Mitigating ASIC Monopolization in Blockchain Consensus: A Dynamic Multi-Stage SHA-256 and SHA-3 Interleaving Protocol

Peter Wongsoredjo - 13523039
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: peterwongsoredo@gmail.com , 13523039@std.stei.itb.ac.id

***Abstract*—Proof-of-Work (PoW) blockchains anchor their security in the assumption that hashing power is broadly distributed. In practice, the maturation of Application-Specific Integrated Circuits (ASICs) has concentrated that power within a handful of industrial fabrication houses, eroding the egalitarian premise of permissionless consensus. This paper introduces the ASIC-Resistant Interleaving Protocol (ARIP), a consensus-layer extension that replaces a static single-hash kernel with a dynamic, nonce-parity-driven pipeline that interleaves SHA-256 and SHA-3 (Keccak) across successive mining attempts. Because the two primitives derive from architecturally divergent constructions, Merkle–Damgård compression versus the sponge permutation, the alternating composite forces any fixed-function datapath to stall, denying ASIC designers a single circuit they can optimize end-to-end. We implement ARIP alongside baseline SHA-256 and SHA-3 in Python and benchmark all three across difficulty levels one through five, measuring wall-clock latency via a high-resolution performance counter and processor footprint via system utilization sampling. Results confirm that ARIP incurs a deliberate, structurally rooted computational premium, roughly twice the per-attempt cost, while preserving a commodity-CPU hardware footprint. We argue this latency penalty is precisely the disruption mechanism, not an inefficiency.**

***Keywords*—Blockchain, Cryptographic Agility, Proof-of-Work, SHA-256, SHA-3, ASIC Resistance**

I. INTRODUCTION

The security of a Proof of Work on a blockchain is a direct result of how widely its hashing power is distributed. Satoshi Nakamoto's original model framed mining as a one CPU one vote lottery, where honest participants form the majority of computational effort, deterring adversaries from rewriting history [1]. This presumes an equal playing field of commodity hardware, yet the reality of the past decade has been the opposite.

The introduction of Application-Specific Integrated Circuits (ASICs) has compressed the economics of mining into a narrow oligopoly. ASICs are computer chips meant to solve one thing at a speed far faster than any programmable device, i.e. solving the SHA-256 equation on bitcoin. The consequence is structural, profitable participation now requires access to bleeding-edge semiconductor production, and capital that only a certain firm possesses.

We frame the adversary not as a single malicious miner, but as the *centralizing market force* itself. Three concrete threats emerge from this, first the 51% attack. When a coalition of miners, each backed by an ASIC vendor, collectively approach a majority of network hashrate, they can then censor transactions, perform double spends, and reorganize the chain. Second, when a manufacturer dominates ASIC fabrication, it can withhold inventory, monopolizing the ASIC market, and mine on its own hardware before shipping, converting the advantage itself. Third, the barrier to entry rises until ordinary participants are excluded entirely, contradicting the foundational premise that anyone may join the consensus with commodity equipment.

We contend that ASIC resistance should be treated as a property of cryptographic agility rather than a static choice of a single hard ASIC function, since a fixed hash function, however memory hard it is, eventually yields, because the target is stationary. ASICs Resistant Interleaving Protocol (ARIP) operationalizes this thesis, by routing each mining attempt through one of two architecturally incompatible hash constructions, selected dynamically by the parity of the candidate nonce. ARIP forces any potential ASIC to embed both a Merkle-Damgard engine and a sponge engine and switch between them unpredictably, destroying the advantage that makes single-function ASICs economical.

II. THEORETICAL FRAMEWORK

A. *Blockchain Architecture*

At its architectural core, the blockchain is a decentralized, peer-to-peer distributed ledger designed to operate maliciously without a centralized clearing house or trusted intermediary. This framework organizes data into distinct block bodies where transaction sets are securely condensed into an efficient Merkle root [6]. This root is packaged into the block header alongside metadata including a version, live timestamp, difficulty target, and search nonce. Immutability is strictly achieved via cryptographic hash pointers, every block header embeds the unique digest of its immediate predecessor, generating a backward-linked chronological sequence. Any unauthorized perturbation of historical data alters that block's hash and immediately fractures the sequential chain of pointers downstream, creating a structural disruption that is instantly audited and rejected by network nodes holding synchronized copies of the ledger.

To securely govern updates to this shared ledger across adversarial participants, the network employs digital signatures and a decentralized consensus protocol. The network reaches agreement on the canonical chain using Proof-of-Work (PoW), a consensus rule requiring miners to resolve a cryptographic puzzle by executing a brute-force search for a nonce. This process demands that the block header's cumulative hash falls below a network-defined difficulty target, establishing a computationally expensive lottery that is trivial for peer nodes to verify. Because this mechanism forces nodes to expend massive computational work to secure historical integrity, the internal structure of the hash function acts as the definitive economic bottleneck, deciding whether consensus remains egalitarian or succumbs to specialized hardware monopolies.

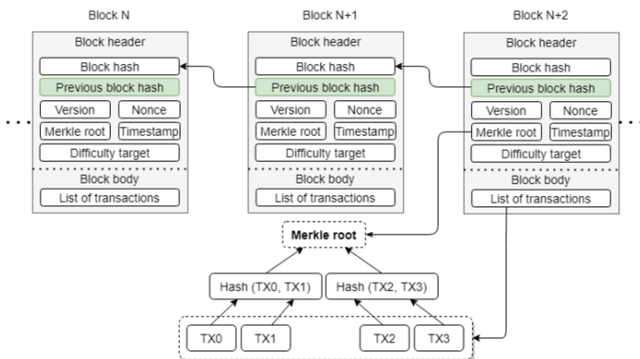


Figure 2.1 Blockchain Components
Source:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi-dan-Koding/2025-2026/40-Penggunaan-kriptografi-di-dalam-blockchain-2026.pdf

B. *The Sponge Construction (SHA-3 / Keccak)*

As decrypted, the sponge construction means to "absorb input data into a state and squeeze out the hash value" [2]. The internal state has a width

$$b = r + c,$$

Partitioned into a rate r (bits exchanged with the message for the data input and the hash output) and a capacity c (the hidden portion that is never directly touched by input or output and that governs the security level). For SHA3-256, $b = 1600$ bits, with $c = 512$, and $r = 1088$.

Hashing proceeds in two phases, in the absorbing phase, the padded message is split into r-bit blocks, each XORed into the portion of the state, and followed by an application of the Keccak-f permutation.

$$S_i \leftarrow f(S_{i-1} \oplus (P_i \,||\, 0^c)),\ i = 1, \ldots, k$$

Once all input is absorbed, the squeezing phase truncates successive r-bit chunks, Z_j, from the internal state, until the requested length is produced

$$Z_j \leftarrow Trunc_r(S_{k+j-1}),\ j = 1, 2, \ldots, m$$

With the internal state renewal for the next turn being

$$(S_{k+j}) \leftarrow f(S_{k+j-1})$$

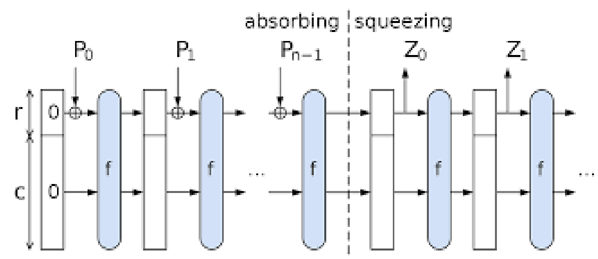


Figure 2.2 The SHA-3 Algorithm
Source:
https://medium.com/@kylacoin/exploring-the-double-sha-3-hashing-algorithm-strengths-applications-and-security-11c65f1d71d8

C. *The Merkle-Damgard Construction (SHA-256)*

SHA-256, is a classical Merkle-Damgard iterated hash with a fixed 256-bit output [3]. The padded message is divided into 512-bit blocks $M_1, \ldots, M_n$, and a fixed input length compression function g, is chained across them from an initialization vector IV.

$$H_0 = IV,\ H_i = g(H_{i-1}, M_i),\ digest = H_n,$$

Each invocation of the compression function g expands the 512-bit block into a 64-word message schedule. It then runs 64 rounds of modular additions (modulo 2^{32}) and bitwise logical operations (such as majority and choose functions) across eight 32-bit working registers.

By design, SHA-256 enforces a strict avalanche effect, a single bit perturbation in the input block propagates thoroughly, causing approximately half of the output bits to flip rendering the resulting digests statistically uncorrelated. This deterministic randomness is the exact cryptographic property exploited by Proof-of-Work (PoW) consensus

mechanisms, where each sequential nonce increment acts as an independent, uniform lottery draw.

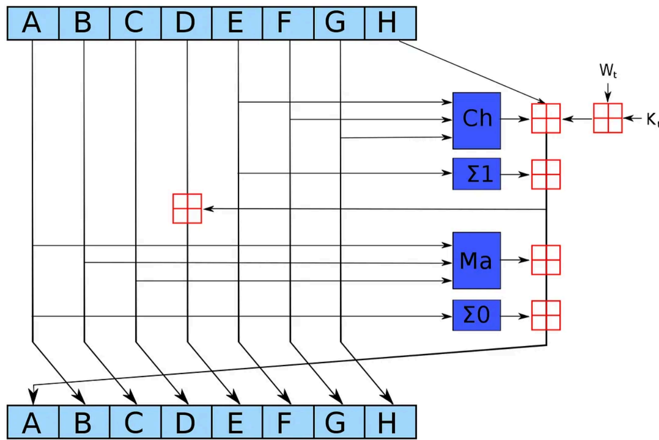


Figure 2.3 SHA-256 Algorithm Visualization
Source:
https://infosecwriteups.com/breaking-down-sha-256-algorithm-2ce61d86f7a3

D. Multi-Hash ASIC Mitigation and the ARIP Contrast

Prior multi-hash Proof-of-Work (PoW) protocols mitigate hardware specialization by selecting different hash functions based on block data. While this forces manufacturers to implement all candidate circuits, it allows them to idle unused circuits during a given block to optimize power efficiency. ARIP sharpens this defense by composing both functions into a single, nested evaluation where the execution order is governed strictly by nonce parity

$$H(M,\text{nonce}) = \begin{cases} \text{SHA3-256}(\text{SHA256}(M)), & \text{nonce} \equiv 0 \pmod 2, \\ \text{SHA256}(\text{SHA3-256}(M)), & \text{nonce} \equiv 1 \pmod 2. \end{cases}$$

Two properties distinguish this from a mere function selector. First, every attempt exercises both engines, so there is no idle circuit to switch off, the work is genuinely doubled, not multiplexed. Second, because the search increments the nonce by one on each failed attempt, the parity, and therefore the composite ordering, alternates on every single trial. The selector is not block-stable; it oscillates at the granularity of the innermost loop.

E. Random Oracle Assumption and Pipeline Stalling

Under the Random Oracle Assumption, both SHA-256 and SHA-3 behave as ideal mathematical black boxes producing uniform, unpredictable outputs. ARIP preserves this property by nesting the two functions. Because both composite orderings are equally computationally hard to solve, alternating execution paths based on nonce parity introduces no statistical vulnerabilities or shortcuts into the mining lottery.

The primary defense of ARIP, however, relies on Pipeline Stalling Theory. Specialized ASIC hardware achieves extreme hashing throughput by keeping a fixed combinational datapath continuously saturated with identical instructions. By forcing the execution pathway to alternate on every single nonce iteration $(Merkle - Damgard \rightarrow Sponge$ vs $Sponge \rightarrow Merkle - Damgard)$, ARIP introduces a structural hazard that forces the ASIC to continuously flush and reconfigure its internal hardware pipeline. This persistent stalling eliminates the processing efficiency that makes hardware specialization economically viable.

III. METHODOLOGY AND IMPLEMENTATION

This section walks through the system one component at a time. For each component the procedural logic is described first, and the concrete code that realizes it follows immediately afterward.

A. Constructing a Block

The atomic unit of the system is a block. Each block holds a positional index, an arbitrary data payload, a reference to the hash of the block preceding, a creation timestamp, a search counter (the nonce, initialized to zero), and a slot for the digest that will certify it. Instantiating a block simply records these fields, the timestamp is captured at creation, and the digest remains empty until the block is mined.

```
class Block:
    # Initializes a block with its position, payload, previous h
    def __init__(self, index, data, previous_hash, algorithm):
        self.index = index
        self.data = data
        self.previous_hash = previous_hash
        self.algorithm = algorithm
        self.timestamp = time.time()
        self.nonce = 0
        self.hash = None
```

Figure 3.1 Class Block Init

B. Computing the Digest

To be hashed, the block's fields are concatenated in a fixed canonical order into a single message string of

$$M = index \,||\, data \,||\, previous\ hash \,||\, timestamp \,||\, nonce$$

Under the interleaving protocol, the digest is instead the result of a two-stage combination with the order selected by the parity of the nonce.

$$H(M,\text{nonce}) = \begin{cases} \text{SHA3-256}(\text{SHA256}(M)), & \text{nonce} \equiv 0 \pmod 2, \\ \text{SHA256}(\text{SHA3-256}(M)), & \text{nonce} \equiv 1 \pmod 2. \end{cases}$$

Crucially, the data handed over from the first step to the second step is the readable hexadecimal text of the middle hash rather than its raw binary computer bytes. This means the bridge connecting the two different hashing designs is a specific 64-character text string. Passing text instead of raw bytes creates a major physical bottleneck for specialized hardware, because this exact text hand-off point forces an industrial ASIC chip's rigid electronic assembly line to completely stall and reconfigure its processing pipeline.

```python
# Builds the block string and returns its hex digest using the baseline or ARIP interleaving rule.
def compute_hash(self):
    block_string = f"{self.index}{self.data}{self.previous_hash}{self.timestamp}{self.nonce}"
    if self.algorithm == "sha256":
        return hashlib.sha256(block_string.encode()).hexdigest()
    if self.algorithm == "sha3_256":
        return hashlib.sha3_256(block_string.encode()).hexdigest()
    if self.nonce % 2 == 0:
        return hashlib.sha3_256(hashlib.sha256(block_string.encode()).hexdigest().encode()).hexdigest()
    return hashlib.sha256(hashlib.sha3_256(block_string.encode()).hexdigest().encode()).hexdigest()
```

Figure 3.2 Compute Hash Function

C. *Mining a Block*

Standard proof of work involves a sequential search for a counter value, or nonce, that forces a block's cryptographic hash to meet a specific difficulty target. This target is defined by a required number of leading zero characters. Miners begin testing at zero and increment the counter by exactly one after each failure until they generate a hash that matches the required zeros. For the interleaving protocol, this steady step by step increment means the counter constantly flips between odd and even values, forcing the system to alternate strictly between its two internal structural configurations throughout the mining process.

The expected number of attempts required to satisfy a target of d leading zero characters is 16^d for all three protocols, since the target is identical and under the Random Oracle Assumption, each construction yields uniform digests. The protocols differ only in the cost per attempt, the baselines perform a single hash evaluation, whereas the interleaving protocol performs two chained evaluations on every trial. This per-attempt asymmetry is the empirical signature analyzed in *IV*.

```python
# Performs the Proof-of-Work by incrementing the nonce u
def mine_block(self, difficulty):
    target = "0" * difficulty
    while not self.compute_hash().startswith(target):
        self.nonce += 1
    self.hash = self.compute_hash()
    return self.hash
```

Figure 3.3 Mine Block Function

D. *Linking the Chain*

A chain is initialized for a fixed difficulty and protocol and is seeded with a first, self-referential genesis block that is mined immediately. Every subsequent block is created with its predecessor's sealed digest as its previous hash reference, mined to satisfy the difficulty target, and appended to the chain. The chained reference is what makes the structure tamper-evident, altering any earlier block changes its digest and breaks every reference that follows.

```python
class Blockchain:
    # Sets up the chain with its difficulty and algorithm, then creates the genesis block.
    def __init__(self, difficulty, algorithm):
        self.difficulty = difficulty
        self.algorithm = algorithm
        self.chain = [self.create_genesis_block()]

    # Builds and mines the first block of the chain with a fixed previous hash.
    def create_genesis_block(self):
        genesis = Block(0, "Genesis Block", "0", self.algorithm)
        genesis.mine_block(self.difficulty)
        return genesis

    # Creates the next block linked to the last block, mines it, appends it, and returns it.
    def add_block(self, data):
        previous_block = self.chain[-1]
        new_block = Block(len(self.chain), data, previous_block.hash, self.algorithm)
        new_block.mine_block(self.difficulty)
        self.chain.append(new_block)
        return new_block
```

Figure 3.4 Class Blockchain

E. *Benchmarking*

Each protocol and difficulty configuration is evaluated by measuring its total mining time and cumulative CPU usage. The process begins by initializing a new blockchain instance to record baseline processor activity. The system then mines five consecutive blocks, using a high-precision timer to track the exact elapsed time. Once complete, the total accumulated CPU consumption is recorded. This methodology effectively yields two primary performance metrics: overall mining latency and the system's computational footprint.

The experiment evaluates every combination of the five difficulty levels and three protocols in a strict, one-by-one sequence. To prevent background resource competition from distorting the timing data, the system avoids parallel processing entirely. Each unique configuration produces a single row of performance metrics, forming the complete dataset analyzed in Section IV.

```python
# Mines 5 blocks at a given difficulty and returns the elapsed time and average CPU usage.
def benchmark(algorithm, difficulty):
    chain = Blockchain(difficulty, algorithm)
    psutil.cpu_percent(interval=None)
    start = time.perf_counter()
    for i in range(5):
        chain.add_block(f"Block {i}")
    elapsed = time.perf_counter() - start
    cpu = psutil.cpu_percent(interval=None)
    return elapsed, cpu


# Runs the benchmark for all three protocols across difficulty levels 1 to 5 and collects the metrics.
def run_benchmarks():
    records = []
    for difficulty in range(1, 6):
        for algorithm in ["sha256", "sha3_256", "arip"]:
            elapsed, cpu = benchmark(algorithm, difficulty)
            records.append({"Difficulty": difficulty, "Algorithm": algorithm, "Time": elapsed, "CPU": cpu})
    return records
```

Figure 3.5 Benchmarking

IV. RESULTS AND ANALYSIS

The ARIP evaluation is done along two orthogonal axes, macro-level throughput (how long the lottery takes to win), and micro-level hardware footprint (how heavily the host processor is loaded). The interpretation is deliberately two-pronged because ASIC resistance lives in the gap between algorithmic cost and the hardware’s ability to amortize it.

A. Macro-Level Throughput Evaluation

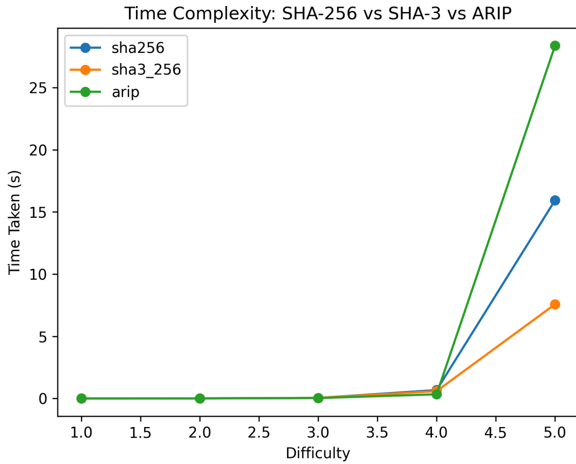


Figure 4.1 Dynamic Time Complexity between SHA-256, SHA-3, and ARIP

The latency profile is reported in Table I. For difficulties one through three, the absolute times are sub-50-millisecond and dominated by interpreter and instrumentation overhead, so the three curves are effectively indistinguishable. The protocols separate sharply only as the search space explodes.

Table I. Mining Latency (seconds) per difficulty

Difficulty	SHA-256	SHA-3	ARIP
1	0.000104	0.000192	0.000196
2	0.002136	0.001206	0.002871
3	0.039100	0.040712	0.029568
4	0.672498	0.588082	0.331265
5	15.943559	7.577006	28.383034

At difficulty five, where the expected attempt count is largest and the per-attempt cost dominates, the structural story emerges clearly, ARIP requires 28.38 seconds, against 15.94 for SHA-256 and 7.58 s for SHA-3. ARIP is the slowest protocol and by a wide margin. This is the expected and desired outcome. ARIP performs two full hash evaluations per nonce trial. Its per-attempt work is therefore approximately the sum of a SHA-256 and a SHA-3 invocation. The observed 28.38 is somewhat equal to 15.94 + 7.58 + the epsilon in the relationship, being the branch-evaluation and string re-encoding overhead at the construction boundary.

The lower-difficulty has somewhat of an anomaly, especially for ARIP where it's faster 0.331 s at difficulty four. These are artifacts of single-sample variance not evidence of an efficiency advantage. Because each block embeds a live timestamp, the number of nonces searched differs run-to-run, and at low attempt counts that variance dwarfs the deterministic per-attempt cost. A re-execution of the harness would shift the figures, but it cannot reorder the protocols at high difficulty, where ARIP's roughly 2x per-attempt cost is structural, fixed by the fact that it always runs both engines. The latency is the protocol working as intended.

B. Micro-Level Hardware Footprint Analysis

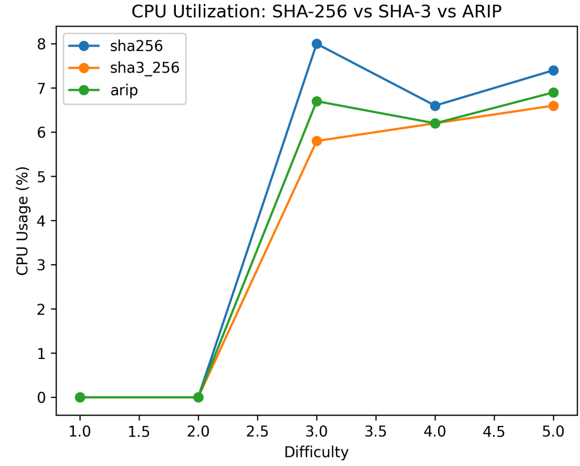


Figure 4.2 System CPU Utilization Versus Difficulty for the Three Protocols

The processor-footprint profile is reported in Table II.

Table II. System CPU Utilization (%) per difficulty

Difficulty	SHA-256	SHA-3	ARIP
1	0.0	0.0	0.0
2	0.0	0.0	0.0
3	8.0	5.8	6.7
4	6.6	6.2	6.2
5	7.4	6.6	6.9

At difficulties one and two, the workload completes faster than the sampling resolution of the utilization probe, yielding a floored reading of 0% for all three protocols. From difficulty three onwards the readings stabilize, and the salient observation is how *tightly clustered* the three footprints are. Every protocol sits within a narrow 5.8% - 8.0% band, and at difficulty five the spread between the lightest and the heaviest is well under 2%.

This near-parity is very much significant. ARIP's enormous latency premium does not translate into a proportionally enormous CPU footprint, because the harness is single-threaded and the bottleneck is serial hash evaluation rather than parallel core. On commodity hardware, ARIP

looks ordinary, its CPU bound serial workload is indistinguishable, in resource-occupancy terms, from running two stock hash functions back to back. There is no exotic memory pressure or specialized resource demand that would advantage one class of machine over another.

C. Why This Structure Thwarts ASICs

The two analyses combine into the core resistance argument. The throughput data establish that ARIP's cost is additive across two architecturally disjoint constructions. The footprint data establish that this cost is borne by a perfectly ordinary serial CPU. An ASIC designer confronting ARIP cannot replicate the SHA-256 playbook of unrolling a single compression pipeline, for three reasons:

1. Datapath divergence. Since winning the lottery requires both a Merkle-Damgard arithmetic engine and a Keccak permutation engine on the same side, these share essentially no reusable structure, silicon area and power must be spent on both, halving the efficiency relative to a single-function ASIC before.
2. No idle-circuit multiplexing. Unlike function-selector multi-hash schemes, ARIP exercises both engines on every attempt. Neither circuit can be clock-gated or powered down to save energy, the worst case is the only case.
3. Branch unpredictability at loop granularity. Because the composite ordering flips with nonce parity on every iteration, the inter-engine routing reverses direction each trial. A pipelined datapath cannot stay saturated across the construction boundary, it must serialize the hand-off between two opposite orderings, incurring exactly the pipeline-stall penalty described before. The amortization that makes single-function ASICs economical is structurally unavailable.

The result is that ARIP narrows the gap between specialized and commodity hardware not by making the function slower in the abstract, but by making the specialization itself unprofitable, precisely the consensus layer cryptographic agility posited.

V. CONCLUSION

This paper introduced ARIP, a Proof-of-Work consensus extension that dynamically interleaves SHA-256 and SHA-3 according to nonce parity, and benchmarked it against both baselines across difficulty levels one through five. The empirical findings are unambiguous and, we argue, favorable since ARIP imposes a structural per-attempt cost approximately equal to the sum of its two constituent hashes while maintaining a commodity-CPU footprint statistically indistinguishable from the baselines. We have argued that this latency premium is not an inefficiency to be apologized for but the very mechanism of ASIC disruption, by forcing both an architecturally divergent Merkle–Damgård engine and a sponge engine onto any prospective miner, and by reversing their composite ordering on every nonce increment, ARIP denies fixed-function silicon the pipelined amortization that underwrites the ASIC oligopoly.

Several avenues remain for strengthening the argument from a software demonstration to a hardware guarantee:

1. Hardware-level HDL testing. The decisive claim is architectural and therefore belongs in silicon. Implementing ARIP's parity router and synthesizing it for an ASIC device to try on would quantify the real pipeline-stall penalty, the combined die area of the dual engines, and the energy-per-hash gap versus a single-function reference.
2. Statistical rigor. The present figures are single samples per configuration. Repeating each run many times and reporting means with confidence intervals would eliminate the low-difficulty variance artifacts observed in Table I.

ARIP demonstrates that ASIC resistance can be reframed from a search for one unbuildable function into a discipline of cryptographic agility at the consensus layer, making the target move faster than silicon can be specialized to catch it.

Code Link at Github

https://github.com/PeterWongsoredjo/sha-256vs-3onblockchain

Video Link at Youtube

https://youtu.be/tmOiHsvdpFE

Acknowledgment

The author would like to express utmost gratitude to Mr. Dr. Ir. Rinaldi Munir, M.T. as the lecturer of II4021 Kriptografi for his valuable guidance and passion to spread knowledge, which in turn contributes greatly to the completion of this paper. The author would also like to thank all his friends and family members for their moral support throughout the process of researching and writing this paper.

References

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008. https://bitcoin.org/bitcoin.pdf (Accessed 17 June 2026)

[2] National Institute of Standards and Technology, FIPS PUB 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, 2015. Reference description available at: https://grokipedia.com/page/SHA-3 (Accessed 17 June 2026)

[3] National Institute of Standards and Technology, FIPS PUB 180-4: Secure Hash Standard (SHS), 2015. Initial public comments: https://csrc.nist.gov/csrc/media/Projects/crypto-publication-review-project/documents/initial-comments/fips180-4-initial-public-comments-2022.pdf (Accessed 17 June 2026)

[4] "ASIC-Resistance of Multi-Hash Proof-of-Work Mechanisms for Blockchain Consensus Protocols," ResearchGate publication, 2018: https://www.researchgate.net/publication/328644921_ASIC-Resistance_of_Multi-Hash_Proof-of-Work_Mechanisms_for_Blockchain_Consensus_Protocols (Accessed 17 June 2026)

[5] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Keccak sponge function family main document," 2011. https://keccak.team/obsolete/Keccak-main-2.0.pdf (Accessed 17 June 2026)

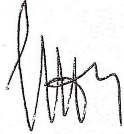
[6] https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi-dan-Koding/2025-2026/40-Penggunaan-kriptografi-di-dalam-blockchain-2026.pdf (Accessed 17 June 2026)

[7] https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi-dan-Koding/2025-2026/27-Beberapa-fungsi-hash-2026.pdf (Accessed 17 June 2026)

PERNYATAAN

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

Bandung, 18 Juni 2026



Peter Wongsoredjo 13523039