Analisis Perbandingan Algoritma SHA-256 dan Keccak-256 dalam Smart Contract EVM

Ida Bagus Rizky Brahmantya[a1], Gst. Ayu Vida Mastrika Giri[a2],

Program Studi Informatika, Fakultas Matematika dan Ilmu Pengetahuan Alam,
Universitas Udayana
Jalan Raya Kampus UNUD, Bukit Jimbaran, Kuta Selatan, Badung, Bali, Indonesia
[1]bagusrisky123@gmail.com
[2]vida@unud.ac.id

Abstract

The advancement of blockchain technology has increased the demand for efficient and secure hash algorithms, particularly in the development of smart contracts. This study uses smart contracts on the Ethereum network with inputs of 32, 128, and 1024 characters to evaluate SHA-256 and Keccak-256 based on execution cost, gas fees, and security. According to the results, Keccak-256 is more effective for smart contract computations because it consistently has lower execution costs for all input sizes. Both algorithms perform similarly for big inputs (1024 characters), suggesting comparable storage efficiency at scale, even if its gas charge is marginally greater than SHA-256 for tiny inputs (32 and 128 characters). Keccak-256 and SHA-256 both have strong defenses against brute-force assaults. Both provide similar security, while Keccak-256 takes a little longer to calculate. All things considered, Keccak-256 offers better efficiency, which qualifies it for widespread smart contract implementation. Further research is recommended to explore performance in more complex blockchain environments and execution gas and to optimize gas for practical implementation.

Keywords: *Blockchain, Smart Contract,* SHA-256, Keccak-256, *Hash Algorithm*

1. Pendahuluan

Perkembangan teknologi blockchain telah membuka jalan bagi berbagai inovasi di era digital, mulai dari *cryptocurrency* dan *smart contract. Smart contract* sendiri adalah program komputer yang berjalan otomatis di dalam sistem *blockchain* tanpa memerlukan intervensi pihak ketiga [1]. Salah satu komponen vital dalam *blockchain* dan *smart contract* adalah algoritma hash, yang bertanggung jawab atas integritas data, memvalidasi transaksi, dan memastikan keamanan jaringan. Diantara algoritma hash yang umum digunakan terdapat dua algoritma hash yang sering digunakan dalam implementasi blockchain adalah *SHA-256* dan *Keccak-256*.

SHA-256, yang dikembangkan oleh *National Security Agency (NSA)*, menjadi standar utama dalam *bitcoin* dan banyak sistem keamanan berbasi *blockchain* lainya. Sementara itu, *Keccak-256*, yang merupakan dasar dari *SHA-3* yang distandarisasi oleh *National Institute of Standards and Technology (NIST)*, banyak digunakan dalam ekosistem *ethereum* untuk berbagai aplikasi *smart contract.* Meskipun keduanya memiliki fungsi dasar yang serupa berupa output *hash* yang unik, terdapat perbedaan dari segi arsitektur desain, performa, dan tingkat keamanan. Perbedaan ini mempengaruhi aspek penting dalam pengembangan *smart contract*, salah satunya adalah biaya gas (*gas fee*) [2].

Setiap operasi dalam *smart contract* memerlukan sejumlah biaya eksekusi yang disebut *gas fee*. *Gas fee* merupakan satuan biaya yang dibayarkan pengguna kepada validator untuk memproses dan memvalidasi transaksi di blockchain[3]. Penggunaan gas dalam transaksi pada blockchain *ethereum* umumnya dipengaruhi oleh parameter yang digunakan saat menjalankan kode *solidity*, sehingga pemahaman akan hal ini dapat membantu pengembang merancang *smart contract* yang lebih efisien dalam konsumsi gas[4]. Oleh karena itu, efisiensi algoritma hash dalam mengolah data berkontribusi langsung terhadap total biaya yang harus dikeluarkan pengguna.

Selain itu, pemilihan algoritma juga mempengaruhi kecepatan eksekusi *smart contract* dan potensi keamanan jangka panjang.

Penelitian-penelitian sebelumnya mengenai algoritma hash dalam konteks blockchain umumnya berfokus pada komponen kinerja dasar, seperti kecepatan pemrosesan, konsumsi sumber daya, dan efisiensi throughput algoritma. Banyak penelitian, seperti yang dilakukan oleh Kuznetsov[5] dan Sevin & Mohammed[6], mempelajari metrik seperti *cycles per byte*, *megabytes per second*, dan laju hash (KHash/sec) di sistem blockchain umum. tetapi seringkali tidak membahas kinerja eksekusi *smart contract* atau konsumsi *gas fee* di *Ethereum Virtual Machine (EVM)*. Selain itu, komponen keamanan kriptografi biasanya hanya diuji secara teoritis tanpa menguji serangan gabungan atau ketahanan *preimage* secara langsung.

Penelitian ini membandingkan algoritma *SHA-256* dan *Keccak-256* dari segi *execution cost, gas fee,* dan keamanan terhadap *brute force* untuk menilai efisiensi dan ketahanannya dalam *smart contract* di jaringan *Ethereum*. *SHA-256* digunakan di Bitcoin, sedangkan *Keccak-256* merupakan standar di Ethereum, sehingga pemahaman mendalam terhadap keduanya penting untuk pengembangan aplikasi blockchain yang aman dan efisien. Penelitian ini bertujuan memberikan rekomendasi berbasis data kepada pengembang dalam memilih algoritma *hash* yang ideal berdasarkan efisiensi, keamanan, dan biaya, sehingga pemilihan tidak hanya berdasarkan popularitas, tetapi juga pertimbangan teknis yang objektif dan komprehensif.

2. Metode Penelitian

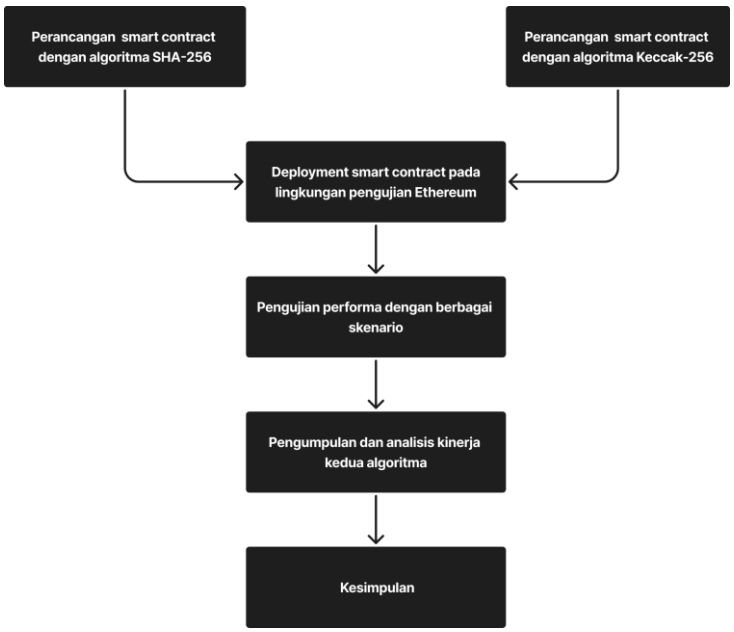
2.1. Data Penelitian

Penelitian ini menggunakan data teks dan integer untuk menguji kinerja algoritma *hash SHA-256* dan *Keccak-256* dalam *smart contract*. *Input* digunakan sebagai parameter utama dalam proses hashing pada setiap implementasi *smart contract*. Data yang diuji dikumpulkan secara buatan, atau disusun langsung oleh peneliti, sehingga memiliki kendali penuh atas variasi panjang dan kompleksitas *input*. Tiga kategori data yang digunakan terdiri atas *string* pendek (32 karakter), *string* sedang (128 karakter), dan *string* panjang (lebih dari 1000 karakter), masing-masing dibuat secara manual untuk mendukung proses pengujian yang teratur. Dibandingkan dengan dataset publik atau sumber eksternal, data yang digunakan dalam penelitian ini dibuat secara buatan untuk memenuhi kebutuhan pengujian. Hal ini memungkinkan kontrol penuh atas parameter pengujian dan memberikan fleksibilitas untuk secara adil dan konsisten membandingkan hasil antara dua algoritma.

Hasil hashing data input ini digunakan untuk mengukur berbagai komponen kinerja algoritma. Ini termasuk *execution cost, gas fee,* dan *keamanan algoritma*. Selanjutnya, data hasil pengujian dikumpulkan dan dianalisis untuk mengetahui kelebihan dan kekurangan setiap algoritma ketika digunakan pada jaringan *Ethereum*.

p-ISSN: 2986-3929
e-ISSN: 3032-1948

2.2. Tahapan Penelitian



Gambar 1. Tahapan Penelitian

Penelitian ini dilakukan dalam beberapa langkah sebagai berikut gambar 1, yang berisi tahapan yang digambarkan secara sistematis. Pertama adalah merancang *smart contract* dengan mengimplementasikan dua algoritma hash yang berbeda yaitu *SHA-256* dan *Keccak-256* secara terpisah. Pada tahap ini juga termasuk membuka kode *smart contract* dengan menggunakan bahasa *solidity* yang terintegrasi beserta masing-masing fungsi hash dari algoritma tersebut.

Setelah perancangan selesai, *smart contract* dari kedua algoritma di *deploy* ke jaringan *ethereum*. Tujuannya dari tahapan ini adalah untuk memastikan bahwa kedua *smart contract* dapat berfungsi dengan baik dan semestinya. Selanjutnya berbagai skenario diujikan. Untuk menjaga validitas perbandingan, pengujian dilakukan untuk kedua algoritma dalam kondisi yang seragam. Pada tahap pengumpulan, data dari hasil pengujian dikumpulkan dan dianalisis. Ini dilakukan dengan membandingkan performa *smart contract*.

Tahapan terakhir dari penelitian ini adalah menyusun kesimpulan berdasarkan hasil analisis. kesimpulan yang diperoleh akan berguna untuk menentukan algoritma mana yang lebih baik untuk digunakan untuk menerapkan *smart contract* pada *ethereum*.

2.3. Tahap Algoritma Hash

Tujuan dari bagian ini adalah untuk mengevaluasi dan membandingkan kinerja kedua algoritma hashing, yaitu *SHA-256* dan *Keccak-256*, saat mengembangkan *smart contract* dalam jaringan *Ethereum*. Tujuannya adalah untuk membahas efisiensi pelaksanaan kontrak, penggunaan gas, dan keamanan, yang berdampak pada kinerja sistem secara keseluruhan.

a. Implementasi Smart Contract dengan SHA-256

Pada bagian ini, smart contract dikembangkan menggunakan fungsi hashing bawaan Solidity, yaitu *SHA-256*, untuk mengenkripsi input berupa string atau bytes. Fokus implementasi ini adalah mengamati pengaruh perubahan data input terhadap konsumsi gas saat fungsi dijalankan. Pengujian dilakukan di *Remix IDE*, dengan pemanggilan fungsi *SHA-256* secara berulang menggunakan variasi input untuk memantau pola konsumsi sumber daya. Kemudian data tersebut digunakan untuk mengukur beban kinerja jaringan saat kontrak dieksekusi.

b. Implementasi Smart Contract dengan Keccak-256

Pada bagian ini, fungsi *Keccak-256* yang tersedia dalam *Solidity* digunakan untuk menerapkan algoritma hashing standar *Ethereum*, yang juga digunakan dalam berbagai komponen inti seperti penghitungan alamat kontrak dan validasi transaksi. Keunggulan utama *Keccak-256* adalah integrasinya yang erat dengan platform Ethereum. Untuk memastikan hasil yang dapat dibandingkan secara adil, pengujian dilakukan dengan input yang sama seperti pada pengujian *SHA-256*.

2.4. Matrik Evaluasi

Penelitian ini mengevaluasi kinerja dua algoritma hash, *SHA-256* dan *Keccak-256*, dalam *smart contract* di jaringan *Ethereum* dengan fokus pada tiga metrik utama: *execution cost, gas fee,* dan keamanan algoritma. *Execution cost* digunakan untuk mengukur efisiensi pemrosesan atau eksekusi fungsi hashing di dalam *EVM*, semenetara *has fee* digunakan untuk mengukur efisiensi biaya eksekusi fungsi hashing. Efisiensi dalam kedua aspek ini penting untuk menekan biaya operasional di jaringan publik seperti *Ethereum*. Selain itu, keamanan algoritma dianalisis berdasarkan ketahanannya terhadap serangan kriptografi, khususnya *brute force*. Dengan mengevaluasi aspek efisiensi dan keamanan secara menyeluruh, penelitian ini memberikan gambaran komprehensif mengenai keandalan kedua algoritma dalam menjaga integritas data dan efisiensi implementasi smart contract.

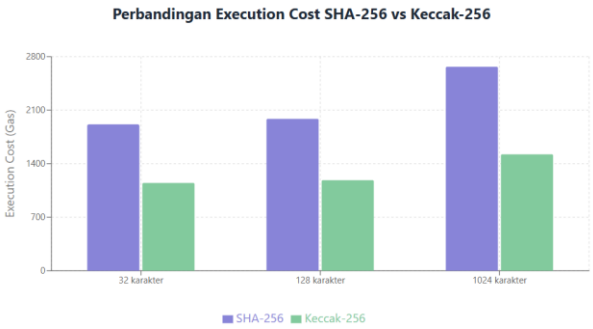
3. Hasil dan Pembahasan

Hasil penelitian ini diperoleh melalui serangkaian tahapan sistematis, dimulai dengan pembuatan data uji hingga penerapan dan evaluasi algoritma dalam kontrak pintar jaringan *Ethereum*. Data yang digunakan adalah *string* buatan dengan tiga variasi, pendek (32 karakter), sedang (128 karakter), dan panjang (lebih dari 1000 karakter). Peneliti membuat *string* ini secara manual untuk memastikan bahwa semuanya konsisten dan parameter pengujian dikontrol sepenuhnya. Setelah data disiapkan, langkah berikutnya adalah penerapan dua kontrak pintar yang berbeda yang menggunakan algoritma hash *SHA-256* dan *Keccak-256*. *Smart Contract* dibuat menggunakan bahasa *Solidity* dan diuji dalam lingkungan pengembangan *Remix IDE* dengan terhubung ke jaringan pengujian *Ethereum*. Pada tahap ini, setiap *input* diuji secara paralel pada kedua algoritma untuk memastikan bahwa kondisi pengujian adalah setara.

Pengujian dilakukan untuk mengukur tiga komponen utama: *execution cost* sebagai jumlah gas yang dibutuhkan secara langsung untuk menjalankan fungsi *hashing*, *gas fee* sebagai indikator efisiensi eksekusi fungsi hash, serta keamanan algoritma yang dievaluasi melalui pendekatan *brute force* untuk menilai ketahanannya terhadap serangan kriptografi. Hasil dari setiap pengujian dibandingkan guna mengidentifikasi perbedaan kinerja antara kedua algoritma. Penelitian ini tidak hanya menyajikan data perbandingan, tetapi juga memberikan analisis mendalam mengenai kelebihan dan kekurangan masing-masing algoritma. Temuan ini diharapkan dapat membantu pengembang blockchain dalam memilih algoritma hash yang paling sesuai untuk kebutuhan aplikasi *smart contract*.

p-ISSN: 2986-3929
e-ISSN: 3032-1948

3.1. Perbandingan *Execution Cost*



Gambar 2. Perbandingan *Execution Cost*

Berdasarkan gambar diatas, dilakukan perbandingan *execution cost* antara algoritma hash SHA-256 dan Keccak-256 berdasarkan panjang input yang berbeda, yaitu 32 karakter (pendek), 128 karakter (sedang), dan 1024 karakter (panjang). Tujuan dari pengujian ini adalah untuk menilai efisiensi komputasi dari masing-masing algoritma, karena *execution cost* mencerminkan jumlah gas yang dibutuhkan untuk menjalankan logika hashing di dalam kontrak pintar. Evaluasi ini penting karena semakin rendah execution cost, semakin ringan beban pemrosesan kontrak di jaringan Ethereum. Hasil menunjukkan bahwa untuk setiap kategori panjang string, Keccak-256 secara konsisten menunjukkan konsumsi gas yang lebih rendah daripada SHA-256.

Tabel 1. *Execution cost* SHA-256

Panjang String	SHA-256
32 karakter	1914 gas
128 karakter	1986 gas
1024 karakter	2667 gas

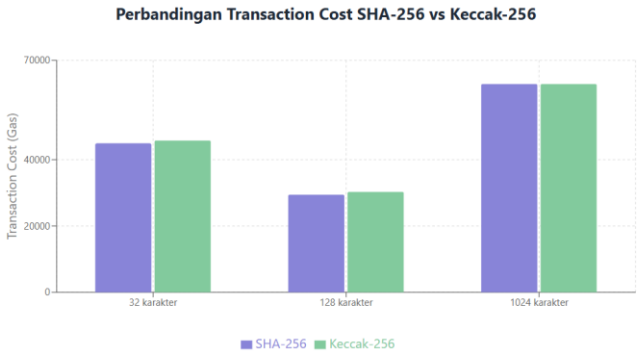
Tabel 2. *Execution cost* Keccak-256

Panjang String	Keccak-256
32 karakter	1148 gas
128 karakter	1184 gas
1024 karakter	1522 gas

Untuk input pendek 32 karakter, SHA-256 membutuhkan 1914 gas, sedangkan Keccak-256 hanya 1148. Untuk input sedang 128 karakter, SHA-256 membutuhkan 1986 gas, sedangkan Keccak-256 hanya 1184. Untuk input panjang 1024 karakter, SHA-256 membutuhkan 2667 gas, sedangkan Keccak-256 hanya 1522 gas. Data ini menunjukkan bahwa Keccak-256 lebih efisien. Data ini secara konsisten menunjukkan bahwa Keccak-256 memiliki *execution cost* yang lebih rendah dibandingkan SHA-256 pada berbagai panjang input.

p-ISSN: 2986-3929
e-ISSN: 3032-1948

3.2. Perbandingan Gas Fee



Gambar 3. Perbandingan *Transaction Cost*

Berdasarkan grafik hasil pengujian pada gambar 4 menunjukan bahwa algoritma *SHA-256* dan Keccak-256 tidak berbeda terlalu jauh dalam jumlah hal *Transaction cost* pada setiap kategori panjang string. Nilai *Transaction cost* kedua algoritma sedikit berbeda untuk setiap panjang string. Terdapat sedikit perbedaan antara string 32 dan 128 karakter. *Keccak-256* sedikit lebih besar daripada SHA-256, menunjukkan bahwa SHA-256 lebih baik dengan input pendek. Sebaliknya, pada string berukuran 1024 karakter, *SHA-256* dan *Keccak-256* memiliki nilai yang sama.

Tabel 3. *Gas Fee* SHA-256

Panjang String SHA 256	transaction cost	Gas Fee
32 karakter	44954 gas	44,954 × 0.887 × 10^{-9} = 0.000039874 ETH
128 karakter	29426 gas	29,426 × 0.887 × 10^{-9} = 0.000026105 ETH
1024 karakter	62820 gas	62,820 × 0.887 × 10^{-9} = 0.000055699 ETH

Tabel 4. *Gas Fee* Keccak-256

Panjang String keccak 256	transaction cost	Gass Fee
32 karakter	45763 gas	45,763 × 0.887 × 10^{-9} = 0.000040594 ETH
128 karakter	30271 gas	30,271 × 0.887 × 10^{-9} = 0.000026839 ETH
1024 karakter	62820 gas	62,820 × 0.887 × 10^{-9} = 0.000055699 ETH

Berdasarkan hasil perhitungan pada Tabel 3 dan Tabel 4, dapat disimpulkan bahwa algoritma SHA-256 memiliki efisiensi gas yang sedikit lebih baik dibandingkan dengan Keccak-256. Biaya transaksi (gas fee) dihitung dengan cara mengalikan jumlah gas yang digunakan (transaction cost) dengan harga gas sebesar 0.887 Gwei, kemudian dikonversi ke dalam satuan ETH menggunakan faktor 10^{-9}, karena 1 Gwei = 10^{-9} ETH.

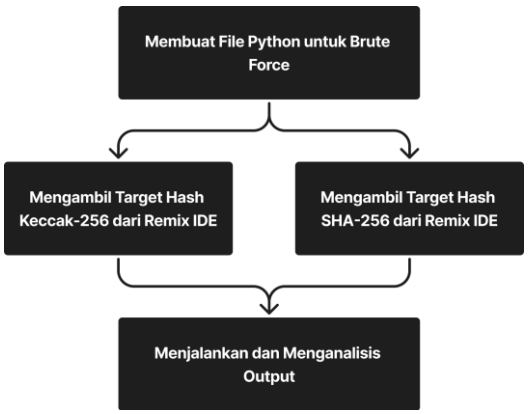
$$\text{Gas Fee (ETH)} = \text{Transaction Cost} \times \text{Gas Price } (0.887 \text{ Gwei} \times 10^{-9} \text{ ETH})$$

Gambar 4. Rumus *Gas Fee*

Transaction cost Keccak-256 sedikit lebih tinggi dibandingkan *SHA-256* pada string 32 dan 128 karakter, selisih 809 dan 845 gas, menunjukkan bahwa *SHA-256* lebih efisien untuk *input* pendek. Namun, pada *string* 1024 karakter, keduanya memiliki nilai yang sama, sehingga efisiensi relatif seimbang untuk data berukuran besar.

p-ISSN: 2986-3929
e-ISSN: 3032-1948

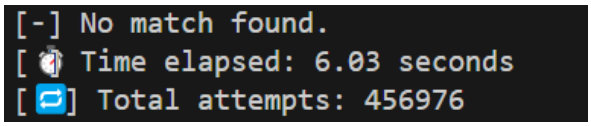
3.3. Perbandingan Keamanan Algoritma Hash



Gambar 5. Tahap Brute Force

Pada tahap ini, menggunakan pendekatan brute-force berbasis Python sebagai metode evaluasi praktis terhadap kekuatan dua algoritma hash populer, yaitu Keccak-256 dan SHA-256. Eksperimen ini dimulai dengan membuat berkas Python yang dimaksudkan untuk melakukan proses brute-force, dengan menghasilkan berbagai kombinasi string secara sistematis berdasarkan panjang tertentu, kemudian mencocokkannya dengan nilai *target hash* yang telah ditentukan sebelumnya. Nilai hash tersebut diperoleh dengan memanfaatkan fungsi keccak-256 dan sha-256 dalam smart contract Solidity, yang diambil dari Remix IDE. Setelah proses hashing terhadap string tertentu dilakukan, nilai output hash disalin dan digunakan sebagai acuan dalam program brute-force.

a. Brute Force Keccak-256



Gambar 6. Hasil Brute Force Keccak-256

Hasil percobaan brute-force program membutuhkan waktu sekitar 6.03 detik untuk mencoba seluruh kombinasi string dengan 4 karakter dari alfabet huruf kecil (a-z), seperti yang ditunjukkan pada gambar 6. Jumlah percobaan atau kombinasi total 456.976 dilakukan, yang sesuai dengan rumus sebagai berikut :

- Jumlah kombinasi untuk panjang k:

$$\text{Banyak kombinasi}^{k} = 26^{k}$$

Gambar 7. Rumus Kombinasi Huruf dalam String

Dalam kasus ini, karena k(panjang string) = 4, maka = 264 = 456.976 kombinasi, 26 di peroleh dari huruf kecil (a-z). Dari sini dapat dihitung kecepatan rata-rata program dalam waktu per hash:

p-ISSN: 2986-3929
e-ISSN: 3032-1948

$$\text{Waktu per hash} = \frac{\text{Total waktu}}{\text{Jumlah kombinasi}} = \frac{6 \text{ detik}}{456.976} = 0.0000132 \text{ detik}$$

Gambar 8. Rumus Waktu Hashing

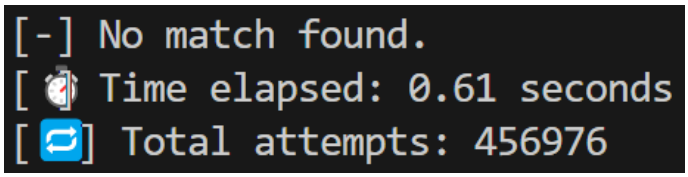
Dari gambar diatas waktu estimasi dihitung berdasarkan hasil untuk panjang 4, waktu yang dibutuhkan adalah sekitar 6 detik untuk 456.976 kombinasi, sehingga 1 hash/percobaan membutuhkan 0.0000132 detik.

Tabel 5. Estimasi Durasi Brute Force Keccak-256

Pajang (k)	Kombinasi (26k)	Estimasi Waktu (detik)	Estimasi Waktu (tahun)
1	26	26×0.0000132=0.000343	-
2	6767	0.0089	-
3	17.576	0.232	-
4	456.976	6.03	-
5	11.881.376	156.75 (2.6 menit)	-
6	308.915.776	4.077.692 (1.13 jam)	-
7	8.031.810.176	105.962.813 (29.4 jam)	-
8	208.827.064.576	2.754.190.450 (31.9 hari)	0.087 tahun
9	5.429.503.678.976	71.620.690.191	2.27 tahun
10	141.167.095.653.376	1.862.138.235.000	59 tahun
11	3.670.344.486.987.776	48.435.594.413.000	1.536 tahun

Semakin pendek string yang digunakan sebagai input dalam proses hashing, lebih cepat proses brute-force terhadap nilai hash. Sebagai contoh, algoritma Keccak256 hanya membutuhkan waktu sekitar enam detik untuk menguji string berisi empat karakter untuk melihat semua kombinasi yang mungkin. Namun, ketika target hash berasal dari hasil deploy smart contract yang menggunakan Keccak256, seperti yang ditunjukkan pada Remix IDE, panjang input sebenarnya mungkin jauh lebih panjang, bahkan dapat mencapai lebih dari 32 karakter. Dalam hal ini, jumlah kombinasi yang harus diuji meningkat secara eksponensial, menjadi 26^{32}, dan estimasi waktu yang dibutuhkan untuk mencoba semua kemungkinan, bahkan dengan komputer kontemporer, dapat mencapai jutaan hingga milyaran tahun. Ini menunjukkan bahwa algoritma hash Keccak256 memiliki tingkat keamanan yang sangat tinggi. Akibatnya, metode brute-force menjadi tidak praktis dan secara komputasi tidak layak untuk memulihkan input asli dari nilai hash.

b. Brute Force SHA-256



Gambar 9. Hasil Brute Force SHA-256

Hasil percobaan brute-force program membutuhkan waktu sekitar 0.61 detik untuk mencoba seluruh kombinasi string dengan 4 karakter dari alfabet huruf kecil (a-z), seperti

yang ditunjukkan pada gambar 6. Jumlah percobaan atau kombinasi total 456.976 dilakukan. rumus banyak dari kombinasi tetap sama, yang berbeda sekarang nilai hasil dari rumus waktu estimasi, sebagai berikut:

$$\text{Waktu per hash} = \frac{\text{Total waktu}}{\text{Jumlah kombinasi}} = \frac{0.61 \text{ detik}}{456.976} = 0.00000134 \text{ detik}$$

Gambar 10. Rumus Waktu Hashing

Dari gambar diatas waktu estimasi dihitung berdasarkan hasil untuk panjang 4, waktu yang dibutuhkan adalah sekitar 0.61 detik untuk 456.976 kombinasi, sehingga 1 hash/percobaan membutuhkan 0.00000134 detik.

Tabel 6. Estimasi Durasi Brute Force SHA-256

Pajang (k)	Kombinasi (26k)	Estimasi Waktu (detik)	Estimasi Waktu (tahun)
1	26	0.000035	-
2	6767	0.0009	-
3	17.576	0.02356	-
4	456.976	0.61235	-
5	11.881.376	15.92664	-
6	308.915.776	413.93617	0.0131 tahun
7	8.031.810.176	10,761.6256	0.3413 tahun
8	208.827.064.576	279,802.2565	8.87 tahun
9	5.429.503.678.976	7,272,858.9278	230.64 tahun
10	141.167.095.653.376	189,094,312.176	5,993.4 tahun
11	3.670.344.486.987.776	4,917,452,116.56	155,949.7 tahun

Hasil di atas menunjukkan bahwa brute force terhadap string pendek seperti 4 karakter dapat dilakukan dalam waktu kurang dari satu detik, tetapi waktu meningkat secara eksponensial seiring panjang input. Dengan kata lain, jika kita menggunakan input yang tampaknya pendek, seperti 32 karakter acak, jumlah kombinasi yang mungkin (26^{32}) akan sangat besar, dan brute force akan membutuhkan miliaran tahun, bahkan jika komputer tercepat sekalipun dapat melakukannya.

4. Kesimpulan

Berdasarkan hasil analisis komprehensif terhadap tiga aspek utama *execution Cost, gas Fee,* dan keamanan dapat disimpulkan bahwa:

- *Execution Cost*: Keccak-256 lebih murah untuk menjalankan fungsi hashing pada smart contract karena biaya operasinya lebih rendah daripada SHA-256 untuk semua panjang string.
- *Gas Fee*: *Keccak-256* sedikit lebih tinggi pada *input* 32 dan 128 karakter dibandingkan *SHA-256*, tetapi keduanya sama pada 1024 karakter, menunjukkan bahwa keduanya efektif untuk data besar.
- Aspek Keamanan: Meskipun kedua algoritma sangat tahan terhadap serangan brute-force, Keccak-256 membutuhkan waktu komputasi yang lebih lama, menunjukkan keunggulan dalam keamanan jangka panjang.

p-ISSN: 2986-3929
e-ISSN: 3032-1948

Temuan ini mendorong pengembang blockchain untuk mempertimbangkan lebih lanjut penggunaan *Keccak-256* dalam pengembangan *smart contract* yang lebih efisien dan hemat biaya, terutama karena biaya *execution*nya lebih rendah daripada *SHA-256*. Meskipun *Keccak-256* menunjukkan *gas fee* sedikit lebih tinggi untuk input pendek (32 dan 128 karakter), efisiensinya setara dengan SHA-256 pada data besar. Untuk penelitian lebih lanjut, disarankan untuk melakukan penelitian yang lebih mendalam tentang bagaimana algoritma hash ini berdampak pada skala jaringan blockchain yang lebih besar dan kompleks. Selain itu, disarankan untuk mempelajari optimasi lebih lanjut untuk penggunaan gas dan penyimpanan data. Selain itu, analisis keamanan juga perlu dilakukan untuk memastikan ketahanan algoritma dalam jangka panjang dalam konteks blockchain yang terus berkembang.

Daftar Pustaka

[1] H. Taherdoost, "Smart Contracts in Blockchain Technology: A Critical Review," *Information*, vol. 14, no. 2, p. 117, Feb. 2023, doi: https://doi.org/10.3390/info14020117.
[2] D. Lee, "Economics of Layer 2 Scaling: Estimating Price Elasticity of Layer 2 Gas Fees," Jan. 2024, doi: https://doi.org/10.2139/ssrn.4916662.
[3] S. Fan, H. Zhang, Y. Zeng, and W. Cai, "Hybrid Blockchain-Based Resource Trading System for Federated Learning in Edge Computing," *IEEE Internet of Things Journal*, pp. 1–1, 2020, doi: https://doi.org/10.1109/jiot.2020.3028101.
[4] M. M. A. Khan, H. M. A. Sarwar, and M. Awais, "Gas consumption analysis of Ethereum blockchain transactions," *Concurrency and Computation: Practice and Experience*, vol. 34, no. 4, Nov. 2021, doi: https://doi.org/10.1002/cpe.6679.
[5] O. Kuznetsov, O. Peliukh, N. Poluyanenko, S. Bohucharskyi, and I. Kolovanova, "Comparative Analysis of Cryptographic Hash Functions in Blockchain Systems." Available: https://ceur-ws.org/Vol-3550/paper7.pdf
[6] A. Sevin and A. Osman, "Comparative Study of Blockchain Hashing Algorithms with a Proposal for HashLEA," *Applied Sciences*, vol. 14, no. 24, pp. 11967–11967, Dec. 2024, doi: https://doi.org/10.3390/app142411967.
[7] O. Kuznetsov, O. Peliukh, N. Poluyanenko, S. Bohucharskyi, and I. Kolovanova, "Comparative Analysis of Cryptographic Hash Functions in Blockchain Systems." Accessed: Aug. 24, 2024. [Online]. Available: https://ceur-ws.org/Vol-3550/paper7.pdf
[8] R. Verma, N. Dhanda, and V. Nagar, "Enhancing Security with In-Depth Analysis of Brute-Force Attack on Secure Hashing Algorithms," *Proceedings of Trends in Electronics and Health Informatics*, pp. 513–522, 2022, doi: https://doi.org/10.1007/978-981-16-8826-3_44.