https://cdnjs.cloudflare.com/ajax/libs/js-sha256/0.9.0/sha256.min.js" integrity="sha256-cVdRFpfbdE04SloqhkavI/PJBWCr+TuyQP3WkIKaiYo=" crossorigin="anonymous"></script>

<script>
var blockchain = new Blockchain(4);
var block1 = blockchain.newBlock("Second Block");
```

```
blockchain.addBlock(block1);
var block2 = blockchain.newBlock("Third Block");
blockchain.addBlock(block2);
var block3 = blockchain.newBlock("Fourth Block");
blockchain.addBlock(block3);
</script>

</head>
<body>
<h2>JavaScript Alert</h2>

<button onclick="validitas()">Cek validitas hash blo
ck</button>
<br/><br/>
<button onclick="display()">Tampilkan blockchain</bu
tton>
<br/><br/>
<button onclick="tambahkanBlok()">Tambahkan Blok</bu
tton>
<br/><br/>
<button onclick="validitas()">Cek lagi hash blok</bu
tton>
<br/><br/>
<button onclick="display()">Tampilkan blockchain</bu
tton>

<script>
function validitas() {alert("Blockchain Validity: " +
+ blockchain.isBlockchainValid());}
function display() {alert(blockchain.display());}
function tambahkanBlok() {
var block4 = new Block(12, "falseHash", Date.now(),
"Block Invalid");
blockchain.addBlock(block4);
alert("Blok baru telah ditambahkan");
}
</script>
</body>
</html>
```

3.9. Implementasi blockchain menggunakan nodeJS

Kode pada pasal 3.8 dapat dirubah sedikit sehingga bersifat *server side* atau berjalan di sisi server dengan menjalankannya menggunakan nodeJS. Sedikit perubahan pada kode di atas

menyebabkan kita dapat menjalankannya pada konsol nodeJS atau command line dimana nodeJS sudah terinstal. Bahkan kita dapat membuatnya menjadi sebuah aplikasi *package* nodeJS yang dapat diinstall menggunakan npm. Kode dengan sedikit perubahan ini telah saya tuliskan dengan lengkap dan dapat diunduh pada github yaitu di https://github.com/elangbijak4/Simple_Blockchain_nodejs/tree/master/jsblockchain

Penulisan ulang kode pada pasal 3.8 adalah sebagai berikut:

Terlebih dulu buat file calculateHash.js sebagai berikut:

```
hashok= require('./sha256.min.js');
function calculateHash(block) {
return hashok.sha256(block.index + block.previousHas
h + block.timestamp + block.data + block.nonce);
}
exports.calculateHash = calculateHash;
```

Lalu buat kelas Block dalam file Block.js sebagai berikut:

```
hitunghash = require('./calculateHash.js');
class Block {
constructor(index, previousHash, timestamp, data) {
this.index = index;
this.previousHash = previousHash;
this.timestamp = timestamp;
this.data = data;
this.nonce = 0;
this.hash = hitunghash.calculateHash(this); // de
fined later
}
}

Block.prototype.mineBlock = function(difficulty) {
this.nonce = 0;
var zeros = "0".repeat(difficulty);
while (this.hash.substring(0, difficulty) != zer
os) {
this.nonce++;
this.hash = hitunghash.calculateHash(this);
}
}
exports.Block = Block;
```

Kemudian buat kelas Blockchain di dalam Blockchain.js sebagai berikut:

```
blok = require('./Block.js');
hitunghash = require('./calculateHash.js');
class Blockchain {
    constructor(difficulty) {
        this.difficulty = difficulty;
        this.blocks = [];
        // Add Genesis Block
        var genesisBlock = new Block.Block(0, null, Date.now(), "Genesis block");
        genesisBlock.mineBlock(this.difficulty);
        this.blocks.push(genesisBlock);
    }
}

Blockchain.prototype.newBlock = function (data) {
    var latestBlock = this.blocks[this.blocks.length - 1];
    return new Block.Block(latestBlock.index + 1, latestBlock.hash,
                           Date.now(), data);
}

Blockchain.prototype.addBlock = function(block) {
    block.mineBlock(this.difficulty);
    this.blocks.push(block);
}

Blockchain.prototype.isFirstBlockValid = function () {
    var firstBlock = this.blocks[0];

    if (firstBlock.index != 0)
        return false;

    if (firstBlock.previousHash != null)
        return false;

    if (firstBlock.hash == null ||
        hitunghash.calculateHash(firstBlock) != firstBlock.hash)
        return false;
```

```
        return true;
    }

Blockchain.prototype.isValidBlock = function (block, previousBlock) {
    if (previousBlock.index + 1 != block.index)
        return false;

    if (block.previousHash == null ||
        block.previousHash != previousBlock.hash)
        return false;

    if (block.hash == null ||
        hitunghash.calculateHash(block) != block.hash)
        return false;

    return true;
}

Blockchain.prototype.isBlockchainValid = function () {
    if (!this.isFirstBlockValid())
        return false;

    for (var i = 1; i < this.blocks.length; i++) {
        var block = this.blocks[i];
        var previousBlock = this.blocks[i - 1];

        if (!this.isValidBlock(block, previousBlock))
            return false;
    }

    return true;
}

Blockchain.prototype.display = function() {
    for (var i = 0; i < this.blocks.length; i++) {
        var block = this.blocks[i];
        var str = "Block #" + block.index + " [" +
                  "previousHash: " + block.previousHash +
                  ", " +
                  "timestamp: " + block.timestamp +
                  ", " +
                  "data: " + block.data + ", " +

```

```
        "hash: " + block.hash + "]";
    //console.log(str);
    return str;
}
}
exports.Blockchain = Blockchain;
```

Kemudian buat aplikasi yang nantinya dipanggil di konsol nodeJS sebagai aplikasi blockchain yang berjalan di atas nodeJS. Aplikasi dalam contoh ini diberi nama MyBlockchain.js atau sembarang nama uang bisa anda berikan. Kita dapat menambahkan fungsi yang bertindak sebagai server web dalam kode ini sesuai imajinasi, akan tetapi dalam contoh ini fungsi itu tidak dimasukkan, cukup hanya dalam bentuk yang sederhana yang dapat berjalan di nodeJS. Anda cukup memanggilnya di command line sebagai “node MyBlockchain.js”. Kode sumber untuk aplikasi ini adalah sebagai berikut:

```
rantaiblok = require('./Blockchain.js');
Block = require('./Block.js');
var blockchain = new rantaiblok.Blockchain(4);
var block1 = blockchain.newBlock("Second Block");
blockchain.addBlock(block1);
var block2 = blockchain.newBlock("Third Block");
blockchain.addBlock(block2);
var block3 = blockchain.newBlock("Fourth Block");
blockchain.addBlock(block3);

console.log("Blockchain Validity: " + blockchain.isBlockchainValid());

console.log(blockchain.display());

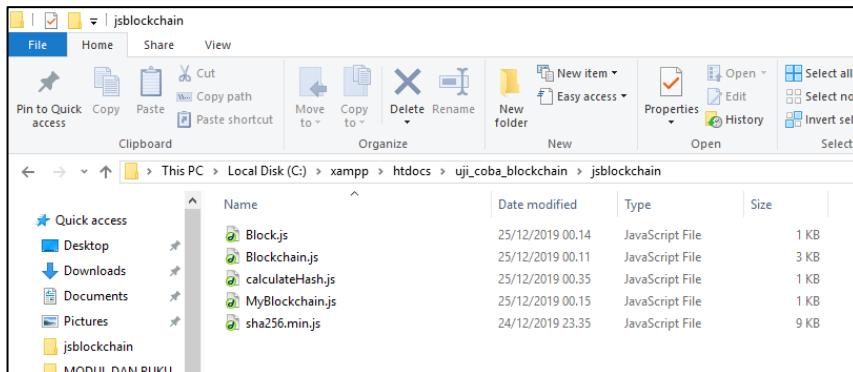
var block4 = new Block.Block(12, "falseHash", Date.now(), "Block Invalid");
blockchain.addBlock(block4);

console.log("Blockchain Validity: " + blockchain.isBlockchainValid());

console.log(blockchain.display());
```

Buat semua file dalam satu folder sebagaimana gambar 4.6.

Tetapi sebenarnya dapat sembarang saja asal tautan referensi pada halaman kode disesuaikan dengan posisi file-file tersebut.



Gambar 4.6. Semua file jika dalam satu folder

Hasil eksekusi kode blockchain ini di NodeJS adalah sebagai berikut:

```
C:\xampp\htdocs\uji_coba_blockchain\jsblockchain>node MyBlockchain.js
```

[ENTER]

Blockchain Validity: true

Block #0 [previousHash: null, timestamp: 1577209408946, data:

Genesis block, hash:

```
0000af089011cf28fcaec28075b0a5b5fd69a41c3bf78696952714c3c934
27d3]
```

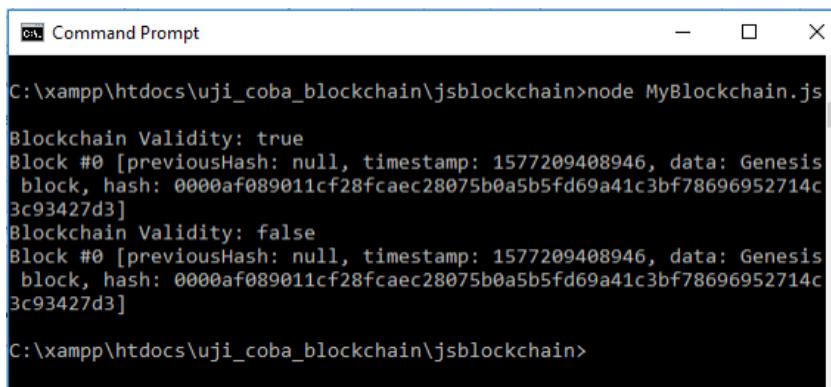
Blockchain Validity: false

Block #0 [previousHash: null, timestamp: 1577209408946, data:

Genesis block, hash:

```
0000af089011cf28fcaec28075b0a5b5fd69a41c3bf78696952714c3c934
27d3]
```

Gambar 4.7 memberikan hasil *screenshot* manakala aplikasi blockchain ini dijalankan menggunakan nodeJS.

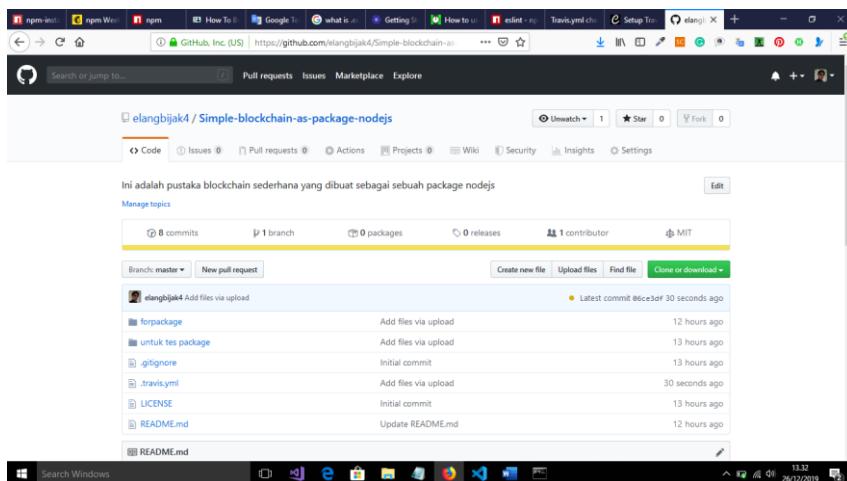


```
C:\xampp\htdocs\uji_coba_blockchain\jsblockchain>node MyBlockchain.js
Blockchain Validity: true
Block #0 [previousHash: null, timestamp: 1577209408946, data: Genesis
block, hash: 0000af089011cf28fcaec28075b0a5b5fd69a41c3bf78696952714c
3c93427d3]
Blockchain Validity: false
Block #0 [previousHash: null, timestamp: 1577209408946, data: Genesis
block, hash: 0000af089011cf28fcaec28075b0a5b5fd69a41c3bf78696952714c
3c93427d3]
C:\xampp\htdocs\uji_coba_blockchain\jsblockchain>
```

Gambar 4.7. Hasil eksekusi aplikasi pada nodeJS

3.10. Implementasi blockchain sebagai package atau pustaka di dalam nodeJS

Dengan merubah sedikit kode sumber pada pasal 3.9, kita membuatnya menjadi sebuah *package* NodeJS yang terpublikasi dan bisa digunakan dimana saja untuk membangun aplikasi blockchain di dalam lingkup nodeJS. Sedikit perubahan kode itu dapat dilihat pada <https://github.com/elangbijak4/Simple-blockchain-as-package-nodejs>. Package ini dapat dikembangkan lebih lanjut untuk membangun sebuah *private blockchain* atau *consorsium blockchain* yang lebih kompleks.



Gambar 4.8. Implementasi blockchain sebagai *package* nodeJS

Package ini dipublikasikan di ekosistem npm dengan nama *percobaan-simple-blockchain*. Untuk menggunakan *package* ini, dapat dilakukan dengan langsung saja menginstalnya pada root folder proyek aplikasi dengan mengetikkan pada *command line*:

npm install percobaan-simple-blockchain

Setelah *pacakge* diinstal, Kemudian tambahkan file sha256.min.js di dalam folder modul package yang baru diinstal. File sha256.min.js adalah file javascript eksternal yang dapat diperoleh di <https://cdnjs.cloudflare.com/ajax/libs/js-sha256/0.9.0/sha256.min.js>

Saya sengaja tidak memasukkan sebagai bagian dari package secara default karena ini merujuk ke kode milik orang lain yang resmi dan karena merujuknya secara langsung di dalam *package* percobaan-simple-blockchain juga bukanlah langkah yang aman dan bijak, sebab orang bisa saja merubah fungsi SHA256 di dalam pustaka tersebut menjadi berbahaya dan kita tetap merujuknya untuk dieksekusi pada proyek blockchain kita. Sehingga koneksi langsung kode sumber dari luar secara *live* harus kita putuskan.

Ini berbeda kasusnya jika file pustaka javascript di rujuk di sisi *client* (file html yang dieksekusi *browser*). Dalam konteks ini, merujuknya relatif lebih aman karena tidak dieksekusi di sisi server.

Akan tetapi, pada dasarnya nodeJS yang baru juga memiliki pustaka bawaan yang bernama “*crypto*”. Sha256.min.js dapat digantikan menggunakan pustaka ini secara langsung tanpa harus instal lagi.

3.11. Pembuatan dompet blockchain (*wallet*) di atas blockchain yang telah dibuat

Setelah infrastruktur blockchain telah kita buat, maka selanjutnya adalah membangun aplikasi antarmuka user yang menggunakan blockchain. Aplikasi pengguna blockchain ini biasanya disebut sebagai *wallet*.

Pembuatan *wallet* dapat dilihat dari dua sisi. Pertama pada sisi server yang merupakan master dari blockchain atau penyelenggara layanan blockchain atau juga dari *node* yang berwenang membuat *wallet*. Kedua pada sisi *client* yang memegang antarmuka aplikasi *wallet*.

Pada sisi server, pada dasarnya ketika seorang user mendaftar untuk menggunakan blockchain maka user mendapat sepasang kunci (*public key* dan *private key*), alamat wallet (biasanya adalah *public key* si user atau bisa juga konkatenasi string nomor registrasi dan *public key*), akun semacam inbox untuk menerima pesan atau mengirim pesan.

Pada sisi client, pada dasarnya hanyalah algoritma untuk menandatangani transaksi, mengirim transaksi atau menerima transaksi. Akan tetapi, ini menjadi lebih kompleks jika disesuaikan konteks penggunaan *blockchain*.

Karena itu, dari keduanya, sisi server dan sisi client, yang paling mendasar untuk dibuat kode sumbernya adalah:

- Kode untuk membuat pasangan kunci *private key* dan *public key* untuk diberikan kepada user yang baru mendaftar dan membuat *wallet*. (sisi server)
- Kode untuk membuat transaksi (sisi client)
- Kode untuk menandatangani transaksi (sisi client)
- Kode untuk memverifikasi tanda tangan di transaksi (sisi client dan server)

Berikut ini adalah contoh kode sederhana untuk membuat aplikasi *wallet* bagi user yang ingin menggunakan blockchain yang telah kita buat. Semua kode untuk *wallet* dibuat oleh hasezoey (github), penulis hanya menambahkan penjelasan tentang cara kerja kode-kode tersebut.

- Kode untuk membuat pasangan kunci (kode ini diambil dari <https://github.com/Savjee/SavjeeCoin/tree/master/src>)

```
| const EC = require('elliptic').ec;
```

```
/* Algoritma enkripsi asimetrik menggunakan elliptic-curve cryptography (ECC) yang diimplementasikan sebagai modul npm Elliptic
*/
const ec = new EC('secp256k1');

/*membuat pasangan kunci lalu menerjemahkan masing-masing ke bentuk hexa
*/
const key = ec.genKeyPair();
const publicKey = key.getPublic('hex');
const privateKey = key.getPrivate('hex');

// Print the keys to the console
console.log();
console.log('Your public key (also your wallet address, freely shareable)\n', publicKey);

console.log();
console.log('Your private key (keep this secret! To sign transactions)\n', privateKey);
```

- Kode untuk membuat transaksi

Berikut ini adalah kelas transaksi yang berisi struktur data transaksi, dan fungsi utama untuk melakukan *hashing* dan tanda tangan transaksi

```
class Transaction {
    /**
     * @param {string} fromAddress
     * @param {string} toAddress
     * @param {number} amount
     */

    constructor(fromAddress, toAddress, amount) {
        //Bentuk struktur data transaksi
        this.fromAddress = fromAddress;
        this.toAddress = toAddress;
        this.amount = amount;
        this.timestamp = Date.now();
    }
}
```

```
/**  
 * Creates a SHA256 hash of the transaction  
 *  
 * @returns {string}  
 */  
  
//Algoritma untuk melakukan hash terhadap transaksi  
calculateHash() {  
    return crypto.createHash('sha256').update(  
        this.fromAddress + this.toAddress + this.amount +  
        this.timestamp).digest('hex');  
}  
  
/**  
 * Signs a transaction with the given signingKey (   
 * which is an Elliptic keypair  
 * object that contains a private key). The signature  
 * is then stored inside the  
 * transaction object and later stored on the  
 * blockchain.  
 *  
 * @param {string} signingKey  
 */  
  
//signingKey adalah var signingKey = ec.genKeyPair()  
//dimana ec berasal dari  
//const ec = new EC('secp256k1') berasal dari  
//const EC = require('elliptic').ec  
//elliptic adalah modul npm yang diinstal di aplikasi  
//penjelasan rinci untuk ini di contoh cara pakainya  
//di https://www.npmjs.com/package/elliptic/v/6.5.2  
  
signTransaction(signingKey) {  
    // You can only send a transaction from the wallet that  
    // is linked to your  
    // key. So here we check if the fromAddress matches  
    // your publicKey  
    if (signingKey.getPublic('hex') !== this.fromAddress)  
        {throw new Error('You cannot sign transactions for  
        other wallets!');  
    }  
  
    //Bagian ini adalah bagian penandatanganan transaksi  
    //Pengirim menandatngani dengan cara mengenkripsi  
    //hash dari transaksi menggunakan private key yang  
    //dia miliki  
    const hashTx = this.calculateHash();
```

```
    const sig = signingKey.sign(hashTx, 'base64');

    this.signature = sig.toDER('hex');
}

/**
 * Checks if the signature is valid (transaction has
 * not been tampered with).
 * It uses the fromAddress as the public key.
 *
 * @returns {boolean}
 */

//Kode untuk memeriksa validitas transaksi
isValid() {
/*If the transaction doesn't have a from address we
assume it's amining reward and that it's valid. You
could verify this in adifferent way (special field f
or instance)
*/
}

//Cek dulu apakah alamat pengirim kosong?
if (this.fromAddress === null) return true;
if (!this.signature || this.signature.length === 0)
{throw new Error('No signature in this transaction')
; }

//Ambil public key dari alamat wallet (fromAddress)
//lalu ubah ke format hexa
const publicKey = ec.keyFromPublic(this.fromAddress,
'hex');

//Kemudian verifikasi hash transaksi sebelum
//ditandatangani (yaitu this.calculateHash())
//terhadap hash transaksi sesudah ditandatangani
//(yaitu this.signature)
//Verifikasi dengan cara mendekrip tanda tangan
//pengirim(yaitu this.signature)menggunakan
//public key pengirim, lalu mengekstrak
//hash transaksi di dalamnya kemudian membandingkannya
//dengan hash transaksi sekarang
//(yaitu this.calculateHash())
return publicKey.verify(this.calculateHash(), this.s
ignature);
}
```

}

Berikut ini adalah potongan kode untuk menambahkan transaksi. Kode ini disimpan di sisi server. Kode ini dapat disimpan pada kelas Block atau kelas Blockchain.

```
/**  
 *Potongan kode ini menambahkan transaksi yang sudah  
 *dibuat ke antrian atau pool transaksi.  
 *@param {Transaction} transaction  
 */  
addTransaction(transaction) {  
  
    //cek terlebih dulu apakah alamat pengirim dan  
    //penerima transaksi ada atau tidak, jika tidak ada  
    //maka error  
    if (!transaction.fromAddress || !transaction.toAddress) {throw new Error('Transaction must include from  
and to address');}  
  
    //Verifikasi transaksi apakah tandatangan valid?  
    if (!transaction.isValid()) {  
        throw new Error('Cannot add invalid transaction to c  
hain');  
    }  
  
    //Cek apakah pengirim koin memiliki jumlah koin yang  
    //cukup?  
    if (transaction.amount <= 0) {  
        throw new Error('Transaction amount should be higher  
than 0');  
    }  
  
    // Making sure that the amount sent is not greater t  
    han existing balance  
    if (this.getBalanceOfAddress(transaction.fromAddress)  
    ) < transaction.amount) {  
        throw new Error('Not enough balance');  
    }  
  
    //Tambahkan transaksi ke pool atau antrian transaksi  
    this.pendingTransactions.push(transaction);  
    debug('transaction added: %s', transaction);
```

```
}

//Validasi transaksi sisi server
/**
 * Validates all the transactions inside this block (signature + hash) and
 * returns true if everything checks out. False if the block is invalid.
 *
 * @returns {boolean}
 */
hasValidTransactions() {
    for (const tx of this.transactions) {
        if (!tx.isValid()) {
            return false;
        }
    }

    return true;
}

//Tambahkan transaksi yang terkirim dari client ke
//pool transaksi, antrian.
//Juga mengirim reward berupa koin transaksi ke
//penambang(miner) yang berhasil menambang
/**
 * Takes all the pending transactions, puts them in a
 * Block and starts the mining process.
 * It also adds a transaction to send the mining
 * reward to the given address.
 *
 * @param {string} miningRewardAddress
 */
minePendingTransactions(miningRewardAddress) {
    const rewardTx = new Transaction(null, miningRewardAddress, this.miningReward);
    this.pendingTransactions.push(rewardTx);

    const block = new Block(Date.now(), this.pendingTransactions, this.getLatestBlock().hash);
    //Lakukan penambangan untuk menambahkan block
    block.mineBlock(this.difficulty);

    debug('Block successfully mined!'); //menulis log.
    //Push blok baru ke blockchain yang berhasil ditambang
    this.chain.push(block);
}
```

```
this.pendingTransactions = [];
}

/** 
 * Returns the balance of a given wallet address.
 *
 * @param {string} address
 * @returns {number} The balance of the wallet
 */
getBalanceOfAddress(address) {
let balance = 0;
for (const block of this.chain) { //untuk setiap blok di blockchain
for (const trans of block.transactions) { //untuk setiap transaksi di blok
if (trans.fromAddress === address) { //cek apakah alamat fromAddress transaksi = alamat yang hendak di cek balance nya (address)

//balance = balance - jumlah_koin_yang_hendak_dikirim_di_transaksi
balance -= trans.amount;
}

if (trans.toAddress === address) {
balance += trans.amount; //balance = balance - jumlah_koin_yang_hendak_diterima_di_transaksi
}
}
}

debug('getBalanceOfAdrees: %s', balance); //menulis log?
return balance; //nilai koin yang tersisa
}

/** 
 * Returns a list of all transactions that happened to and from the given wallet address.
 *
 * @param {string} address
 * @return {Transaction[]}
 */
getAllTransactionsForWallet(address) {
const txs = [];
}
```

```
for (const block of this.chain) {
    for (const tx of block.transactions) {
        if (tx.fromAddress === address || tx.toAddresses === address) {
            txs.push(tx);
        }
    }
}

debug('get transactions for wallet count: %s', txs.length);
return txs;
}
```

Demikianlah demonstrasi pembuatan blockchain beserta *wallet*. Kode di atas dapat diinstal di komputer dengan mengetik:

```
npm install --save savjeeecoin
```

Kode sumber di atas adalah milik Xavier Decuyper dalam proyek blockchain yang dibuatnya yang dapat dipelajari dan dilihat pada <https://github.com/Savjee/SavjeeCoin>. Proyek ini berlisensi MIT, yaitu kita dapat memodifikasi dan menambahkan di dalam pustaka ini dengan bebas juga menggunakannya. Pustaka ini terletak pada folder “scr” dan halaman contoh kode javascript untuk menggunakannya dapat dilihat pada folder “test”.

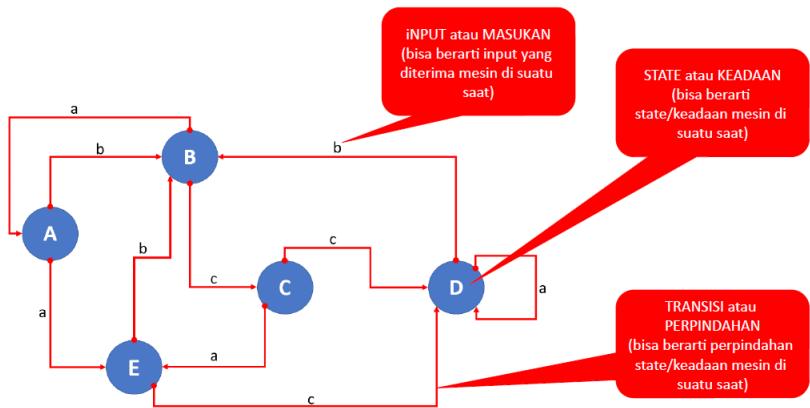
BAB 4

Filosofi Blockchain Ethereum

A. Blockchain sebagai automata (transaction-based state machine)

Sebagai sebuah arsitektur yang berbeda dengan arsitektur blockchain bitcoin, Ethereum adalah sebuah arsitektur blockchain sendiri. Pandangan pertama Ethereum terhadap blockchain, adalah automata atau sebuah *transaction-based state machine*.

Sebagai sebuah automata, blockchain keseluruhan adalah sebuah automata atau sebuah mesin transisi dimana transisinya dipicu oleh transaksi. Gambar 4.1 memberikan contoh sebuah automata atau mesin transisi. Akan tetapi pada contoh gambar 4.1, transisi dari satu state ke state yang lain dipicu oleh karakter input berupa a, b dan c.



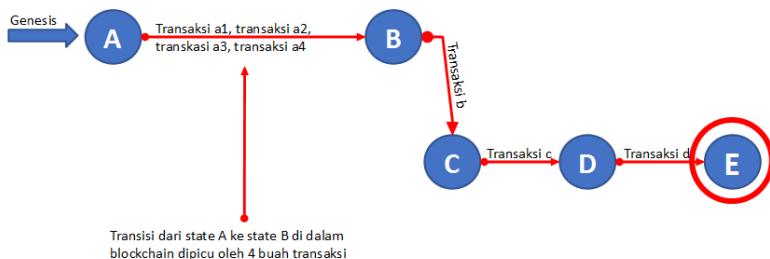
Gambar 4.1 Contoh sebuah automata

Gambar 4.1 hanyalah memberi contoh gambaran automata secara umum. Akan tetapi jika kita menggambarkan blockchain sebagai sebuah automata, maka blockchain yang berupa automata itu dapat diilustrasikan sebagaimana gambar 4.2. Gambar 4.2 menunjukkan sebuah automata dengan state A adalah state awal yang di dalam blockchain dinyatakan sebagai genesis.



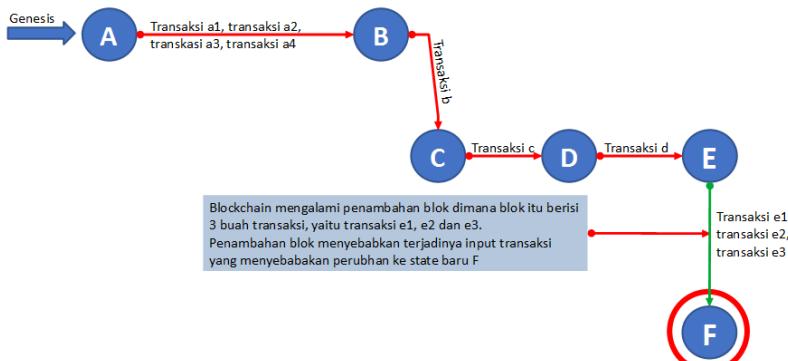
Gambar 4.2 Blockchain sebagai automata

Transisi di dalam blockchain mulai dari state A ke state berikut lalu state berikutnya lagi dipicu oleh input yang berupa transaksi atau sekumpulan transaksi (dapat dipicu oleh lebih dari 1 transaksi). Gambar 4.3 mengilustrasikan bagaimana jika perpindahan/transisi state dari state A ke state B dipicu oleh input 4 buah transaksi.



Gambar 4.3 Contoh transisi state yang dipicu oleh 4 buah transaksi

State E diberi lingkaran merah karena dia adalah state terakhir dari blockchain. Jika blockchain menambah blok maka state terakhir berubah sehingga bukan lagi E tetapi bertambah satu state baru di dalam blockchain. Gambar 4.4 mengilustrasikan bagaimana automata blockchain merubah statenya ke state baru manakala terjadi penambahan blok



Gambar 4.4 Penambahan blok menyebabkan perubahan state blockchain ke state baru yaitu F

Di dalam ekspresi bentuk instruksi untuk automata atau mesin Transisi state, gambar 4.4 dapat dinyatakan ke dalam serangkaian fungsi transisi yang lazim digunakan di dalam studi tentang automata. Pernyataan blockchain pada gambar 4.4 ke dalam ekspresi fungsi transisi dapat dinyatakan sebagai berikut:

$$(1) \quad S_{n+1} = \Upsilon (S_n, T)$$

Dimana S_n , S_{n+1} adalah state-state dari blockchain, dalam hal ini adalah A, B, C, D, E dan F.

Υ adalah fungsi transisi untuk mesin transisi blockchain.

T adalah transaksi-transaksi yang ditambahkan ke blok dari blockchain.

Jika dirinci yaitu sebagai berikut:

$$(2) \quad B = \Upsilon (A, (\text{transaksi } a_1, \text{transaksi } a_2, \text{transaksi } a_3, \text{transaksi } a_4))$$

$$(3) \quad C = \Upsilon (B, \text{transaksi } b)$$

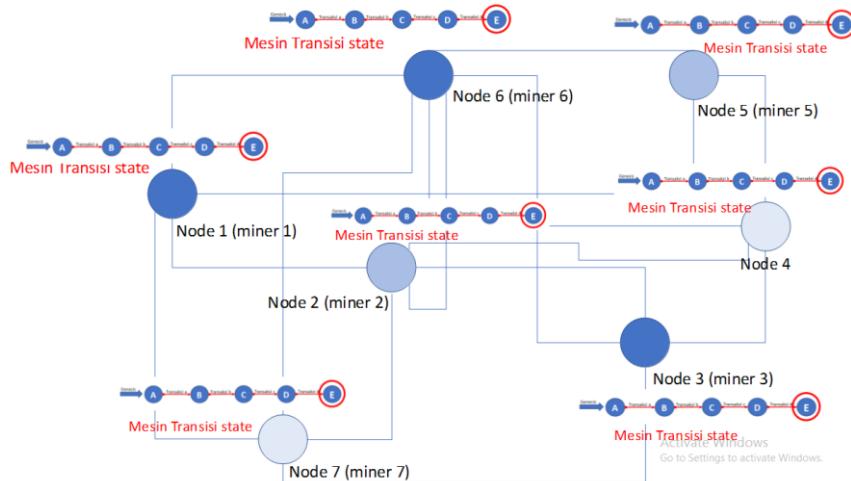
$$(4) \quad D = \Upsilon (C, \text{transaksi } c)$$

$$(5) \quad E = \Upsilon (D, \text{transaksi } d)$$

$$(6) \quad F = \Upsilon (E, (\text{transaksi } e_1, \text{transaksi } e_2, \text{transaksi } e_3))$$

State yang dimaksud di dalam blockchain ini adalah *world state*. Setiap saat terjadi penambahan blockchain maka Ethereum akan merubah *world state*. *World state* adalah sebuah data yang disimpan dalam bentuk *key-value* dan dia tidak tersimpan sebagai suatu data di dalam sebuah blok dari blockchain. *Key-value* *wolrd state* dari blockchain tersimpan di dalam basisdata node atau *client Ethereum*. Inshaa Allah sebuah penjelasan yang rinci tentang bagaimana *world state* di konstruksikan atau dibuat di dalam blockchain.

Gambar 4.5 memberikan ilustrasi bagaimana jika blockchain pada gambar di atas direplikasi atau di salin ke keseluruhan node dalam jaringan blockchain Ethereum. Pada dasarnya mereka menyalin juga seluruh state dari blockchain, sehingga pada dasarnya mereka juga memelihara mesin transisi atau automata seperti pada gambar 4.2.

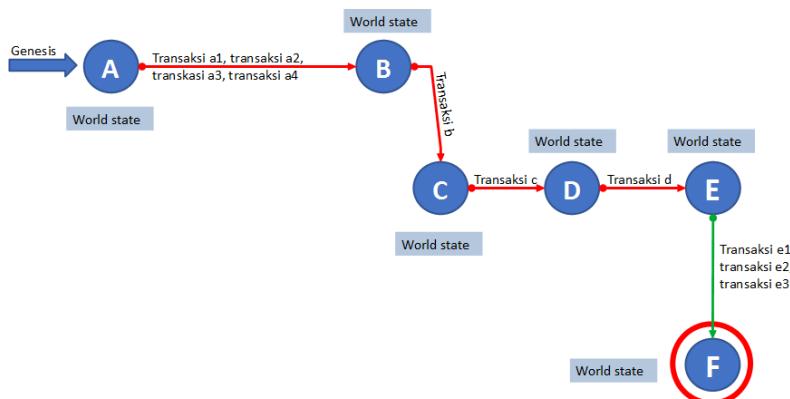


Gambar 4.5 Gambar keseluruhan blockchain Ethereum yang memelihara mesin transisi yang sama

1. World state di dalam blockchain Ethereum

Seperi dijelaskan sebelumnya, bahwa pada dasarnya blockchain dalam ethereum dapat dilihat sebagai mesin transisi yang berbasis transaksi (Wood, no date), yaitu mesin atau automata yang transaksinya dipicu oleh terjadinya transaksi.

State yang dimaksud pada automata blockchain pada dasarnya adalah isi dari *world state*. Jadi jika digambarkan adalah seperti gambar 4.6 di bawah.



Gambar 4.6 Perubahan world state

2. Anatomi blok di dalam blockchain Ethereum

Pada dasarnya blok di dalam Ethereum adalah mirip dengan anatomi blok di dalam blockchain bitcoin. Akan tetapi kita dapat membagi dua secara garis besar anatomi sebuah blok di dalam Ethereum. Pertama blok memuat *header* blok, kedua memuat data transaksi asli (tidak di hash) sebagai *payload*.

B. Konstruksi Blockchain menggunakan Azure Blockchain Workbench

Azure Blockchain Workbench (ABW), terkait dengan nama Azure, sebagai layanan cloud di microsoft, ABW hanyalah sebuah sub bagian dari seluruh layanan Azure. Akan tetapi AWB berdiri sendiri membentuk sebuah layanan blockchain. Yaitu layanan untuk membangun sebuah infrastruktur blockchain bagi perusahaan. ABW dapat dilihat sebagai kumpulan *service* di *cloud* yang membentuk kerangka atau *framework* untuk membangun blockchain. Ada beberapa layanan lain di *cloud* Azure. Seperti misalnya layanan untuk membangun infarstruktur bigdata di *cloud*.

Arsitektur blockchain ABW mengikuti arsitektur Ethereum. Sehingga kita mengenal *smart contract* di dalamnya. Secara garis besar, ABW memberikan 2 layanan blockchain, pertama membangun *business logic blockchain* dan kedua adalah membangun *smart contract*.



Glossary

- Blockchain : File penyimpanan yang terdistribusi terdesentralisasi dalam sejumlah komputer node user di internet. File terdiri dari rantai blok yang saling bertaut hash dan setiap bloknya menyimpan sejumlah transaksi yang terenkripsi atau tidak.
- Smart contract : Sebuah skrip kode atau untai kode yang tersimpan di dalam blok dari *blockchain* sebagai sebuah catatan transaksi atau sejumlah transaksi. Untai kode ini dibaca dan dieksekusi oleh sebuah mesin virtual di komputer node.
- Chain code : Serupa *smart contract*, hanya saja istilah *chain code* digunakan pada ekosistem blockchain *hyperledger*, sedang *smart contract* digunakan pada ekosistem blockchain Ethereum.
- Block* : Istilah abstrak untuk menyatakan rekord di dalam file *blockchain*.
- Transaksi : Istilah abstrak dan umum untuk menyatakan catatan di dalam blok. Transaksi bisa dilihat sebagai rekord juga. Tetapi rekord yang tercatat di dalam rekord (*block*).





Indeks



Daftar Pustaka

- Haq, I. and Esuka, O. M. (2018) ‘Blockchain Technology in Pharmaceutical Industry to Prevent Counterfeit Drugs Blockchain Technology in Pharmaceutical Industry to Prevent Counterfeit Drugs’, (March). doi: 10.5120/ijca2018916579.
- Nakamoto, S. (no date) ‘Bitcoin : A Peer-to-Peer Electronic Cash System’, pp. 1–9. Available at: www.bitcoin.org.
- Peyrott, S. E. (2017) *An Introduction to Ethereum and Smart Contracts*, Auth0. Auth0.
- Wood, D. G. (no date) ‘Ethereum: a secure decentralised generalised transaction ledger’.



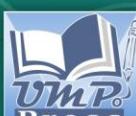
BLOCKCHAIN

FILOSOFI DAN KONSTRUKSI

Segala puji bagi Allah dan terima kasih kepada Nya, Tuhan Semesta Alam, yang telah memberikan kemampuan dan pengetahuan kepada kami untuk menyusun modul ini, serta Shalawat dan Salam kepada Rasulullah Nabi Muhammad Shallallahuaihi Wasallam.

Modul ini kami buat sebagai bentuk dari semangat kami untuk menyumbang bagi kemajuan dan kejayaan bangsa agar dapat menjadi sebuah bangsa yang berdiri sama tinggi dan sejajar dengan bangsa-bangsa lain, dapat menjadi pemimpin bagi kemajuan peradaban dan meretas jalan kepada kerjasama dan kolaborasi yang membawa kita kepada lapis demi lapis ketinggian ilmu pengetahuan dan teknologi yang universal yang pada gilirannya diharapkan dapat membawa kepada kesejahteraan dan kemakmuran bersama. Baik bersama sebagai sebuah bangsa dan bersama sebagai kelangsungan species manusia di bumi.

Buku ini adalah salah satu karya dan insyaa Allah secara konsisten akan disusul dengan buku-buku berikutnya. Pokok bahasan buku yang ditulis semata-mata untuk berbagi ilmu pengetahuan. Penulis berharap, buku ini dapat menginspirasi pembaca agar dapat lebih mudah dalam mempelajari dan menambah referensi terkait *BLOCKCHAIN* Filosofi dan Konstruksi.



Diterbitkan Oleh :
Unmuh Ponorogo Press
Anggota IKAPI, Anggota APPTI
Jalan Budi Utomo 10 Ponorogo 63471
Telp. (0852 9825 4709)
Email : unmuhpress@umpo.ac.id / umpopress@gmail.com

ISBN 978-602-0791-37-1



umpopress.umpo.ac.id



Unmuh Ponorogo Press



Unmuh_Ponorogo_Press



@umpopress