Implementasi dan Perbandingan Blockchain dengan Algoritma Hash
Keccak dan BLAKE2

Fadhil Imam Kurnia 13515146
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13515146@std.stei.itb.ac.id

***Abstrak* – Teknologi *blockchain* semakin berkembang dan dapat digunakan dalam berbagai bidang kehidupan manusia. Salah satu komponen penting dalam *blockchain* adalah fungsi *hash* yang berguna untuk menjaga konsistensi data dalam *blockchain*, dan memastikan tidak ada perubahan data yang terjadi. Penggunaan *blockchain* oleh banyak orang sekaligus menuntut penggunaan fungsi *hash* dengan performa yang handal. Fungsi *hash* Keccak dan BLAKE2 merupakan fungsi *hash* yang relatif baru dan dapat digunakan dalam teknologi *blockchain*. Dalam makalh ini akan dilakukan perbandingan penggunaan Keccak dan BLAKE2 pada Blockchain sederhana. Fokus utama perbandingan dilakukan dengan mengukur waktu dan penggunaan memori untuk menambahkan blok baru ke *blockchain*.**

Kata kunci – blockchain, hash, Keccak, BLAKE2

I. PENDAHULUAN

Teknologi Blockchain sebenarnya sudah lama dikemukakan, kemudian pada tahun 2008 muncul makalah tentang mata uang digital Bitcoin yang memanfaatkan teknologi Blockchain didalamnya. Semenjak munculnya mata uang digital tersebut, penggunaan Blockchain menjadi semakin banyak. Mata uang digital lainnya yang memanfaatkan Blockchain diantaranya adalah Ethereum, Litecoin, dan XXX. Selain itu Blockchain juga coba diimplementasikan untuk P2P Cloud Storage Network[1], P2P File System[2], hingga database[3]. Teknologi Blockchain juga dapat diaplikasikan untuk kebutuhan lain di berbagai bidang, misalnya untuk pencatatan kepemilikan dokumen, penyimpanan catatan medis, hingga pemilu online.

Algoritma hash digunakan dalam Blockchain untuk memastikan data yang dituliskan valid, sehingga perubahan data secara sepihak akan sulit untuk dilakukan. Terutama jika Blockchain diimplementasikan pada sistem terdistribusi dengan konsensus tertentu. Setiap perubahan data pada suatu blok akan mengakibatkan hash yang dihasilkan berbeda. Sehingga blok-blok selanjutnya menjadi tidak valid. Oleh karena itu, untuk melakukan perubahan secara sepihak dibutuhkan waktu komputasi yang lama.

Pemanfaatan Blockchain dilakukan sebelum adanya fungsi hash baru yang lebih aman. Oleh karena itu beberapa teknologi yang memanfaatkan Blockchain diketahui masih menggunakan fungsi hash yang lama. Misalkan saja Bitcoin yang masih menggunakan SHA-2. Untuk saat ini algoritma hash tersebut masih dapat mendukung berjalannya beragam teknologi yang menggunakan Blockchain. Namun kita juga perlu mempertimbangkan aspek keamanan dan skalabilitas pada waktu yang akan datang.

Padahal sekarang ini mulai banyak algoritma hash baru yang dikembangkan. Beberapa algoritma hash baru yang cukup populer diantaranya seperti KECCAK dan BLAKE2. Kedua algoritma tersebut merupakan finalis dalam penentuan SHA-3 yang merupakan standar terbaru untuk algoritma hash. Kita dapat mempertimbangkan penggunaan algoritma hash tersebut dalam teknologi Blockchain.

Pada makalah ini, akan dilakukan perbandingan penggunaan algoritma hash KECCAK dan BLAKE2 pada Blockchain sederhana. Perbandingan dilakukan dengan menghitung waktu yang dibutuhkan untuk penambahan blok baru, waktu yang dibutuhkan untuk melakukan proof-of-work (POW), serta efisiensi penggunaan memori.

II. DASAR TEORI

A. Fungsi *Hash*

Fungsi *hash* adalah fungsi yang menerima masukan sembarang *string* dengan panjang bebas dan menghasilkan sebuah *string* dengan panjang tetap. Biasanya string yang dihasilkan memiliki ukuran yang jauh lebih kecil dibanding string masukannya. Hasil keluaran fungsi hash disebut dengan *message digest* atau *hash value*. Fungsi hash bersifat searah karena *message digest* tidak dapat diubah menjadi string pembentuknya.

Dalam kriptografi, biasanya fungsi *hash* digunakan untuk memastikan integritas dari suatu data. Sedikit perubahan pada

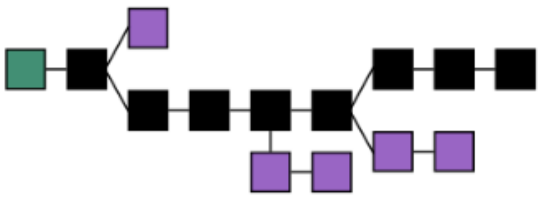
string masukan dalam fungsi *hash* akan mengakibatkan hasil yang jauha berbeda. Beberapa fungsi hash yang sudah cukup lama diantaranya seperti MD4, MD5, SHA-1, dan SHA-2.

Salah satu sifat yang ingin dicapai dalam sebuah fungsi hash H adalah untuk setiap string masukan x, tidak mungkin ada string y, $x \neq y$ sedemikian sehingga $H(x) = H(y)$. Namun, dalam kenyataannya pada beberapa algoritma fungsi hash terdapat kolisi, yaitu keadaan dimana dua buah string sembarang memiliki nilai hash yang sama.

Pada tahun 2007, *National Institute of Standards and Technology* (NIST) di Amerika Serikat membuat kompetisi untuk mengembangkan algoritma hash baru. Fungsi hash yang memenangkan kompetisi tersebut akan menjadi standar fungsi *hash* baru dengan nama SHA-3. Beberapa fungsi hash yang menjadi finalis dalam kompetisi tersebut diantaranya BLAKE, Grøstl, JH, Keccak, dan Skein. Setelah melalui seleksi panjang, pada tahun 2012 NIST mengumumkan Keccak sebagai pemenang kompetisi tersebut. Akhirnya Keccak menjadi acuan untuk pembuatan fungsi *hash* SHA-3.

C. *Blockchain*

Blockchain merupakan list kontigu yang terdiri dari blok-blok yang saling berkaitan. List pada blockchain tersebut dapat terus bertambah hingga tak terhingga. Blok awal pada suatu blockchain disebut dengan blok genesis. Setiap blok memiliki nilai hash dari blok sebelumnya, waktu pembuatan blok, dan data. Desain blockchain tersebut membuatnya resisten terhadap pengubahan data, karena setiap perubahan pada suatu blok akan membuat blok selanjutnya menjadi tidak valid. Untuk menambah keamanan, blockchain disimpan dalam jaringan terdistribusi dengan protokol peer-to-peer tertentu. Sehingga list blok tersebut terduplikasi di banyak server.



Gambar 1. formasi blockchain, list yang berwarna hitam merupakan list utama, list yang berwarna ungu merupakan list orphan, blok hijau merupakan blok genesis

Stuart Haber dan W. Scott Stornetta pada tahun 1991 pertama kali mendeskripsikan list blok yang menggunakan kriptografi untuk mengamankannya. Akhir-akhir ini teknologi blockchain mulai banyak digunakan untuk berbagai keperluan, terutama dalam bidang keuangan untuk membuat *cryptocurrency*. Penggunaannya sebagai distributed ledger dapat juga dimanfaatkan untuk mencatat event seperti medical record, managemen identitas, transaction processing, catatan kepemilikan, pelacakan bahan makanan, hingga pencatatan pemilu.

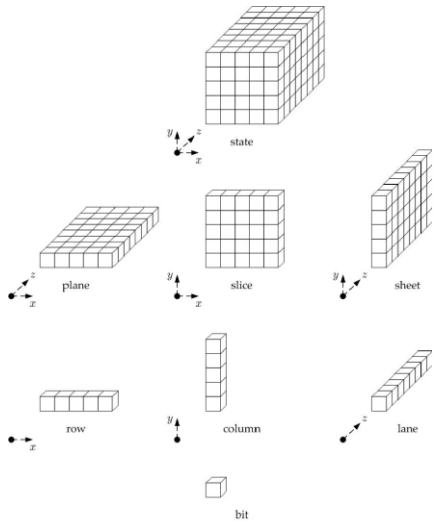
D. Fungsi *Hash* Keccak

Fungsi *hash* Keccak didesain oleh Guido Bertoni, Joan Daemen, Michaël Peeters, dan Gilles Van Assche. Berdasarkan klaim dari pendesainnya, Keccak mampu melakukan 12,5 cpb (*cycles per byte*) pada komputer dengan CPU Intel Core 2. Hal ini menunjukkan bahwa algoritma ini sangat cepat bahkan lebih cepat dibandingkan dengan finalis kompetisi NIST lainnya.

Keccak mengadopsi *sponge structure* (struktur spons). Pada struktur tersebut, input pada fungsi *hash* akan "diserap" ke dalam state *hash* dan diberi *rate* tertentu, lalu keluaran hash "diperas" keluar dengan *rate* yang sama. Untuk menyerap r bit data, data di XORkan ke bit yang memimpin state, lalu permutasi blok diaplikasikan. Untuk melakukan pemerasan, r bit pertama dari state dijadikan sebagai output, dan permutasi blok akan diaplikasikan lagi jika dibutuhkan. Permutasi blok pada Keccak sendiri dibagi menjadi empat tahap, yaitu tahap theta Θ, tahap rho ρ dan pi Π, tahap chi χ, dan tahap iota ι.

Pusat dari semuanya adalah "*capacity*" c dari fungsi Keccak, dengan c=1600-r yang tidak disentuh oleh masukan maupun keluaran data. Untuk menghitung nilai *hash* keccak, inisialisasi *state* menjadi 0, kemudian tambahkan pad pada masukan, dan pecah masukan data menjadi potongan r-bit. Serap input kedalam state, XOR-kan, kemudian aplikasikan ke permutasi blok.

Struktur pada Keccak adalah array tiga dimensi, jadi dalam melakukan perhitungan, tidak hanya didasarkan pada panjang dan lebar, namun juga lane, dan struktur seperti inilah yang membentuk state pada kecak.

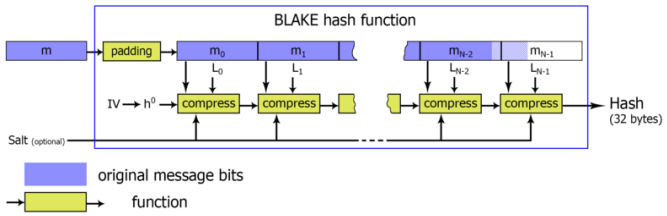


Gambar 2. Struktur State pada Keccak

E. Fungsi *Hash* BLAKE2

Pada bulan Desember 2012, setelah kompetisi yang diadakan oleh NIST berakhir, versi baru dari fungsi *hash* BLAKE diumumkan oleh Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, dan Christian Winnerlein. Versi terbaru dari fungsi *hash* tersebut dinamakan BLAKE2 dan dimaksudkan untuk menggantikan MD5 dan SHA-1 yang sudah ditemukan kolisinya. Secara garis besar fungsi *hash* BLAKE2 sama dengan fungsi *hash* BLAKE, beberapa perubahan yang dilakukan diantaranya:

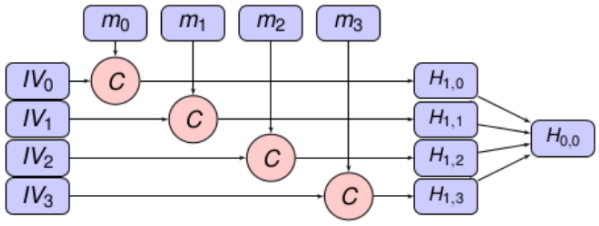
- Pengurangan jumlah putaran, dari 14 menjadi 10
- Merubah beberapa konstanta
- Mengakomodir mode paralel untuk meningkatkan performa.
- Mengakomodir mode *tree* untuk data yang besar.



Gambar 3. Diagram Blok Fungsi Hash BLAKE2

Dibandingkan dengan fungsi hash lainnya, BLAKE2 memiliki performa yang baik. Beberapa keunggulan yang dimiliki oleh BLAKE2 diantaranya:

- Performa yang lebih baik dibanding dengan MD5
- Penggunaan memori 32% lebih sedikit dibanding BLAKE
- Mendukung mode paralel dan *tree*.
- Padding yang minimal sehingga lebih mudah dan cepat diimplementasikan.



Gambar 3. Mode Paralel pada Fungsi *Hash* BLAKE2

III. PEMBAHASAN

Guna melakukan penelitian ini, terlebih dahulu dibuat *blockchain* sederhana yang dapat menggunakan fungsi *hash* yang berbeda-beda. Blockchain sederhana yang dibuat diberi nama FBlockchain dan dapat menggunakan fungsi *hash* Keccak dan BLAKE2. FBlockchain tersebut dibuat dengan bahasa Java 1.8 dan berbasis CLI. Tampilan dari program yang dibuat dapat dilihat pada gambar 4.

Gambar 4. Tampilan Program Blockchain Sederhana, FBlockchain

Fungsi *hash* Keccak memiliki keunggulan karena berbeda jauh dari fungsi *hash* lainnya jika ditinjau dari cara kerjanya. Keccak menggunakan struktur spons, sedangkan pada BLAKE2 strukturnya dikembangkan dari SHA-2 yang sudah ada. Oleh karena itu BLAKE2 sebenarnya lebih riskan silihat dari sisi keamanan.

Namun jika dilihat dari sisi performa, beberapa perbandingan yang telah dilakukan menunjukan bahwa fungsi *hash* BLAKE lebih cepat dibandingkan dengan Keccak. Pengujian menggunakan Intel Core i5-3210M (*Ivy Bridge*) menunjukan bahwa Keccak melakukan proses dengan 12.8 cpb, sedangkan BLAKE melakukan proses dengan 3.5 cpb. Fungsi *hash* BLAKE2 yang merupakan versi terbari dari BLAKE memiliki performa yang lebih baik lagi dibandingkan dengan fungsi *hash* BLAKE.

IV. ANALISIS

Dalam penelitian yang dilakukan ini akan diukur waktu yang dibutuhkan untuk menambahkan blok baru ke *blockchain* yang menggunakan fungsi hash Keccak dan BLAKE2. Selain itu juga akan dilakukan pengukuran memori yang dibutuhkan untuk melakukan penambahan blok ke *blockchain* tersebut. Proses pengukuran dilakukan dengan program FBlockchain yang sudah diimplementasikan terlebih dahulu. Spesifikasi komputer yang digunakan untuk melakukan pengujian adalah sebagai berikut:

- Processor: Intel(R) Core(TM) i3-2100
- CPU @3.10GHz 3.40GHz
- RAM: 4.00 GB 3.
- System Type: 64-bit Operating System
- Hard Disk: 1TB
- VGA: NVIDIA GeForce GT 430
- Operating System: Ubuntu 17.08

Untuk mengukur kecepatan dan penggunaan memori, terdapat beberapa standar yang perlu didefinisikan agar hasil yang diperoleh menjadi lebih konsisten dan objektif. Standar yang didefinisikan meliputi jenis fungsi hash yang digunakan, dan prosedur simulasi.

Fungsi hash Keccak dan BLAKE2 yang diimplementasikan dan digunakan untuk pengujian menerima sembarang masukan dan menghasilkan 256 bit keluaran. Pemilihan keluaran 256 bit dilakukan karena banyak penggunaan fungsi *hash* pada ukuran keluaran tersebut. Kedua fungsi hash tersebut juga diimplementasikan secara serial. Dengan jenis keluaran (*output*) yang sama proses membandingkan *blockchain* akan lebih konsisten.

Dalam melakukan setiap eksperimen menggunakan *blockchain* dengan Keccak dan *blockchain* dengan BLAKE2, prosedur yang digunakan selalu sama. Pengujian dilakukan dengan program FBlockchain yang sudah dibuat sebelumnya. Pertama akan dibuat blok genesis untuk ditambahkan ke *blockchain.* Kemudian akan dilakukan pengujian secara eksponensial dengan ukuran data sebesar 2 byte, 4 byte, 8 byte, dan seterusnya hingga 524288 byte. Pada proses penambahan blok baru, data acak akan dimasukan dalam blok kemudian proses hash dilakukan, lalu blok tersebut ditambahkan ke *blockchain* yang sudah ada.

Untuk melakukan pengukuran waktu penambahan blok baru, waktu sebelum dan sesudah penambahan akan dihitung selisihnya. Sedangkan untuk melakukan penghitungan penggunaan memori terlebih dahulu perlu dilakukan proses pembersihan memori dengan *garbage collector* di Java. Kemudian penggunaan memori sebelum dan setelah penambahan blok dapat ditentukan. Penambahan data dilakukan secara eksponensial untuk melihat performa dengan data kecil dan data besar. Setiap pengujian dengan ukuran data yang sama akan dilakukan sebanyak 5 kali, kemudian dihitung rata-rata dari waktu dan penggunaan memori yang dilakukan. Hasil pengukuran waktu dan penggunaan memori kemudian akan divisualisaikan dalam grafik kartesian agar dapat terlihat lebih jelas.

Hasil pengukuran waktu untuk penambahan blok dapat dilihat pada Tabel 1. Sedangkan pengukuran penggunaan memori dapat dilihat pada Tabel 2.

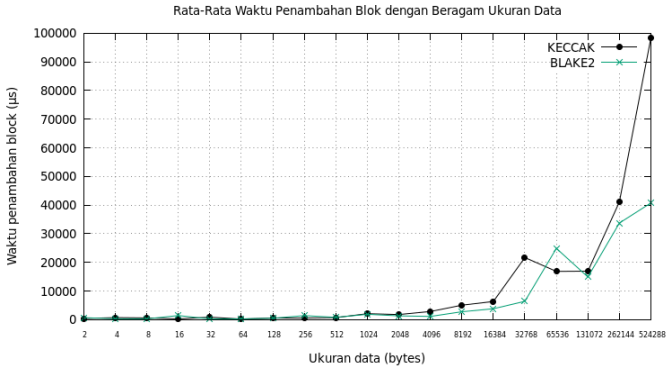
Tabel 1. Waktu Penambahan Blok pada Blockchain dengan Keccak dan BLAKE2

# Data (bytes)	Keccak (ns)	BLAKE2 (ns)
2	388094	688568
4	710103	282419
8	592387	266400
16	263018	1433441
32	864404	371050
64	197862	378713
128	427183	485044
256	419966	1396114
512	712480	808690
1024	2073941	1849594
2048	1770320	1323208
4096	2848091	1114356
8192	5046488	2749208
16384	6344481	3798892
32768	21677707	6397346
65536	16843765	24889450
131072	16925797	14902026
262144	41103687	33742640
524288	98326401	40757561

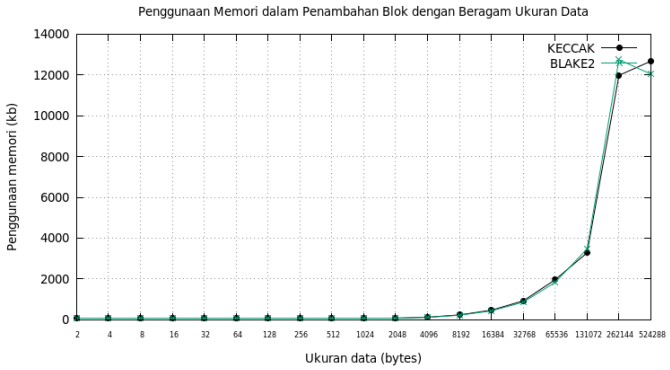
Tabel 2. Penggunaan Memori untuk Penambahan Blok pada Blockchain dengan Keccak dan BLAKE2

# Data (bytes)	Keccak (bytes)	BLAKE2 (bytes)
2	62936	62936
4	62936	62936
8	62936	62936
16	62936	62936
32	62936	62936
64	62936	62936
128	62936	62936
256	62936	62936
512	62936	62936
1024	62936	62936
2048	63008	62936
4096	12525	116036
8192	239944	223336
16384	469272	436344
32768	928048	862344
65536	1967128	1835872
131072	3279875	3474398
262144	11983752	12744785
524288	12676235	12044769

Visualisasi dari hasil percobaan dapat dilihat pada Gambar 5 dan gambar 6.



Gambar 5. Waktu Penambahan Blok dengan Beragam Ukuran Data pada Blockchain Keccak dan BLAKE2



Gambar 6. Penggunaan Memori untuk Penambahan Blok dengan Beragam Ukuran Data pada Blockchain Keccak dan BLAKE2

Dari grafik perbandingan kecepatan penambahan blok baru pada Gambar 5, dapat dilihat bahwa secara umum penggunaan fungsi *hash* BLAKE2 menghasilkan performa *blockchain* yang lebih baik dibandingkan dengan Keccak, terutama pada penambahan blok dengan data yang berukuran besar. Namun pada beberapa blok dengan ukuran tertentu, yaitu dengan ukuran 16 byte, 256 byte, 512 byte, dan 65536 byte penggunaan Keccak memerlukan waktu yang lebih cepat.

Kemudian dari grafik perbandingan penggunaan memori untuk penambahan blok baru pada Gambar 6, dapat dilihat bahwa penggunaan memori untuk kedua jenis *blockchain* tersebut dapat dikatakan relatif sama. Namun jika diperhatikan secara lebih detail, *blockchain* dengan fungsi *hash* BLAKE2 membutuhkan memori sedikit dibawah *blockchain* dengan fungsi *hash* Keccak.

VI. KESIMPULAN DAN SARAN

Fungsi *hash* Keccak dan BLAKE2 dapat dipertimbangkan untuk digunakan dalam *blockchain* baru yang akan dikembangkan. Kedua fungsi *hash* tersebut belum ditemukan kolusinya dan dikembangkan untuk menyempurnakan fungsi *hash* yang sudah lebih dahulu populer seperti MD5 dan SHA-2. Jika dibandingakan dari aspek kecepatan dan penggunaan memori untuk proses *hashing*, *blockchain* yang menggunakan fungsi *hash* BLAKE2 lebih baik jika dibandingkan dengan *blockchain* yang menggunakan Keccak.

Kedepannya dapat dilakukan penelitian lebih lanjut dengan fungsi *hash* baru lainnya, atau dengan mengaktifkan mode paralel atau mode *tree* pada beberapa fungsi hash yang mendukungnya.

UCAPAN TERIMA KASIH

Dengan selesainya penulisan makalah ini, penulis mengucapkan syukur atas rahmat yang telah diberikan oleh Tuhan Yang Maha Esa karena telah selesainya makalah dengan baik dan tepat waktu. Selanjutnya, penulis mengucapkan terima kasih kepada Bapak Rinaldi Munir sebagai dosen mata kuliah IF4020 Kriptografi yang telah memberikan pengetahuan-pengetahuan mengenai algoritma-algoritma kriptografi baik di dalam kelas maupun di luar kelas karena pengetahuan tersebut sangat berguna untuk menyelesaikan makalah ini. Selain itu, penulis juga ingin mengucapkan terima kasih kepada teman-teman yang sudah membantu penulis baik secara langsung maupun tidak langsung sehingga makalah ini dapat selesai dengan baik.

REFERENSI

[1] Storj A Peer-to-Peer Cloud Storage Network. Shawn W., Tome B., Josh B.,James P., Gordon H., Patrick G., Philip H., Chris P. '16
[2] IPFS - Content Addressed, Versioned, P2P File System. Benet J. '15
[3] BigchainDB: A Scalable Blockchain Database. McConaghy T, Marques R, Müller A, De Jonghe D, McConaghy T, McMullen G, Henderson R, Bellemare S, Granzotto A. '17
[4] Keccak Specifications. Guido Bertoni, Joan Daemen, Michael Peeters dan Gilles Van Assche1. ‘08
[5] BLAKE2: simpler, smaller, fast as MD5. Jean-Philippe Aumasson , Samuel Neves, Zooko Wilcox-O’Hearn, Christian Winnerlein. ‘13
[6] Munir, Rinaldi, Fungsi Hash, Program Studi Teknik Informatika.

PERNYATAAN

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

Bandung, 10 Mei 2018

Fadhil Imam Kurnia
13515146

LAMPIRAN

A. Hasil pengujian kecepatan blockchain dengan Keccak

```
Test with 2 bytes of payload
474036ns    369288ns    392249ns    335766ns    369133ns    = 388094ns
Test with 4 bytes of payload
682784ns    758112ns    742023ns    657128ns    710470ns    = 710103ns
Test with 8 bytes of payload
815756ns    606924ns    561463ns    524005ns    453789ns    = 592387ns
Test with 16 bytes of payload
318169ns    311511ns    308058ns    197270ns    180084ns    = 263018ns
Test with 32 bytes of payload
222771ns    174536ns    3520275ns   203531ns    200908ns    = 864404ns
Test with 64 bytes of payload
201741ns    185287ns    206296ns    205222ns    190766ns    = 197862ns
Test with 128 bytes of payload
282870ns    273303ns    966064ns    279267ns    334414ns    = 427183ns
Test with 256 bytes of payload
419583ns    459237ns    414943ns    415789ns    390279ns    = 419966ns
Test with 512 bytes of payload
683369ns    679087ns    714573ns    712172ns    773201ns    = 712480ns
Test with 1024 bytes of payload
1379197ns   1342676ns   1606278ns   4944560ns   1096994ns   = 2073941ns
Test with 2048 bytes of payload
2249556ns   2029559ns   1640186ns   1647832ns   1284470ns   = 1770320ns
Test with 4096 bytes of payload
2651116ns   2746490ns   4012849ns   2332046ns   2497954ns   = 2848091ns
Test with 8192 bytes of payload
6536457ns   4869534ns   7056965ns   3412610ns   3356877ns   = 5046488ns
Test with 16384 bytes of payload
6242108ns   6212332ns   6351570ns   6690407ns   6225990ns   = 6344481ns
Test with 32768 bytes of payload
11645756ns  11988495ns  22662380ns  32428910ns  29662994ns  = 21677707ns
Test with 65536 bytes of payload
30876206ns  16390749ns  15371827ns  12606009ns  8974036ns   = 16843765ns
Test with 131072 bytes of payload
16792482ns  14550408ns  12325682ns  20015297ns  20945118ns  = 16925797ns
Test with 262144 bytes of payload
32571810ns  58550406ns  39764367ns  46440726ns  28191126ns  = 41103687ns
Test with 524288 bytes of payload
98791933ns  94358877ns  100768415ns 96367121ns  101345660ns = 98326401ns
Test with 1048576 bytes of payload
129063284ns 159648601ns 187018018ns 229456178ns 160678077ns = 173172831ns
Validity: true
```

B. Hasil pengujian kecepatan blockchain dengan BLAKE2

```
Test with 2 bytes of payload
330634ns    402169ns    2180890ns   294809ns    234341ns    = 688568ns
Test with 4 bytes of payload
252234ns    311646ns    288245ns    311096ns    248877ns    = 282419ns
Test with 8 bytes of payload
247842ns    251499ns    258819ns    270984ns    302859ns    = 266400ns
```

```
Test with 16 bytes of payload
6101706ns  253933ns  238660ns  250663ns  322246ns  = 1433441ns
Test with 32 bytes of payload
535861ns  254537ns  262247ns  267267ns  535340ns  = 371050ns
Test with 64 bytes of payload
369732ns  370837ns  374371ns  410934ns  367693ns  = 378713ns
Test with 128 bytes of payload
465697ns  465572ns  501170ns  466877ns  525906ns  = 485044ns
Test with 256 bytes of payload
4573635ns  991292ns  569409ns  433670ns  412564ns  = 1396114ns
Test with 512 bytes of payload
740262ns  1001570ns  720008ns  755861ns  825753ns  = 808690ns
Test with 1024 bytes of payload
1657182ns  1910061ns  1904688ns  1885167ns  1890874ns  = 1849594ns
Test with 2048 bytes of payload
3623678ns  927615ns  982666ns  455134ns  626947ns  = 1323208ns
Test with 4096 bytes of payload
1088032ns  1130933ns  1087398ns  1121731ns  1143687ns  = 1114356ns
Test with 8192 bytes of payload
2182608ns  3795796ns  3345181ns  2241050ns  2181409ns  = 2749208ns
Test with 16384 bytes of payload
3796567ns  4370030ns  3595761ns  3617814ns  3614290ns  = 3798892ns
Test with 32768 bytes of payload
6814017ns  6887244ns  6787253ns  5874190ns  5624029ns  = 6397346ns
Test with 65536 bytes of payload
47912134ns  32277979ns  16723332ns  18969260ns  8564547ns  = 24889450ns
Test with 131072 bytes of payload
12941451ns  14421747ns  15888076ns  15752103ns  15506757ns  = 14902026ns
Test with 262144 bytes of payload
31239916ns  56164539ns  19239933ns  46277491ns  15791325ns  = 33742640ns
Test with 524288 bytes of payload
61830777ns  50496548ns  43007942ns  21972644ns  26479898ns  = 40757561ns
Test with 1048576 bytes of payload
56753551ns  77931147ns  73326710ns  63642528ns  79183549ns  = 70167497ns
Validity: true
```

C. Hasil pengujian penggunaan memori blockchain dengan Keccak

```
Test with 2 bytes of payload
62936b  62936b  62936b  62936b  62936b  = 62936b
Test with 4 bytes of payload
62936b  62936b  62936b  62936b  62936b  = 62936b
Test with 8 bytes of payload
62936b  62936b  62936b  62936b  62936b  = 62936b
Test with 16 bytes of payload
62936b  62936b  62936b  62936b  62936b  = 62936b
Test with 32 bytes of payload
62936b  62936b  62936b  62936b  62936b  = 62936b
Test with 64 bytes of payload
62936b  62936b  62936b  62936b  62936b  = 62936b
Test with 128 bytes of payload
62936b  62936b  62936b  62936b  62936b  = 62936b
Test with 256 bytes of payload
62936b  62936b  62936b  62936b  62936b  = 62936b
Test with 512 bytes of payload
62936b  62936b  62936b  62936b  62936b  = 62936b
Test with 1024 bytes of payload
```

```
62936b  62936b  62936b  62936b  62936b  = 62936b
Test with 2048 bytes of payload
62936b  62936b  62936b  62936b  63296b  = 63008b
Test with 4096 bytes of payload
125256b 125256b 125256b 125256b 125256b  = 125256b
Test with 8192 bytes of payload
239944b 239944b 239944b 239944b 239944b  = 239944b
Test with 16384 bytes of payload
469272b 469272b 469272b 469272b 469272b  = 469272b
Test with 32768 bytes of payload
928048b 928048b 928048b 928048b 928048b  = 928048b
Test with 65536 bytes of payload
1967128b    1967128b    1967128b    1967128b    1967128b    = 1967128b
Test with 131072 bytes of payload
1110248b    5505240b    5505240b    2139208b    2139440b    = 3279875b
Test with 262144 bytes of payload
13993064b   13692072b   13952168b   4591448b    13690008b   = 11983752b
Test with 524288 bytes of payload
19234072b   11745424b   11743360b   12060688b   8597632b    = 12676235b
Test with 1048576 bytes of payload
18187376b   23334128b   33032208b   23595024b   18354208b   = 23300588b
Validity: true
```

D. Hasil pengujian penggunaan memori blockchain dengan BLAKE2

```
est with 2 bytes of payload
62936b  62936b  62936b  62936b  62936b  = 62936b
Test with 4 bytes of payload
62936b  62936b  62936b  62936b  62936b  = 62936b
Test with 8 bytes of payload
62936b  62936b  62936b  62936b  62936b  = 62936b
Test with 16 bytes of payload
62936b  62936b  62936b  62936b  62936b  = 62936b
Test with 32 bytes of payload
62936b  62936b  62936b  62936b  62936b  = 62936b
Test with 64 bytes of payload
62936b  62936b  62936b  62936b  62936b  = 62936b
Test with 128 bytes of payload
62936b  62936b  62936b  62936b  62936b  = 62936b
Test with 256 bytes of payload
62936b  62936b  62936b  62936b  62936b  = 62936b
Test with 512 bytes of payload
62936b  62936b  62936b  62936b  62936b  = 62936b
Test with 1024 bytes of payload
62936b  62936b  62936b  62936b  62936b  = 62936b
Test with 2048 bytes of payload
62936b  62936b  62936b  62936b  62936b  = 62936b
Test with 4096 bytes of payload
112952b 116808b 116808b 116808b 116808b  = 116036b
Test with 8192 bytes of payload
223336b 223336b 223336b 223336b 223336b  = 223336b
Test with 16384 bytes of payload
436344b 436344b 436344b 436344b 436344b  = 436344b
Test with 32768 bytes of payload
862344b 862344b 862344b 862344b 862344b  = 862344b
Test with 65536 bytes of payload
1835872b    1835872b    1835872b    1835872b    1835872b    = 1835872b
```

Test with 131072 bytes of payload
3451600b 4981184b 4981184b 1856080b 2101944b = 3474398b
Test with 262144 bytes of payload
12621336b 12638192b 12900352b 12925856b 12638192b = 12744785b
Test with 524288 bytes of payload
9677896b 9952648b 9647936b 9748136b 21197232b = 12044769b
Test with 1048576 bytes of payload
29856656b 35977488b 23428664b 15208480b 15161440b = 23926545b
Validity: true