SYNTIA – Systems and Technologies in Artificial Intelligence Applications.
Volume 1 Issue 2 (June 2026). Page: 91-100.

Gas Cost of Keccak-256 and SHA-256 in an Ethereum Smart Contract

Rahmat Kurniawan[1], Yuri Yudhaswana Joefrie[2*], Sai Theja Kunta[3]

[1] *Informatics Engineering Study Program, Faculty of Engineering, Universitas Tadulako, Palu, Indonesia.*
[2] *Department of Information Technology, Faculty of Engineering, Universitas Tadulako, Palu, Indonesia.*
[3] *Department of Computer Science, Technische Universität Darmstadt, Darmstadt, Germany.*

**Corresponding author: yuri.yudhaswana@untad.ac.id*

Abstract: Smart contracts often store a hash of a record instead of the record, and Ethereum offers Keccak-256 as an opcode and SHA-256 as a precompiled contract. A Hardhat experiment on a course-evaluation contract compared their gas cost in 18 scenarios (2 algorithms, 3 functions, 3 input lengths) of 10 transactions each, and reported Keccak-256 as cheaper by 0.44–2.66%. We re-analyze the reported mean, minimum, maximum, and standard deviation. All 18 scenarios are reproduced exactly by nine repeated transactions at the minimum and one first transaction at the maximum, so the means mix two regimes. For the 850-character submit and verify functions the repeated transactions sit at the EIP-7623 calldata floor (56,030 gas) under both algorithms, which turns a first-transaction gap of 2,518–2,521 gas into a reported mean gap of 252. The first-transaction gaps follow 455 + 76.4w gas for w input words, within 8 gas, about 12 times the cost-schedule difference of 6 gas per word. Keccak-256 stays cheaper, by 0.4–4.1% of first-transaction gas, but the size of the gap is not a property of the hash functions alone. Per-transaction logs and the contract source are needed to confirm the reconstruction.

Keywords : Ethereum; gas cost; Keccak-256; SHA-256; smart contract.

Received: March 16, 2026 | Revised: April 29, 2026 | Accepted: May 25, 2026 | Published: June 30, 2026

1. Introduction

A smart contract that records documents, grades, or evaluations rarely stores the record itself. Storage on a public blockchain is expensive and visible to everyone, so the usual design stores a fixed-length hash and keeps the record elsewhere; anyone holding the record can later recompute the hash and compare. Ethereum charges for every step of such a design in units of gas, and the fee paid by the user is the gas used multiplied by the gas price [1], [2]. The Ethereum virtual machine (EVM) offers two hash functions that produce 256-bit digests: Keccak-256 as the opcode SHA3, and SHA-256 as a precompiled contract at address 0x02 [2]. SHA-256 is a federal standard that external systems understand [3]; Keccak-256 is the original Keccak submission, which differs from the standardized SHA3-256 in its padding and therefore produces different digests [2], [4], [5]. A developer who needs only an integrity check can use either, and the cost is one of the few quantities that separates them.

Blockchain applications for education, including credentials and evaluation records, are the setting of this study [6]. The contract examined here is a test bed for a course-evaluation process in which students evaluate lecturers (known in Indonesian universities as EDOM): it hashes an evaluator identity or an evaluation text and stores the digest. Gas cost is a recognized concern for such contracts. Studies of deployed code found recurring patterns that waste gas, and tools exist to detect them and to prove bounds on consumption [7]–[10]; the broader literature on contract weaknesses lists gas-related failures among the attack classes [11]. Comparative studies of hash functions in Indonesian-language journals mostly measure speed or avalanche behaviour off-chain and do not report EVM gas [12].

SYNTIA – Systems and Technologies in Artificial Intelligence Applications is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License

A measurement of gas looks easy because the EVM is deterministic: the same transaction in the same state costs the same. That property, however, makes repeated identical runs uninformative unless the state changes between them, and the cost of a transaction depends on the state it touches. Performance benchmarking in other fields has long separated start-up from steady-state behaviour for the same reason [13]. In this article we examine a published experiment of this kind, a comparison of Keccak-256 and SHA-256 on Hardhat Network with 18 scenarios of 10 transactions each. The experiment reported summary statistics only, and its conclusion was that Keccak-256 is cheaper in every scenario, by 0.44% to 2.66%. We ask four questions. (RQ1) How large is the gap in each scenario, and does the ranking hold in every regime? (RQ2) Do the reported means describe the cost a deployed contract would incur? (RQ3) Can the gap be explained by the published gas schedule? (RQ4) How much does the gap matter in practice, compared with other design choices?

The contributions are as follows. We show that the reported statistics of all 18 scenarios are reproduced exactly by one first transaction and nine repeated transactions, and use this to separate two cost regimes. We show that the smallest reported gap, 252 gas for the 850-character input, arises because the repeated transactions are billed at a calldata floor that is identical for both algorithms. We give a gas accounting that reproduces the first-transaction cost of every scenario to within a few thousand gas, and an empirical model of the gap between the algorithms as a function of input length. Finally, we state what a benchmark of this kind must record to support a claim. Section 2 describes the source experiment and our methods, Section 3 reports and discusses results, and Section 4 concludes.

2. Method

2.1. Source experiment

The source experiment used two contracts that differ only in the hash function: one calls keccak256, the other calls sha256 from Solidity 0.8.26 [14]. Each contract has three functions (Table 1). Two of them, submitEvaluation and verifyEvaluator, hash a string argument and store the digest in a mapping from bytes32 to bool. The third, getEvaluationResult, is a view function in the design; to obtain a gas figure the authors called it through a wrapper that is a transaction and writes the input string to a mapping keyed by the result hash. The wrapper is therefore a test construct, and its cost is dominated by storage rather than by reading. The Solidity source of both contracts is available in the project repository, and each of the three functions takes a single string argument, consistent with Table 1.

Table 1. Design of the source experiment.

Factor	Levels
Hash function	keccak256 (opcode SHA3, 0x20); sha256 (precompiled contract, address 0x02)
Function	submitEvaluation and verifyEvaluator (store a digest); getEvaluationResult through a state-changing wrapper (stores the input string)
Input length	24, 144, and 850 characters, that is 1, 5, and 27 words of 32 bytes
Repetitions	10 transactions per scenario; 18 scenarios; 180 transactions
Metric	receipt.gasUsed of each transaction, summarized as mean, minimum, maximum, and standard deviation
Environment	Hardhat 2.28.6 (Hardhat Network), Ethers.js 6.16.0, Node.js 24.5.0, Windows 10

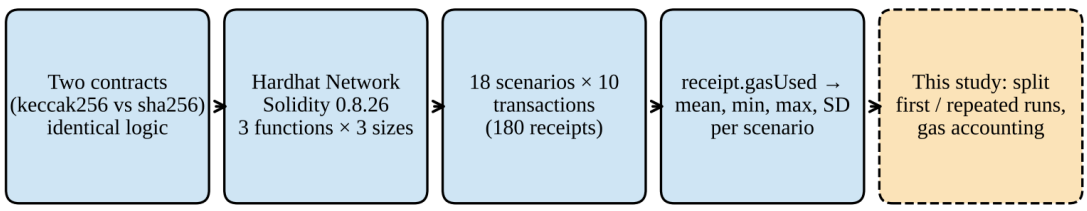


Figure 1. Source experiment and the analysis added in this study.

All runs used Hardhat Network, a local simulation of the EVM, so that gas price and network congestion do not vary [15]. The source reports the input lengths as 24, 107, and more than 685 characters in its method chapter and as 24, 144, and 850 in its results chapter; we use the results-chapter values, which are the ones consistent with the figures in Section 3.3 and Section 3.4. We confirmed this directly against the source's results chapter; the method chapter's 107 and 685+ figures look like a drafting inconsistency in the source rather than a second experiment. The source does not state the hardfork setting. The Hardhat 2 reference lists prague as the default [15], and Section 3.3 gives independent evidence that the Prague rules were applied.

2.2. Data used and the two-point reconstruction

We had the summary table of the source, not the 180 individual gas values: for each scenario the mean, minimum, maximum, and standard deviation of the 10 transactions. The per-transaction Hardhat logs were not available to us, so this reconstruction from the summary statistics is the best evidence we have; Section 3.8 flags it as an assumption to verify once the logs are obtained. Two features of the table suggested a structure. First, the maximum exceeds the minimum by 19,900 gas in all scenarios that write one storage slot. Second, in every scenario the mean equals the minimum plus one tenth of the range. This is what the data would show if nine transactions had cost the minimum m and one had cost the maximum M:

$$\bar{G} = m + \frac{M-m}{10}, \qquad s = \sqrt{\frac{9(m-\bar{G})^2+(M-\bar{G})^2}{10}} = 0.3\,(M-m) \tag{1}$$

The standard deviation in the source is the population value (divisor 10): for submitEvaluation with Keccak-256 and 24 characters, M − m = 19,900, and 0.3 × 19,900 = 5,970, which is the reported figure, whereas the sample standard deviation would be 6,293. We tested Eq. (1) on all 18 scenarios; the largest difference between the reconstructed and reported mean was 0.1 gas and between the reconstructed and reported standard deviation 0.2 gas, both within the rounding of the published table. We call the minimum the repeated-transaction cost and the maximum the first-transaction cost. The reconstruction is consistent with the data and, with 18 exact matches, very likely, but it is not a proof, because other sets of ten values can share the same four statistics.

The reason for two regimes is the state of the storage slot written by the function. A transaction that writes a nonzero value to a slot that was zero pays 22,100 gas: 20,000 for the write and 2,100 for the first (cold) access of the slot in that transaction. A later transaction that writes the same value to the same slot pays 2,100 for the cold access plus 100 for a no-op write, or 2,200 [16]. The difference is 19,900 gas, the observed range. In the experiment every repetition used the same input, so the first transaction created the slot and the other nine rewrote it.

2.3. Gas accounting

Since the 2025 Pectra upgrade the gas charged for a transaction has a floor that depends on its calldata [17], [18]. With z zero bytes and n_z nonzero bytes in the calldata, the number of calldata tokens is $T = z + 4n_z$, and

$$G_{\text{tx}} = 21000 + \max(4T + G_{\text{exec}},\ 10\,T) \tag{2}$$

where G_{exec} is the gas used by the EVM during execution, including storage. The first term inside the maximum is the usual price of 16 gas per nonzero and 4 per zero byte, plus execution; the second is the floor. Transactions with large calldata and cheap execution are billed at the floor. For each scenario we computed T from the ABI encoding of one string argument (4-byte selector, 32-byte offset, 32-byte length, data padded to a multiple of 32 bytes), the intrinsic cost 21,000 + 4T, the storage cost (22,100 per slot for the first transaction and 2,200 for repeated ones, with $1 + \lceil n/32 \rceil$ slots for a string of $n \geq 32$ bytes and one slot for a mapping to bool), and took the remainder as residual execution. The hash opcode or call is part of the residual.

2.4. Model of the gap between the algorithms

The Yellow Paper prices the SHA3 opcode at 30 gas plus 6 per word of input, and the SHA-256 precompile at 60 plus 12 per word [2]. A Solidity call to the precompile is a STATICCALL, and since EIP-2929 precompile addresses are warm, so the call costs 100 gas for the access rather than the 700 gas introduced by EIP-150 [16], [19]. For an input of w words the schedule therefore predicts a difference $\Delta_0(w)$, and we compare it with a straight line fitted by least squares to the observed first-transaction gaps:

$$\Delta_0(w) = (60 + 12w) - (30 + 6w) + 100 = 130 + 6w, \qquad \Delta(w) = \alpha + \beta\, w \tag{3}$$

The fit uses nine gaps at three values of w (1, 5, and 27). It is a line through three clusters, not a sweep, so it describes these inputs and is not tested between or beyond them.

2.5. Practical cost

To judge the size of a gap we convert it to Ether (ETH) at a gas price of 1 gwei, the default initial base fee of Hardhat Network [15], and at 20 gwei, the value used in the source thesis. We do not know the price at which a deployment would operate, and the conversion is an illustration; no exchange rate to a currency is applied. The share of transaction gas is computed relative to the Keccak-256 first-transaction gas of the same scenario.

3. Results and Discussion

3.1. Gaps in the two regimes

Table 2 gives the first-transaction and repeated-transaction gas of each scenario and the gaps between the algorithms. In seven of the nine scenarios the two regimes give the same gap, and the reported mean gap equals both. In the two remaining scenarios, the 850-character submit and verify functions, the gap is 2,518 and 2,521 gas for the first transaction, 0 for the repeated transactions, and 252 in the reported mean, which is one tenth of the first-transaction gap (Fig. 2). Keccak-256 is cheaper in every scenario in the first-transaction regime, and cheaper or equal in the repeated regime.

Table 2. Gas of the first and the repeated transactions, and the gap (SHA-256 minus Keccak-256), by scenario. Function: submit = submitEvaluation, verify = verifyEvaluator, get = getEvaluationResult through the wrapper; length in characters.

Scenario	Keccak-256 first	Keccak-256 repeated	SHA-256 first	SHA-256 repeated	Gap first	Gap repeated	Reported mean gap
submit, 24	46,308	26,408	46,842	26,942	534	534	534
submit, 144	48,588	28,688	49,423	29,523	835	835	835
submit, 850	61,699	56,030	64,217	56,030	2,518	0	252
verify, 24	46,309	26,409	46,846	26,946	537	537	537
verify, 144	48,589	28,689	49,427	29,527	838	838	838
verify, 850	61,700	56,030	64,221	56,030	2,521	0	252
get, 24	46,666	26,766	47,194	27,294	528	528	528
get, 144	159,882	40,623	160,711	41,452	829	829	829
get, 850	661,063	104,004	663,575	106,516	2,512	2,512	2,512

First = maximum and repeated = minimum of the 10 transactions of the source table.

The source ranks the algorithms by the mean over all nine scenarios, 52,485 against 53,276 gas, a difference of 791. The nine reported mean gaps average 791, but the first-transaction gaps average 1,295 gas, 0.99% of the mean Keccak-256 first-transaction gas, and the repeated-transaction gaps average 735 gas, 1.68% of the corresponding mean. The aggregate is thus sensitive to the regime by a factor of almost two, and the source reports neither. An answer to RQ1 is that the ranking holds in both regimes, with a gap between 0 and 2,521 gas that depends on regime and on input length.

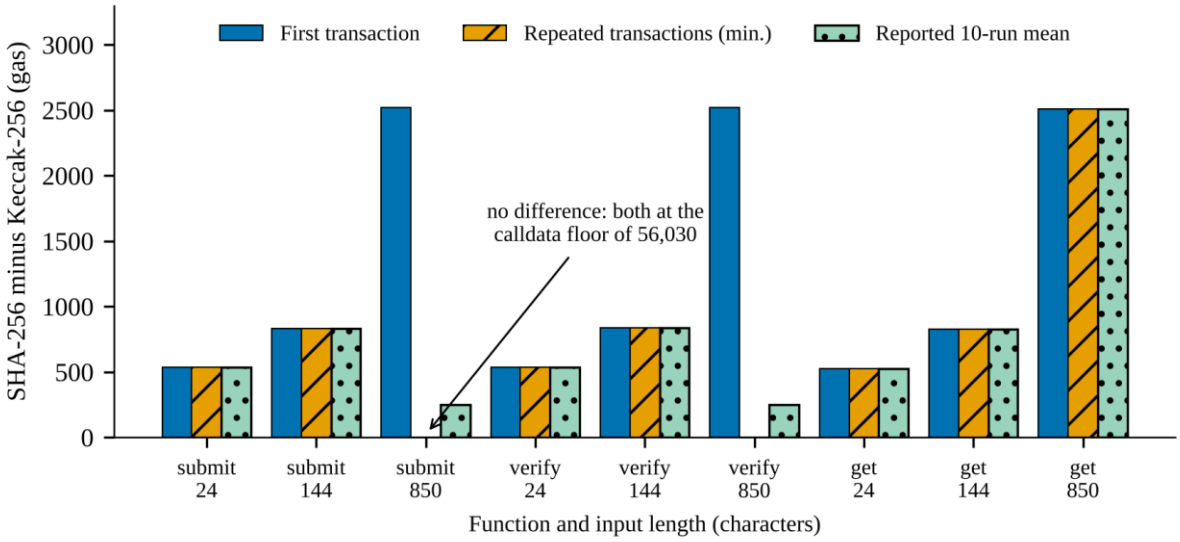


Figure 2. Gap between SHA-256 and Keccak-256 by scenario in three views: the first transaction, the repeated transactions, and the reported 10-run mean.

3.2. Which regime represents a deployment

The repeated regime measures a contract that writes the same digest to the same slot again, which an evaluation process does not do: each evaluation has its own content and its own key, and each evaluator is verified once per key. A deployed contract of this design pays the first-transaction price at every call, unless the design rewrites keys. The first-transaction column of Table 2 is therefore the better estimate of cost for the design described, and the reported means, which weight it at one tenth, understate the first-transaction cost by 8% to 76% depending on the scenario (for example, 28,398 reported against 46,308 gas for the 24-character submission, and 159,710 against 661,063 for the 850-character wrapper). This conclusion depends on the key design of the final system. We confirmed with the system design that each evaluation is keyed by a unique evaluator-evaluation pair, so every call writes to a new storage slot rather than rewriting an existing one. It answers RQ2: the means describe neither regime.

3.3. The 850-character case and the calldata floor

The 252-gas gap of the 850-character submit and verify functions is the smallest in the source, and the source explains it by storage costs dominating the total. The reconstruction suggests a different cause. The repeated transactions cost 56,030 gas under both algorithms and under both functions, which is not what a cost model with different hash costs predicts. The ABI encoding of an 850-character string has 932 bytes: 4 of selector, 32 of offset, 32 of length, and 864 of padded data. Of these, 857 are nonzero (4 in the selector, 1 in the offset, 2 in the length 0x0352, and 850 characters) and 75 are zero, so $T = 75 + 4 \times 857 = 3{,}503$ tokens. Eq. (2) gives a floor of $21{,}000 + 10 \times 3{,}503 = 56{,}030$ gas, exactly the observed value.

The execution cost of the repeated transactions with the long input (about 37,200 + 4,600 = 41,800 gas) is below the floor, so they are billed the floor, which depends on the calldata and is the same for both algorithms. The first transaction pays 22,100 for a new slot and exceeds the floor (61,699 and 64,217 gas). The reported mean gap is therefore 2,518 / 10 = 252 gas, and the smallest difference of the source is an artifact of billing in nine of the ten transactions. The floor applies only from the Pectra upgrade [17], [18]; its exact match is also evidence that the experiment ran under the Prague rules and with the 850-character input. For the 144-character input the floor is 27,780 gas, below the cost of every transaction, and for the 24-character input it is 22,900; neither is affected.

3.4. Where the gas goes

Table 3 and Fig. 3 decompose the Keccak-256 transactions with Eq. (2). In submit the intrinsic cost and the storage write are known, and the residual execution, which contains the hashing, grows from 2,448 gas for 24 characters to 2,776 for 144 and 4,587 for 850: 82 gas per word between the three lengths. The wrapper writes the whole string: a 24-character string occupies one slot, a 144-character string six slots (the length and five words) and an 850-character string 28 slots (the length and 27 words). The difference between the first and repeated transactions is 19,900 gas for one slot, 119,259 for six slots (6 × 19,900 = 119,400, a deviation of 141 gas), and 557,059 for 28 slots (557,200, again 141), which supports the storage layout. The accounting thus reproduces the first-transaction cost of every scenario with a residual execution between 2,400 and 7,300 gas, and the repeated cost of the same functions with the same residuals, except the floor-limited cases and a 141-gas deviation.

Table 3. Gas accounting for Keccak-256 (gas; intrinsic = 21,000 + 4T; storage 22,100 per slot for the first and 2,200 for repeated transactions; residual = observed − intrinsic − storage).

Scenario	Intrinsic	Slots	First: observed	First: residual	Repeated: observed	Repeated: residual
submit, 24	21,760	1	46,308	2,448	26,408	2,448
submit, 144	23,712	1	48,588	2,776	28,688	2,776
submit, 850	35,012	1	61,699	4,587	56,030	floor
get, 24	21,760	1	46,666	2,806	26,766	2,806
get, 144	23,712	6	159,882	3,570	40,623	3,711
get, 850	35,012	28	661,063	7,251	104,004	7,392

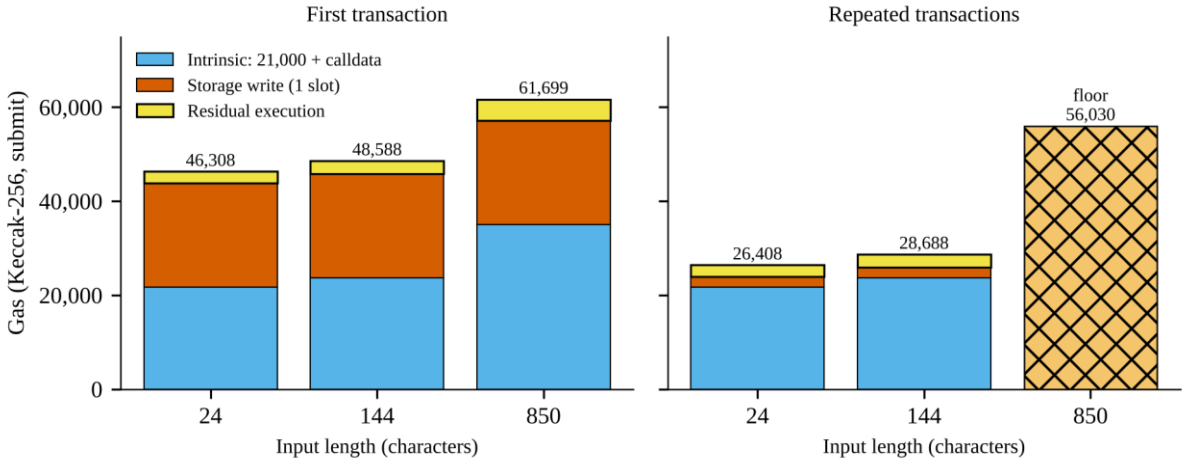


Figure 3. Composition of the gas of submitEvaluation with Keccak-256 for the first transaction and for repeated transactions; the 850-character repeated transactions are billed at the calldata floor.

Two observations follow. First, the hash itself is a very small part of a transaction: the SHA3 opcode costs 36 gas for 24 characters and 192 for 850, which is 0.08% and 0.31% of the first-transaction gas. Second, calldata and storage dominate, and they are the same for both algorithms. The quantity in which the algorithms differ is therefore a small difference between two small terms, which is a reason to measure it carefully and to report it as a gap with its regime rather than as a percentage of a total that is mostly shared.

3.5. Size of the gap and the cost schedule

The first-transaction gaps follow Eq. (3) with $\alpha = 454.6$ gas and $\beta = 76.4$ gas per word; the largest difference between a gap and the line is 7.4 gas (Fig. 4, Table 4). The repeated-transaction gaps of the seven scenarios outside the floor follow the same line within 7 gas. The line is far above the schedule difference $\Delta_0(w) = 130 + 6w$, whose slope is 6 gas per word: the observed slope is about 12.7 times larger, and the

observed gap is 3.9 times the schedule at w = 1 and 8.6 times at w = 27. The source attributes the difference to a CALL overhead of about 700 gas [19]. That value predates the access-list pricing of EIP-2929 under which precompiles are always warm and a call costs 100 gas [16]; even 700 would not explain a slope.

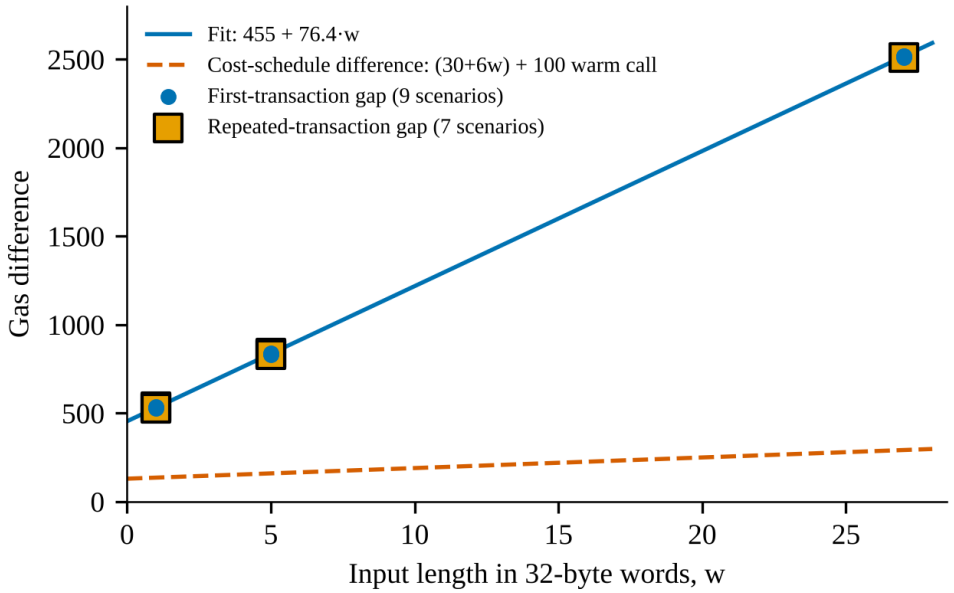


Figure 4. Observed gap against input length in words, the fitted line, and the cost-schedule difference.

Table 4. Observed first-transaction gap, fitted line, and cost-schedule difference (gas).

Words, w	Observed gap	Fitted: 454.6 + 76.4w	Schedule: 130 + 6w	Observed / schedule
1 (24 characters)	528–537	531	136	3.9
5 (144 characters)	829–838	836	160	5.2
27 (850 characters)	2,512–2,521	2,517	292	8.6

The source gives no code, so we cannot say what adds the difference, and we avoid a mechanism that the data do not identify. Plausible sources are the code the compiler generates around the precompile call (copying the argument to memory, checking the return value, reading the result) and a different handling of the string for each algorithm, both of which scale with the number of words. An opcode trace of one transaction per scenario (for example with a debug trace of the Hardhat node) would settle the question and is the first check we recommend. This answers RQ3 in part: the published schedule explains the ranking, since SHA-256 costs more per word than SHA3 in the schedule, but explains only about a twelfth of the slope.

3.6. Practical size of the gap

Table 5 converts the first-transaction gaps to ETH. At 1 gwei, the largest gap of about 2,500 gas costs 2.5 µETH per transaction, and 10,000 transactions cost 0.025 ETH; at 20 gwei the same figures are twenty times larger. The share of transaction gas is 1.2% to 1.7% for short and medium inputs, 4.1% for the 850-character submit and verify functions, and 0.4% for the wrapper, in which storage of the string dominates. A choice of hash function therefore changes the fee by about 1% to 4%, whereas the choice to store a string instead of its digest changes it by a factor of 14: the 850-character wrapper costs 661,063 gas against 46,308 for a 24-character submission, and its 28 storage slots alone cost 618,800. Reducing the data written on chain is the consistent recommendation of the work on gas inefficiency [7], [8], and it removes the dependence of the cost on the hash function altogether when the digest is computed off-chain. This is the answer to RQ4: the gap is real and small compared with design choices that the contract author controls.

Table 5. Illustration of the first-transaction gap in ETH (μETH = 10^{-6} ETH).

Scenario	Gap (gas)	Share of Keccak-256 gas	Per tx at 1 gwei (μETH)	Per tx at 20 gwei (μETH)	10,000 tx at 1 gwei (ETH)	10,000 tx at 20 gwei (ETH)
submit, 24	534	1.15%	0.53	10.7	0.0053	0.107
submit, 144	835	1.72%	0.84	16.7	0.0084	0.167
submit, 850	2,518	4.08%	2.52	50.4	0.0252	0.504
get, 850	2,512	0.38%	2.51	50.2	0.0251	0.502

3.7. What a benchmark of this kind should record

Four practices would have prevented the two problems found here. (i) Record every transaction, not only a summary: a table of 180 numbers makes the two regimes visible at once, whereas a mean over them hides it, and a standard deviation of 5,970 gas with a mean of 28,398 would be a surprise in a deterministic system. (ii) Decide which regime the claim is about and construct it: for a new key per call, use a new input or key in every transaction; for a rewrite, say so. In JVM benchmarking, start-up and steady-state results are reported separately because they answer different questions [13], and the storage state plays the same role here. (iii) State the hardfork, because the calldata floor and the access-list prices change the result and are not visible in the code. (iv) Vary the input length over more than three values, so that a relationship such as Eq. (3) is estimated rather than interpolated. Ten identical transactions add nothing in a deterministic simulator; ten transactions with different keys and inputs would.

3.8. Limitations

The main limitation is that we worked from summary statistics. The two-point reconstruction matched all 18 scenarios, the floor value matched exactly, and the storage differences matched to 141 gas, which together make an error in the interpretation unlikely, but the per-transaction values should be checked before any of the figures is reported as a measurement. The line of Eq. (3) rests on three input lengths. Hardhat Network simulates the EVM and applies the gas schedule of the chosen hardfork; results on another chain, on a layer-2 network, or under a different hardfork can differ, in particular for the floor, which applies only above the Pectra upgrade [17]. The contract source is not available to us, so the residual execution is not attributed to operations. The wrapper function is a test construct, and the cost of reading a stored evaluation by an off-chain caller is zero. Finally, gas is one criterion: SHA-256 may be required for interoperability with systems that follow the federal standard [3], and the two functions are generally considered secure for an integrity check of this kind [4].

4. Conclusion

Re-analysis of the summary statistics of a Hardhat experiment on a course-evaluation contract supports the ranking of the source: Keccak-256 costs less gas than SHA-256 in every scenario, in the first-transaction regime by 528 to 2,521 gas, or 0.4% to 4.1% of transaction gas. It does not support the size of the gap reported in the means. All 18 scenarios consist of one first transaction and nine repeated ones, which differ by 19,900 gas per written slot, so the means describe neither a fresh nor a rewritten record. The smallest reported gap, 252 gas for 850 characters, arises because repeated transactions are billed at the EIP-7623 calldata floor of 56,030 gas, which is identical for both algorithms.

The first-transaction gap follows 455 + 76.4w gas for w input words, which is about twelve times the cost-schedule difference; the source of the extra cost is not identified and is the first thing to examine with an opcode trace. For a contract author the practical points are that the hash choice changes the fee by about 1% to 4%, that storing less data on chain matters far more, and that SHA-256 is justified when interoperability with standardized systems is required. The next steps are to publish the per-transaction logs and the contract source, to repeat the measurement with a new key and input in every transaction and with more than three input lengths, to trace the opcodes, and to test the result on a public test network and a layer-2 network.

AI Disclosure Statement

The authors used Generative AI and AI-Assisted Technologies throughout the preparation of this manuscript: to draft and revise portions of the text, to write and run the scripts behind the experiments and statistical analyses, to generate the figures included in the revision, and to format the document from the authors' own research data and source code. The authors verified the scripts and their outputs, reviewed the resulting text, cross-checked every reference against its original source, and accept full responsibility for the content of this publication.

Author Contributions Statement

Rahmat Kurniawan: Conceptualization, Methodology, Software, Investigation, Formal Analysis, Data Curation, Visualization, Writing – Original Draft.
Yuri Yudhaswana Joefrie: Supervision, Validation, Writing – Review and Editing.
Sai Theja Kunta: Validation, Writing – Review and Editing.

References

[1] V. Buterin, "Ethereum white paper: A next-generation smart contract and decentralized application platform," 2014. [Online]. Available: https://ethereum.org/en/whitepaper/

[2] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," Shanghai version efc5f9a, Feb. 2025. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf

[3] National Institute of Standards and Technology, "Secure hash standard (SHS)," Federal Information Processing Standards Publication 180-4, Aug. 2015, doi: 10.6028/NIST.FIPS.180-4.

[4] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "On the indifferentiability of the sponge construction," in *Advances in Cryptology – EUROCRYPT 2008*, Lecture Notes in Computer Science, vol. 4965. Berlin, Germany: Springer, 2008, pp. 181–197, doi: 10.1007/978-3-540-78967-3_11.

[5] National Institute of Standards and Technology, "SHA-3 standard: Permutation-based hash and extendable-output functions," Federal Information Processing Standards Publication 202, Aug. 2015, doi: 10.6028/NIST.FIPS.202.

[6] A. Alammary, S. Alhazmi, M. Almasri, and S. Gillani, "Blockchain-based applications in education: A systematic review," *Appl. Sci.*, vol. 9, no. 12, Art. no. 2400, Jun. 2019, doi: 10.3390/app9122400.

[7] T. Chen, X. Li, X. Luo, and X. Zhang, "Under-optimized smart contracts devour your money," in *Proc. IEEE 24th Int. Conf. Softw. Anal., Evol. Reengineering (SANER)*, 2017, pp. 442–446, doi: 10.1109/SANER.2017.7884650.

[8] T. Chen, Z. Li, H. Zhou, J. Chen, X. Luo, X. Li, and X. Zhang, "Towards saving money in using smart contracts," in *Proc. 40th Int. Conf. Softw. Eng.: New Ideas Emerg. Results (ICSE-NIER)*, 2018, pp. 81–84, doi: 10.1145/3183399.3183420.

[9] N. Grech, M. Kong, A. Jurisevic, L. Brent, B. Scholz, and Y. Smaragdakis, "MadMax: Surviving out-of-gas conditions in Ethereum smart contracts," *Proc. ACM Program. Lang.*, vol. 2, no. OOPSLA, pp. 1–27, 2018, doi: 10.1145/3276486.

[10] E. Albert, J. Correas, P. Gordillo, G. Román-Díez, and A. Rubio, "GASOL: Gas analysis and optimization for Ethereum smart contracts," arXiv:1912.11929, 2019.

[11] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on Ethereum smart contracts (SoK)," in *Principles of Security and Trust*, Lecture Notes in Computer Science. Berlin, Germany: Springer, 2017, pp. 164–186, doi: 10.1007/978-3-662-54455-6_8.

[12] J. S. G. Sinaga, N. Sitorus, and S. L. Samosir, "Analisis kinerja algoritma hash pada keamanan data: Perbandingan antara SHA-256, SHA-3, dan Blake2" (in Indonesian), *Jurnal Quancom: Quantum Computer Jurnal*, vol. 2, no. 2, pp. 9–16, Dec. 2024, doi: 10.62375/jqc.v2i2.432.

[13] A. Georges, D. Buytaert, and L. Eeckhout, "Statistically rigorous Java performance evaluation," in *Proc. 22nd Annu. ACM SIGPLAN Conf. Object-Oriented Program. Syst. Lang. Appl. (OOPSLA)*, 2007, pp. 57–76, doi: 10.1145/1297027.1297033.

[14] "Solidity 0.8.26 documentation." [Online]. Available: https://docs.soliditylang.org/en/v0.8.26/ (accessed Oct. 12, 2026).

[15] "Hardhat Network reference," Hardhat 2 documentation. [Online]. Available: https://v2.hardhat.org/hardhat-network/docs/reference (accessed Oct. 12, 2026).

[16] V. Buterin and M. Swende, "EIP-2929: Gas cost increases for state access opcodes," Ethereum Improvement Proposals, Sep. 2020. [Online]. Available: https://eips.ethereum.org/EIPS/eip-2929

[17] T. Wahrstätter and V. Buterin, "EIP-7623: Increase calldata cost," Ethereum Improvement Proposals, Feb. 2024. [Online]. Available: https://eips.ethereum.org/EIPS/eip-7623

[18] T. Beiko, "EIP-7600: Hardfork Meta - Pectra," Ethereum Improvement Proposals, Jan. 2024. [Online]. Available: https://eips.ethereum.org/EIPS/eip-7600

[19] V. Buterin, "EIP-150: Gas cost changes for IO-heavy operations," Ethereum Improvement Proposals, Sep. 2016. [Online]. Available: https://eips.ethereum.org/EIPS/eip-150