SPARE: Asymmetric Proof-of-Work–Based DoS Mitigation for IoT Devices

Amirhosein Yari, Majid Khabbazian

***Abstract*—Battery-powered IoT nodes are vulnerable to *signature-verification* denial-of-service (DoS): an adversary can flood a device with well-formed but invalid messages, forcing expensive digital-signature verifications. A standard mitigation is to require proof-of-work (PoW) per message, but this *symmetrically* burdens honest and dishonest senders alike. We address this shortcoming with an *asymmetric* PoW-based mitigation that makes attackers pay while sparing honest parties. A designated Bitcoin miner embeds a request commitment in its coinbase transaction; any non-winning block header whose hash falls below an application-chosen threshold serves as PoW. The sender appends this header and a logarithmic-size Merkle proof; the IoT device first validates this PoW—just a handful of hash evaluations—before attempting the costly signature verification. Because proofs are bound to the miner's payout address, adversaries cannot piggy-back on recycled work: they must grind fresh headers (or effectively mine on the designated miner's behalf), preserving a large resource gap in the defender's favor. We prototype the scheme on an ESP32 MCU and show that PoW verification takes 0.8 ms versus 260 ms for ECDSA ($>300\times$ speed-up); proof packages remain <1 kB, and end-to-end latency is dominated by network RTT even with a moderate-capacity miner; power measurements likewise confirm that hash-based verifications cost orders of magnitude less energy than signatures. The mechanism requires no blockchain modifications, scales to thousands of devices per miner, and immediately hardens firmware updates, certificate rotation, and other unsolicited IoT traffic against signature-verification DoS attacks.**

***Index Terms*—DoS Mitigation, Proof-of-Work, Blockchain, IoT**

I. Introduction

Many IoT deployments—from smart-meter networks to agricultural sensor grids—rely on *externally initiated* commands for firmware updates, configuration changes, or actuation. Because the device does not control (or fully trust) the first hop, it must authenticate every inbound request before executing it. Digital signatures (such as RSA [22] and ECDSA [14]) are the de-facto mechanism: they provide authenticity and non-repudiation while avoiding the key-distribution problems of symmetric schemes, and they enable straightforward credential revocation in large, dynamic, or one-to-many environments.

The signature-verification DoS attack. Signature verification remains expensive on low-power microcontrollers (MCUs). For example, on a common ESP32 MCU, verifying a single ECDSA signature is orders of magnitude slower than computing a SHA-256 hash. An adversary can exploit this cost by flooding the device with syntactically correct but invalid messages as shown in Figure 1, forcing repeated expensive signature verifications that can saturate the CPU and rapidly drain the battery—an attack documented in early sensor-network studies [15] and later observed in TLS handshake flooding [21].

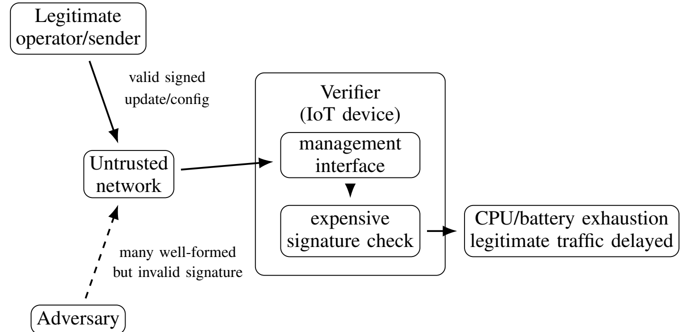

Fig. 1. Signature-verification DoS attack setting. A constrained IoT device exposes a management interface for legitimate signed commands, such as updates or configuration changes. An adversary floods the same interface with syntactically well-formed but invalid signatures, forcing repeated expensive signature verifications before rejection. This can exhaust CPU/energy resources and delay legitimate traffic.

In typical IoT deployments, signature verification is not the common-case per-packet protection mechanism; after enrollment or session establishment, high-volume traffic is usually protected with symmetric keys or MACs. The signature-verification DoS risk arises because, in modern systems, constrained devices may still need to keep a management interface reachable before a low-cost shared-key authenticator or trusted gateway can filter requests—for example, to support configuration changes, software/firmware updates, key or certificate rotation, and signed update manifests [7], [17]. Thus, legitimate signature checks may be infrequent, but the verification endpoint remains a persistent attack surface that an adversary can repeatedly invoke with well-formed but invalid signed objects.

Some deployments introduce software rate limiters: the device processes inbound traffic at full speed until it encounters an invalid signature, then enters a “cool-down” window—for example, accepting at most k messages per minute or sleeping for t seconds before resuming normal service. This caps the number of costly verifications, but it is easy to trigger: an attacker can intentionally trip the limiter with only a handful of forged messages, forcing the device into its own back-off state and delaying or dropping legitimate requests. The result is an unavoidable availability trade-off.

Our approach: recycle discarded PoW. A classic mitigation against resource-exhaustion attacks is to require each sender to

Amirhosein Yari and Majid Khabbazian are with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, Canada (e-mail: ayari1@ualberta.ca ; mkhabbazian@ualberta.ca).

solve a proof-of-work (PoW) puzzle per message. The verifier checks the PoW cheaply and attempts signature verification only if the PoW passes, hence an adversary must expend substantial computation to trigger each expensive signature check. However, because the verifier cannot distinguish honest from malicious senders *a priori*, this approach is *symmetric*: honest senders must also pay the same per-message PoW cost as attackers.

SPARE avoids this blanket client-side PoW penalty by reusing work that miners already perform. Miners on PoW blockchains—with Bitcoin as our concrete instantiation—evaluate vast numbers of *non-winning* block headers while searching for a valid block; these headers are normally discarded because they do not satisfy the network difficulty target. In SPARE, one or more *designated miners* embed a commitment to a sender's request in the coinbase transaction of the block template they are already hashing. Any non-winning header whose hash falls below an application-chosen threshold then serves as a reusable PoW stamp for that request. The miner returns the qualifying header together with a compact Merkle inclusion proof, and the sender attaches this proof package to its signed message. Upon receipt, the IoT verifier validates the proof package using only a handful of fast hash computations, and performs the costly signature verification only after the proof check succeeds.

Two features make SPARE practical. First, SPARE does not burden miners: embedding a short commitment in the coinbase does not alter the mining loop, and the PoW evidence is drawn from headers the miner would compute anyway. Second, SPARE does not require trusting miners: it never replaces end-to-end signatures, thus a message is accepted only if the sender's signature verifies. Moreover, SPARE includes safeguards against miner and client abuse, including a mechanism that yields publicly verifiable evidence if a miner issues a proof for a request with an invalid signature, enabling policy-driven removal of misbehaving miners.

In summary, we make the following contributions.

- We introduce the first DoS mitigation scheme that *shifts the solving cost off honest senders while preserving the adversary's PoW cost*, by reusing non-winning Bitcoin block headers obtained from a designated miner (no blockchain modifications).
- We formalize the threat model and prove that forging a valid proof is as hard as finding a header whose hash falls below an application-chosen threshold (bound to the designated miner's coinbase), and we quantify the resulting computational asymmetry.
- We design and evaluate three practical extensions—concurrent-request batching, verifiable proofs of miner misbehavior, and lightweight economic/rate-limit safeguards—that harden the system against misbehaving miners and abusive clients while preserving sub-kilobyte proofs and logarithmic verifier cost.
- We implement the protocol on an ESP32 MCU and show that PoW validation completes in $0.8\,\mathrm{ms}$, whereas ECDSA verification takes $260\,\mathrm{ms}$ on the same platform ($> 300\times$ speed-up). Even with a moderate-capacity miner, end-to-end latency is dominated by sender–miner round-trip time (RTT), and the proof adds $< 1\,\mathrm{kB}$ per message.

II. Background

Bitcoin mining and block construction. Bitcoin is a decentralized cryptocurrency that maintains a public ledger, known as the blockchain, where transactions are grouped into blocks. Each block contains a block header and a list of transactions. The block header includes, among other fields, a Merkle root (a hash summarizing all transactions in the block), and a nonce. To add a block to the chain, miners must find a nonce such that the *double-SHA-256* hash of the block header (i.e., SHA-256 applied twice) is at most the network-defined difficulty target T_{btc}. This process, known as PoW, requires repeated hashing of the block header with different nonce values until a valid one is found. Due to the probabilistic nature of mining, miners generate many block headers whose hashes do not satisfy T_{btc} and are thus discarded.

The coinbase transaction and data embedding. The coinbase transaction is the first transaction in every Bitcoin block and is created by the miner; unlike regular transactions it has no inputs and mints the block reward (and collects fees). Its sole input's script (*scriptSig*) may contain miner-defined bytes that are not interpreted by consensus validation, and miners routinely embed identifiers (e.g., pool names), making the coinbase a natural place for auxiliary metadata [6]. Small payloads can be placed either in the coinbase input's `scriptSig` or in an `OP_RETURN` output; in either case the data becomes part of the transaction and thus part of the block's transaction Merkle tree.

Because data embedded in the coinbase becomes part of a Bitcoin transaction, it is also committed to by the block's transaction Merkle root, which is recorded in the block header. Thus, coinbase-embedded metadata can be cryptographically tied to a block header without changing Bitcoin's mining process or consensus rules.

Merkle trees and inclusion proofs. In each Bitcoin block, all transactions—including the coinbase—are arranged in a binary Merkle tree. The 32-byte Merkle root is recorded in the block header and cryptographically commits to the exact multiset and order of transactions. We write $H(\cdot)$ for SHA-256 and $H_2(\cdot) := H(H(\cdot))$ for Bitcoin's double-SHA-256 (used for transaction identifiers and block-header hashing). To prove that a particular transaction tx is included in a block with header B, one supplies a Merkle *inclusion proof*: the sequence of sibling hashes along the path from the transaction identifier $H_2(tx)$ to the root, together with the left/right concatenation directions. The verifier recomputes the root using a handful of hash evaluations (time $O(\log n)$ and proof size $O(\log n)$ for n transactions) and checks that it matches the Merkle root in B. This lets a lightweight verifier authenticate coinbase-embedded metadata without downloading the full block.

Difficulty targets and non-winning headers. Bitcoin accepts only blocks whose block-header hash, interpreted as an integer, is at most the network target T_{btc}. During mining, however, a miner explores a vast number of non-winning headers whose hashes exceed T_{btc} and are therefore discarded by Bitcoin

consensus. More generally, for any relaxed target $T > T_{\mathsf{btc}}$, a discarded header with hash at most T is not a valid Bitcoin block, but it still represents evidence that the corresponding header search succeeded at difficulty T. This distinction between Bitcoin's network target and an application-chosen target is used later in the SPARE protocol.

III. System and Threat Models

We consider a resource-constrained verifier $\mathcal{V}$ (e.g., an IoT device) that must authenticate externally initiated management messages such as firmware-update manifests, certificates, configuration changes, or signed control commands. A sender attaches a SPARE proof package to each signed message, and $\mathcal{V}$ checks this package before attempting the expensive signature verification.

Active participants. The active protocol entities are as follows.

- **Sender $\mathcal{S}$:** holds signing key sk_S and public key pk_S, computes $h = H(m)$ for message m, and produces $\sigma_S = \mathsf{Sign}_{\mathsf{sk}_S}(h)$.
- **Designated miner $\mathcal{M}$:** an allow-listed miner $\mathcal{M} \in \mathbf{M}_{\mathsf{desg}}$ that verifies σ_S, embeds the request commitment $c = H(h \| \mathsf{pk}_S \| \sigma_S)$ in a coinbase transaction tx_{cb}, and returns a proof package.
- **Verifier $\mathcal{V}$:** the constrained IoT device that receives (m, σ_S, π), validates the proof package π first, and verifies σ_S only if the lightweight checks pass.
- **Adversary $\mathcal{A}$:** controls the network and may operate arbitrary senders and miners, as detailed below.

The Bitcoin blockchain is used as an enabling PoW and commitment technology.

Trust and State

1) $\mathcal{V}$ stores an authentic copy of pk_S (or a certificate chain anchored in a trusted root).
2) $\mathcal{V}$ stores an allow-list $\mathbf{M}_{\mathrm{desg}}$ of designated miners, including (at minimum) a miner identifier used for binding proofs to a designated miner (e.g., a payout address or coinbase-identifying data). When accountability is enabled, $\mathcal{V}$ additionally stores an associated miner attestation key (or key hash) that allows it to validate a short, publicly verifiable proof of miner misbehavior and to remove the miner from $\mathbf{M}_{\mathrm{desg}}$ (Section VII-B). Regardless of miner behavior, $\mathcal{V}$ accepts a message only if the sender signature verifies under pk_S.
3) Cryptographic primitives (a cryptographic hash function and a digital-signature scheme) provide standard security (preimage/collision resistance for the hash function, and existential unforgeability under chosen-message attacks for signatures).

Adversary Capabilities

- *Network control (safety vs. efficiency).* To prove safety, $\mathcal{A}$ can inject, delay, replay, reorder, or drop packets between any parties. Under such full censorship, *liveness/efficiency cannot be guaranteed* (e.g., $\mathcal{A}$ may indefinitely suppress proof packages and force signature-only fallback). Accordingly, our *efficiency* claims additionally assume a mild availability condition: for an honest request, at least one designated miner contacted by the sender is reachable and a corresponding proof package is delivered to the verifier within the application timeout.
- *Computation.* $\mathcal{A}$ is polynomial-time; breaking SHA-256 or forging digital signatures is infeasible.
- *Miner queries.* $\mathcal{A}$ may submit embedding/proof requests to any miner, including miners in $\mathbf{M}_{\mathrm{desg}}$. However, this does not let $\mathcal{A}$ obtain accepted proof packages "like an authorized sender": (i) proofs are accepted only if the coinbase binds them to an identity in $\mathbf{M}_{\mathrm{desg}}$, thus a non-designated miner must mine while crediting a designated identity (i.e., effectively mine on its behalf); (ii) an honest designated miner returns proofs only after checking the sender signature as in the protocol (Section IV); and (iii) if a designated miner violates this check and returns a proof for an invalid sender signature, Section VII-B provides a publicly verifiable, unframeable mechanism to detect the behavior and support policy-driven removal from $\mathbf{M}_{\mathrm{desg}}$.

Adversary Goals

$\mathcal{A}$ aims to deplete $\mathcal{V}$'s resources by forcing expensive signature verifications, or to delay/deny service to honest traffic. Concretely, $\mathcal{A}$ attempts to:

1) trigger repeated signature verifications on ultimately invalid messages; or
2) inject sufficient bogus traffic to delay or crowd out legitimate messages.

We do not address lower-layer flooding or physical jamming, which are orthogonal.

Security Goal

Safety under full network control. A primary goal is *safety*: even under the strongest adversary in our threat model (including arbitrary packet delay/drop and miner collusion), the verifier accepts a message *only* if the sender's signature verifies under an allow-listed sender key. The PoW proof package is a pre-authentication *filter* that affects only the verifier's *resource expenditure*; it never replaces the digital signature. Therefore, a network adversary can at worst (i) force the system into its signature-only fallback by withholding proof packages, or (ii) deny service by censoring traffic, but it cannot cause acceptance of an unauthenticated message.

Efficiency Goal

Efficiency under timely proof availability. Our efficiency objective is conditional on a mild, deployment-standard *availability* assumption: for an honest request, at least one designated miner contacted by the sender is reachable and a corresponding proof package is delivered to the verifier within the application's timeout. Under this condition, an honest sender incurs $O(1)$ work per message (one signature plus a fixed-size request to miners), the verifier rejects invalid traffic using only cheap hash-based validation before any signature

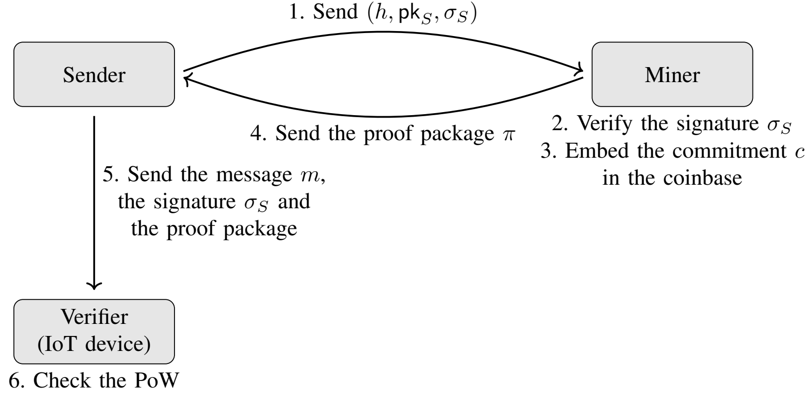

Fig. 2. Protocol flow: (1) Sender sends the message hash and signature; (2) Miner checks the digital signature; (3) Miner embeds the request commitment c in the coinbase and searches for candidate headers satisfying $H_2(B) \le T_{\mathsf{app}}$; (4) Miner sends a proof package π; (5) Sender forwards the message, signature, and proof package to the verifier; (6) Verifier validates the proof package first, then verifies the sender's signature.

verification, and an adversary must still expend expected $\Omega(1/p)$ header trials per attempt to produce an accepted proof, where $p = T_{\mathsf{app}}/2^{256}$. If the availability assumption does not hold (e.g., an adversary indefinitely censors all proofs), the protocol degrades to ordinary signature verification—*preserving safety but not the DoS-cost asymmetry*.

IV. Protocol Description

In this section, we describe our asymmetric PoW-based mitigation protocol. The protocol enables a sender to convince a verifier (e.g., an IoT device) that computational effort has been expended on behalf of a message, without requiring an honest sender to perform the work themselves: the construction recycles discarded PoW results generated by blockchain miners, allowing honest senders to avoid computational burden while forcing adversaries to expend significant effort.

Protocol steps. Figure 2 illustrates the flow among the sender, designated miner, blockchain data structures, and verifier. The protocol proceeds as follows:

1) **Hash and sign.** The sender computes $h = H(m)$ for the message m and produces a digital signature $\sigma_S = \mathrm{Sign}_{\mathsf{sk}_S}(h)$. Then it transmits $(h, \mathsf{pk}_S, \sigma_S)$ (and policy metadata such as T_{app} or a timeout) to one or more *designated miners*.
2) **Embedding request.** The miner verifies σ_S under pk_S to prevent abuse. The miner learns only h (not m), preserving message privacy.
3) **Coinbase commitment.** After verifying σ_S, the miner adds $c = H(h \| \mathsf{pk}_S \| \sigma_S)$ in the coinbase transaction of the block template it is already assembling.[1] Because the coinbase field is miner-defined, this update can be applied on the fly; the miner continues its nonce search without discarding the template. As Bitcoin's PoW is *memoryless*—each header hash is an independent trial—refreshing the coinbase to commit to c neither reduces the success probability nor "wastes" prior work.
4) **Header search at an application threshold.** The miner continues hashing headers until it encounters a block header B whose double-SHA-256 hash value is at most the application threshold T_{app} (with $T_{\mathsf{app}} \ge T_{\mathsf{btc}}$ when interpreted as integers). This threshold is much easier than the Bitcoin network target and yields practical discovery rates.
5) **Proof-package construction.** Upon finding such a header, the miner returns to the sender a *proof package* π consisting of:
   - the complete block header B;
   - the full coinbase transaction tx_{cb} that contains the request commitment c;
   - a Merkle inclusion path proving tx_{cb} is in the block's transaction Merkle root;

   After exporting the package, the miner removes the request commitment from its working template.
6) **Forward to the verifier.** The sender forwards to the verifier: (i) the message m, (ii) the sender's signature σ_S, and (iii) the proof package.
7) **Verifier checks (cheap first, expensive last).** On receipt, the verifier executes the following lightweight tests before any signature verification:
   - Confirm that pk_S is on its list of trusted keys.
   - *Header difficulty:* recompute the double-SHA-256 header hash $H_2(B)$ and check that it is $\le T_{\mathsf{app}}$.
   - *Miner binding:* confirm that tx_{cb} pays to (or otherwise identifies) a designated miner.
   - *Merkle inclusion:* verify the Merkle path from tx_{cb} to the block's Merkle root in B.
   - *Message match:* compute $h = H(m)$ and check that $c = H(h \| \mathsf{pk}_S \| \sigma_S)$.

Only after all lightweight checks succeed does the verifier attempt the (expensive) verification of σ_S under pk_S. The message is accepted as authentic only if *every* check succeeds.

The *timeout* mentioned in Step 1 is an application-level deadline used by the sender to decide whether to retry or fall back. Under our network adversary model, failure to receive a proof package before the timeout is not attributable

[1] If multiple requests are pending, the miner maintains a Merkle tree over the set of $\{c_i\}$ and embeds only the tree root $R = \mathrm{MerkleRoot}(\{c_i\})$ in the coinbase; the proof package then includes a short path from c to R.

to miner behavior (it may be caused by network delay/drop or partitions), and therefore does not by itself justify miner eviction or penalties. Miners are removed from the allow-list only upon publicly verifiable evidence of misbehavior (Sec. VII-B). To improve availability, a sender may query multiple designated miners (in parallel for latency-sensitive messages, or sequentially upon timeout) and forward the first valid proof received.

V. Security Analysis

We now provide a formal analysis of the security properties of the proposed protocol under the adversary model defined in Section III. Our goal is to show that the protocol ensures message authenticity and computational soundness of the PoW.

Roadmap. The analysis separates correctness from cost. Proposition 1 shows that SPARE does not weaken authenticity: the proof package is only a pre-authentication filter, and the sender's signature remains the final acceptance condition. Proposition 2 shows that passing this filter is not free: an accepted proof must either come from a designated miner or require fresh header grinding under T_{app} while being bound to a designated miner identity. Proposition 3 then quantifies the resulting DoS asymmetry: an adversary must spend expected $\Omega(1/p)$ hash trials per accepted proof attempt, whereas the verifier spends only $O(\log n)$ hash computations before deciding whether to perform signature verification. Thus, the security argument is that SPARE preserves ordinary signature-based authenticity while making repeated attempts to reach the expensive verification path costly for the adversary.

Authenticity. The PoW-based pre-authentication filter is *only* a cost-shifting mechanism: it determines which messages are *worth* spending signature-verification effort on, but it never replaces the sender's digital signature. In particular, even if an adversary obtains a proof package that passes all PoW checks, the verifier still accepts a message only if the sender signature verifies under the sender's public key. Therefore, under standard signature security, an adversary cannot cause acceptance of an unauthenticated message; the filter can at most affect resource usage, not correctness. The following proposition formalizes this authenticity guarantee.

Proposition 1 (Authenticity)**.** *Let $\mathcal{A}$ be any probabilistic polynomial-time adversary that can interact with the verifier according to the protocol. Then the probability that $\mathcal{A}$ causes the verifier to accept a tuple (m, σ_S, π) such that σ_S was* not *generated by the legitimate sender is negligible in the security parameter λ:*

$$\Pr\big[\, \mathit{Accept}(m, \sigma_S, \pi) \wedge \sigma_S \not\leftarrow \mathsf{Sign}_{sk_S}(H(m)) \big] \;\leq\; \mathsf{negl}(\lambda). \tag{1}$$

Here Accept denotes the verifier's acceptance predicate defined by the protocol.

Proof. By construction, the verifier accepts (m, σ_S, π) only if *all* of the following hold:

(a) the proof package π passes the PoW checks;
(b) the block's coinbase transaction in π is associated with a designated miner;
(c) $\mathsf{Verify}(\mathsf{pk}_S, H(m), \sigma_S) = \mathsf{true}$, where pk_S is the sender's public key.

Conditions (a) and (b) are independent of the signature σ_S. Consequently, an adversary may satisfy them without knowledge of the sender's secret key. However, to satisfy (c) the adversary must output a signature σ_S that verifies under pk_S on the message digest $H(m)$. Under the security assumption of cryptographic primitives, the probability that a polynomial-time adversary can produce such a signature without access to the sender's secret key is negligible in λ. Hence the probability that the verifier accepts a message whose signature was not created by the legitimate sender is also negligible. □

PoW Binding. The PoW check is designed to be both *lightweight* for the verifier and *hard to forge* for an adversary. At a high level, passing the filter requires presenting a block header whose hash falls below the application threshold T_{app} *and* a Merkle-authenticated coinbase commitment that binds the proof to an allow-listed miner identity. An adversary therefore has only two options: either obtain a genuine proof package from a designated miner, or perform its own grinding to find a suitable header while embedding an allow-listed miner identity (which effectively amounts to mining on that miner's behalf). The following proposition formalizes this binding property and rules out "free" forgery of valid PoW packages.

Proposition 2 (PoW Binding)**.** *Let $\mathcal{A}$ be any probabilistic polynomial-time adversary. Define* $\textsc{Accept}_{\mathrm{PoW}}(h, \pi) = 1$ *iff the verifier's* PoW check alone *approves π for h. Then, except with negligible probability, $\mathcal{A}$ can achieve* $\textsc{Accept}_{\mathrm{PoW}}(h, \pi) = 1$ *only by*

1) *obtaining a valid proof package that was* actually produced *by a miner whose identity is in $\mathbf{M}_{\mathrm{desg}}$, or*
2) *independently performing hashing work sufficient to find a block header with $H_2(B) \leq T_{\mathrm{app}}$ while embedding in the coinbase an identity from $\mathbf{M}_{\mathrm{desg}}$.*

Assume the verifier discards previously accepted proof packages, thus replay is ineffective.

Then, any accepted proof package arises either from a designated miner's participation or from work at least as hard as finding a header under T_{app} while bound to a designated miner.

Proof. A proof package π consists of a block header B, a coinbase transaction tx_{cb} that contains the request commitment c and a miner identity id, and a Merkle proof π_m linking tx_{cb} to the Merkle root in B. Collision resistance of $H(\cdot)$ binds $c = H(h \| \mathsf{pk}_S \| \sigma_S)$ to the request tuple, and π_m binds tx_{cb} to the Merkle root in B.

For $\textsc{Accept}_{\mathrm{PoW}}(h, \pi) = 1$, the verifier checks (i) that $H_2(B) \leq T_{\mathsf{app}}$ and (ii) that $id \in \mathbf{M}_{\mathrm{desg}}$, where $\mathbf{M}_{\mathrm{desg}}$ is the set of designated-miner identities. Because the adversary cannot forge digital signatures under the EUF-CMA assumption, it cannot trick an honest designated miner into embedding an unauthenticated hash. Thus, the only ways to satisfy both checks are either (1) to obtain a genuine proof package from some miner in $\mathbf{M}_{\mathrm{desg}}$, or (2) to produce a valid block header

B with $H_2(B) \leq T_{\mathsf{app}}$ while embedding in the coinbase an identity $id \in \mathbf{M}_{\mathrm{desg}}$.

In case (2), the adversary must perform genuine PoW under the same conditions as an honest miner but without the ability to redeem the block reward, since the coinbase payout must credit a designated miner. Replay of previously accepted proofs is ineffective, because the verifier enforces freshness and discards prior proofs.

Hence, any accepted proof package must correspond either to work performed by a designated miner or to computational effort by the adversary equivalent to finding a block header under T_{app}, thereby preserving the resource asymmetry of the protocol. □

Resource-Asymmetry and DoS Mitigation. The central DoS-mitigation benefit comes from an extreme cost imbalance between *producing* an acceptable PoW package and *verifying* it. To find a header below T_{app} an attacker must spend many hash trials in expectation. In contrast, the verifier performs only a small number of hash computations: two header-hash checks plus a logarithmic number of hashes to verify the Merkle inclusion proof. By forcing this cheap-first verification order, the verifier can discard most unauthenticated traffic without paying the much higher cost of signature verification, while an attacker must invest substantial work to even reach that stage. The following proposition quantifies this asymmetry and shows how it is tunable via T_{app}.

Proposition 3 (Resource-Asymmetry and DoS Mitigation)**.** *Let T_{app} be the application-defined difficulty threshold and let $p = T_{\mathrm{app}}/2^{256}$ denote the probability that a single Bitcoin header trial yields a value below T_{app}. Let n be the number of transactions in the block containing the coinbase. For every proof package π that the verifier accepts,*

1) *the adversary must perform an expected $\Omega(1/p)$ hash evaluations, and*
2) *the verifier performs at most $\mathcal{O}(\log n)$ hash evaluations to validate the Merkle path and header hash.*

Consequently, the attacker's expected computational cost per accepted message exceeds the verifier's cost by a factor of $\Omega(\frac{2^{256}}{T_{\mathrm{app}} \cdot \log n})$, yielding an exponentially large advantage tunable by T_{app}.

Proof. **Adversary cost.** Finding a header satisfying $H_2(B) < T_{\mathsf{app}}$ is a Bernoulli process with success probability p per hash attempt. By the memoryless property of SHA-256 viewed as a random oracle, the expected number of trials before the first success is $\Omega(1/p)$.

Verifier cost. To validate a proof package, the verifier (i) computes one hash of the block header to confirm $H_2(B) < T_{\mathsf{app}}$ and (ii) recomputes at most $\mathcal{O}(\log n)$ intermediate hashes to reconstruct the Merkle root from the coinbase transaction. Hence the total number of hash evaluations is $\mathcal{O}(\log n)$.

Cost ratio. The ratio of expected adversary cost to verifier cost is therefore

$$\Omega(\frac{1/p}{\log n}) = \Omega(\frac{2^{256}}{T_{\mathrm{app}} \cdot \log n}). \tag{2}$$

□

VI. Efficiency Analysis

This section analyzes the efficiency of SPARE along the protocol path in Fig. 2. First, we analyze the overhead imposed on the designated miner, with particular focus on whether serving sender requests interferes with the mining process. We show that it does not: serving a request requires only a signature check and a temporary 32-byte commitment in the coinbase template, while the memoryless header-search loop remains unchanged and the expected on-chain footprint per request is near zero (Section VI-A).

Second, we analyze proof-generation latency, defined as the time from the sender's request until the sender receives a valid proof package. We show that this latency consists of sender–miner RTT, miner-side signature verification, coinbase refresh, and relaxed-threshold header search, and that for practical choices of T_{app} and realistic miner capacities, it is typically dominated by sender–miner RTT (Sections VI-B–VI-C).

Third, we analyze the size of the proof package forwarded to the verifier. We show that, because the package contains only a block header, a coinbase transaction, and a logarithmic Merkle path, it remains below $1\,\mathrm{kB}$ in realistic settings (Section VI-D).

Finally, we analyze the verifier-side cost of the cheap-first validation path. We show that, on an ESP32, proof-package verification takes $\approx 0.84\,\mathrm{ms}$, whereas ECDSA verification takes $\approx 260\,\mathrm{ms}$, yielding a $> 300\times$ verification-time gap that allows invalid traffic without a valid proof to be discarded before the expensive signature check (Section VI-E).

A. The Coinbase Overhead

Embedding a 32-byte commitment value into the coinbase is a one-shot, constant-time tweak to the block template; it neither changes the header-search loop nor the distribution of header hashes. This commitment may be a message hash (baseline), a batched Merkle root (batching), or a request commitment used for accountability (Section VII-B); in all cases it remains fixed-size. Because Bitcoin PoW is *memoryless* (each header hash is an independent trial), refreshing the coinbase to carry a new commitment does not reduce the probability of finding a valid block and does not "waste" prior work. In practice, miners already mutate the coinbase at high frequency while hashing; adding one more 32-byte field is operationally indistinguishable from those routine updates and imposes effectively zero additional CPU load or latency. The miner continues its nonce search without discarding the template.

The on-chain footprint is likewise negligible. The extra data is constant-size and only materializes if the specific template that currently carries the commitment becomes the winning block. Under typical loads a miner cycles through many coinbase templates per second, thus the probability that any given commitment persists until a win is very small; the expected chain data per request is therefore near zero. Even when commitments do appear on chain (e.g., at higher request rates), they add only 32 bytes—roughly 0.003% of a $1\,\mathrm{MB}$ block (and 0.0016% of a $2\,\mathrm{MB}$ block)—far below thresholds that measurably affect block propagation. Moreover, once a proof package is returned to the sender, the miner can immediately

remove the commitment from the working template. In short, the protocol introduces effectively no additional mining cost and only a negligible amortized on-chain overhead, while reusing work the miner would perform anyway.

Example 1. *Suppose a miner receives one sender request and inserts its 32-byte commitment into the coinbase of the current block template. From the miner's perspective, this is only a small template update: the mining loop is unchanged, and each hash trial remains independent of all previous trials. Hence, updating the coinbase does not discard useful progress or reduce the chance of finding a Bitcoin block.*

The commitment is also temporary. Once the miner finds a header that meets the easier SPARE *application threshold, it returns the corresponding proof package to the sender and removes the commitment from its working template. In the common case, that header is not a valid Bitcoin block, so the 32-byte commitment never appears on chain. It appears on chain only in the rare event that the same temporary template also wins the much harder Bitcoin mining race. Thus, the miner-side process remains essentially unchanged, the added data is small, and the expected on-chain footprint per request is negligible.*

B. Latency and Responsiveness

Let τ_{net} be the round-trip time (RTT) between the sender and the miner, and let τ_{sig} denote the miner's signature-verification time (sub-millisecond). Two mining–related delays then dominate:

1) **Coinbase–refresh delay** τ_{cb}. A miner re-templates its coinbase once it exhausts the 2^{32} nonce space of the current header. With hash-rate $\mathcal{R}$ (double SHA-256 hashes/s), the refresh period is $2^{32}/\mathcal{R}$ seconds, thus the wait experienced by a newly arrived message is uniformly distributed on $\big[0,\,2^{32}/\mathcal{R}\big)$, giving

$$\mathbb{E}\big[\tau_{\mathsf{cb}}\big] = \frac{2^{31}}{\mathcal{R}}.$$

2) **Application-level mining delay** τ_{mine}. Each hash hits the application threshold T_{app} with probability $p = T_{\mathsf{app}}/2^{256}$, thus $\tau_{\mathsf{mine}} \sim \mathrm{Exp}(p\mathcal{R})$ and $\mathbb{E}\big[\tau_{\mathsf{mine}}\big] = 1/(p\mathcal{R})$.

Proposition 4 (Expected end-to-end latency)**.** *With the above notation, the expected time from the sender's request to its receipt of a valid proof package is*

$$\mathbb{E}[\tau_{\mathit{total}}] = \tau_{\mathsf{net}} + \tau_{\mathsf{sig}} + \frac{2^{31}}{\mathcal{R}} + \frac{1}{p\mathcal{R}}. \tag{3}$$

Interpretation: T_{app} trades off latency against attacker work—larger T_{app} reduces expected mining delay but reduces the adversary's per-proof cost proportionally.

Example 2 (Concrete latency)**.** *Assume (i) a typical network RTT of $\tau_{\mathsf{net}} = 100$ ms [23], (ii) a signature-check time of $\tau_{\mathsf{sig}} = 1$ ms, and (iii) a designated miner consisting of a single Antminer S21 XP Hyd Hardware [5], capable of computing hash rate $\mathcal{R} \approx 470$ TH/s.*

Coinbase refresh. $\mathbb{E}[\tau_{\mathsf{cb}}] = \dfrac{2^{31}}{\mathcal{R}} \approx 4.5\ \mu s$*—negligible.*

Difficulty choice. Choose an application target $T_{\mathsf{app}} = 2^{35}T_{\mathsf{btc}}$ (35 bits easier than Bitcoin's Threshold $\approx 2.2 \times 10^{53}$ [26]), thus $p = T_{\mathsf{app}}/2^{256}$, giving:

$$\mathbb{E}[\tau_{\mathsf{mine}}] = \frac{1}{p\mathcal{R}} \approx 33\ ms.$$

Total latency.

$$\mathbb{E}[\tau_{\mathit{total}}] \approx 0.100\ s + 0.001\ s + 0.0000045\ s + 0.033\ s\ \approx 134\ ms.$$

Moderate-hash miner. *If instead we use a miner with* ***0.01 % of the Bitcoin network's hash power*** *(about* 90 *PH/s), then* $\mathbb{E}[\tau_{\mathsf{cb}}] \approx 0.02\ \mu s$ *and* $\mathbb{E}[\tau_{\mathsf{mine}}] \approx 33\ ms \times \frac{0.47}{90} \approx 0.17\ ms$*. The end-to-end delay becomes*

$$\mathbb{E}[\tau_{\mathit{total}}] \approx 100\ ms + 1\ ms + 0.00002\ ms + 0.17\ ms \approx 101.2\ ms,$$

thus latency is now dominated by the network RTT[2]*. In both cases the protocol remains sub-second while forcing attackers to perform orders of magnitude more hashing to fabricate an acceptable proof.*

C. Liveness Guarantee

The previous subsection gave the *expected* end-to-end delay. Many applications, however, require a high-confidence bound that a proof package will arrive within a deadline.

Tail bound: Because the mining delay τ_{mine} is exponentially distributed with rate $p\mathcal{R}$, we have $\Pr[\tau_{\mathsf{mine}} > \Delta] = e^{-p\mathcal{R}\Delta}$.

Proposition 5 (Probabilistic liveness)**.** *Let $\varepsilon \in (0,1)$ be a target failure probability and set*

$$\Delta\ =\ \frac{\ln(1/\varepsilon)}{p\mathcal{R}}. \tag{4}$$

Then, with probability at least $1-\varepsilon$, the sender receives a valid proof package within

$$\tau_{\mathit{tot}}^{\max}\ =\ \tau_{\mathsf{net}}\ +\ \frac{2^{32}}{\mathcal{R}}\ +\ \tau_{\mathsf{sig}}\ +\ \Delta. \tag{5}$$

Sketch. The total delay is upper-bounded by

$$\tau_{\text{tot}} \le \tau_{\mathsf{net}} + \tau_{\mathsf{sig}} + \frac{2^{32}}{\mathcal{R}} + \tau_{\mathsf{mine}}.$$

The first three terms are treated as fixed upper bounds supplied by the deployment. For the exponential mining delay,

$$\Pr[\tau_{\mathsf{mine}} > \Delta] = e^{-p\mathcal{R}\Delta} = \varepsilon$$

by the chosen Δ. Hence

$$\Pr[\tau_{\text{tot}} \le \tau_{\text{tot}}^{\max}] \ge 1 - \varepsilon.$$

□

Interpretation: The tail probability decays exponentially in $p\mathcal{R}\Delta$, thus high-confidence deadlines can be met by selecting T_{app} (via p) and/or miner capacity.

Example 3 (99.9% liveness)**.** *Assume as previous example, (i) a typical RTT $\tau_{\mathsf{net}} = 100$ ms, (ii) a signature-check time*

[2]Which in case of having a connection within your continent/country could result in times up to RTT ≈ 30 ms. [23]

Component	Size (bytes)	Comment
Block header B	80	fixed in Bitcoin
Coinbase tx_{cb}	$\lvert C_0\rvert + 32$	$\lvert C_0\rvert \approx$100B baseline, + 32B hash
Merkle proof π_m	$32\lceil \log_2 n\rceil$	inclusion of tx_{cb} in block root

TABLE I
PROOF-PACKAGE COMPONENTS AND THEIR SIZES.

$\tau_{\mathsf{sig}} = 1$ ms, and (iii) a designated miner made of an Antminer S21 XP Hyd miner, so $\mathcal{R} \approx 470$ TH/s.

Difficulty choice. Choose an application target $T_{\mathsf{app}} = 2^{35} \times T_{\mathsf{btc}}$, giving $p = T_{\mathsf{app}}/2^{256}$.

Coinbase-refresh delay. $\mathbf{E}[\tau_{\mathsf{cb}}] = 2^{31}/\mathcal{R} \approx 4.5\,\mu s$.

Tail term for $\varepsilon = 10^{-3}$.
$\Delta = \ln(1/\varepsilon)/(p\mathcal{R}) \approx 0.23$ s.

Total bound.

$$\tau_{tot}^{\max} \approx 0.100\ s + 0.001\ s + 0.0000045\ s + 0.23\ s \approx 0.33\ s.$$

*Hence, with 99.9 % probability, the proof package arrives in under **400 ms**.*

Moderate-hash miner. *Now consider a miner with **0.01 % of Bitcoin's total hash power**, roughly 90 PH/s. Thus*

$$\mathbf{E}[\tau_{\mathsf{cb}}] \approx 0.02\,\mu s, \qquad \Delta \approx 0.23\ s \times \frac{0.47}{90} \approx 1.2\ ms.$$

Hence

$$\tau_{tot}^{\max} \approx 100\ ms + 1\ ms + 0.00002\ ms + 1.2\ ms \approx 102\ ms.$$

With a miner that powerful, the bound is dominated by the 100 ms RTT; proof generation contributes around 1 ms, and the 99.9 % liveness target is comfortably met.

D. Proof-Package Size

A proof package π delivered to the verifier consists of three mandatory objects—block header, coinbase transaction, and Merkle proof. Table I summarises their sizes.

Proposition 6 (Proof-Package Size)**.** *Let n be the number of transactions in the block, and $|C_0|$ be the baseline coinbase length without embedded data. The total size $S(n)$ (in bytes) of a proof package is*

$$\boxed{S(n) = 80 + \big(|C_0| + 32\big) + 32\,\lceil \log_2 n\rceil}. \tag{6}$$

Thus $S(n)$ grows logarithmically in n and is dominated by constants for realistic parameter values.

Proof. The expression is obtained by summing the byte lengths in Table I. ☐

Example 4 (Concrete size estimate)**.** *Assume a block contains $n = 3{,}000$ transactions ($\lceil \log_2 n\rceil = 12$), with $|C_0| = 100\,B$ the proof size is*

$$\begin{aligned} S(3{,}000) &= 80 + (100 + 32) + 32 \times 12 \\ &= 80 + 132 + 384 \\ &= 596 \text{ bytes}. \end{aligned}$$

A proof package remains comfortably under 1 kB even with thousands of block transactions and a 32-byte request commitment. Because Merkle paths scale logarithmically, proof size grows slowly with block size. Furthermore, the Merkle-path portion is not inherent: if a PoW blockchain placed the coinbase near the root of the transaction tree, the inclusion proof for the coinbase would collapse to a single hash—or disappear entirely if the protocol committed the coinbase directly in the header—reducing both bandwidth and verifier computation still further. Bitcoin does not adopt this layout, but it shows that the sizes derived above are upper bounds rather than fundamental limits of the method.

E. Verification Cost for IoT Devices

Let n be the number of transactions handled by the miner within the block. Let $d = \max(1, \lceil \log_2 n\rceil)$ be the depth of the Merkle tree that commits to all n transaction hashes in the block. Let C_{SHA256} denote the cost of a single SHA–256 hash evaluation on the device (in cycles or time), and let C_{sig} be the cost of a single signature verification on the same hardware.

To validate a proof package π, the verifier executes

- one double SHA-256 to recompute the block-header hash;
- one double SHA-256 to recompute the coinbase-transaction hash; and
- d double SHA-256 hashes to check the Merkle proof π_m and confirm that the coinbase transaction is included in the block's Merkle root.

Each double SHA-256 equals two single SHA-256 evaluations, thus the total hashing cost, denoted C_{PoW}, is

$$C_{\mathsf{PoW}} \;=\; 2\,(1 + 1 + d)\,C_{\mathsf{SHA256}} \;=\; (2d + 4)\,C_{\mathsf{SHA256}}. \tag{7}$$

Definition 1 (Cost ratio ρ)**.** *For any $n \geq 1$, the ratio of signature-verification cost to proof-package verification cost is*

$$\rho \;=\; \frac{C_{\mathsf{sig}}}{C_{\mathsf{PoW}}} \;=\; \frac{C_{\mathsf{sig}}/C_{\mathsf{SHA256}}}{2d + 4} \;=\; \frac{C_{\mathsf{sig}}/C_{\mathsf{SHA256}}}{2\max(1, \lceil \log_2 n\rceil) + 4}. \tag{8}$$

Example 5 (Measured timing on ESP32 MCU, $n = 4000$)**.** *We implemented the full proof-verification routine in C, using the ESP32 hardware; the source code and raw timing scripts are available online [25]. On a 240 MHz ESP32 MCU:*

- *Hash cost. A single SHA-256 call takes $C_{\mathsf{SHA256}} \approx 30\,\mu s$.*
- *Signature cost. Verifying an ECDSA-P256 signature takes $C_{\mathsf{sig}} \approx 260\,ms$.*

With $n = 4000$ transactions, the Merkle depth is $d = \lceil \log_2 4000\rceil = 12$. The verifier therefore performs

$$\begin{aligned} &1 \text{ (header)} + d \text{ (path)} + 1 \text{ (coinbase)} \\ &\quad = 14 \text{ double hashes} \\ &\quad = 28 \text{ single SHA-256 evaluations}. \end{aligned}$$

Thus,

$$C_{\mathsf{PoW}} = 28 \times 30\,\mu s \approx 840\,\mu s.$$

Therefore,

$$\text{Cost ratio } \rho = \frac{C_{\mathsf{sig}}}{C_{\mathsf{PoW}}} \approx \frac{260\,ms}{0.840\,ms} \approx 310.$$

MCU	STM32F221	Renesas S7G2	TI-TM4C1294XL	Raspberry Pi Pico
MPU	Cortex-M3	Cortex-M4	Cortex-M4F	RP2040, Cortex-M0+
RSA2048	111	10	88	33
ECDSA	-	40	254	117
PoW	0.61	0.25	0.51	0.87
ρ_{RSA}	180	40	170	40
ρ_{ECDSA}	-	160	500	130

TABLE II
WOLFSSL BENCHMARK RESULTS (ALL TIMES IN MILLISECONDS). "RSA2048" AND "ECDSA" GIVE THE TIME FOR A SINGLE SIGNATURE VERIFICATION; "POW" SHOWS OUR PROOF-PACKAGE CHECK. ρ_{RSA} AND ρ_{ECDSA} ARE THE CORRESPONDING COST RATIOS DEFINED IN DEFINITION 1. THE RENESAS S7G2 FEATURES A HARDWARE SHA ENGINE, BUT THE POW TIMES HERE ARE SOFTWARE-ONLY.

In other words, proof-package verification takes about 0.3% *of the time it takes for a single signature check on the same hardware.*

Benchmarks: Beyond our own timing on the ESP32 board (presented in Example 5), we broadened the evaluation to other popular IoT MCUs. The most common in IoT industry is the STM32 series (F, L, H, and R lines), the ESP32 by Espressif, and the Nordic nRF52 series. To obtain comparable hash and signature timings for these devices, we extracted the published numbers from the WolfSSL benchmark suite [24], one of the most widely used libraries for embedded cryptography. Except for the ESP32–which is built around a Tensilica core–all of these MCUs use ARM Cortex-M microprocessors (MPUs), giving a representative view across the dominant hardware platforms in today's IoT landscape.

As the benchmarks presented in Table II show, a software-only SHA-256 on a Cortex-M3 places PoW verification at under 2.5% of the cost of a single signature check. Because SHA-256 is much simpler to accelerate than public-key arithmetic, the gap widens on newer chips. A Cortex-M4 still hashes entirely in software in a few microseconds, yet RSA-2048 or ECDSA-P256 verification remains in the hundreds of milliseconds. Modern MCUs—such as the `STM32H7`, `nRF52`, and `Renesas DK-S7G2`—add on-chip SHA-256 engines that drive each hash down to mere microseconds, an order-of-magnitude boost, while offering no comparable hardware for RSA or ECDSA. With hardware hashing, PoW verification becomes up to $1000\times$ cheaper in CPU time than a full signature check, further tilting the cost balance toward the verifier. This asymmetry lets our scheme discard unauthenticated traffic early and cheaply, cutting energy draw and maintaining device responsiveness even under attack.

Best-Case Commitment: Header or Near-Header: Bitcoin fixes the coinbase transaction as a Merkle leaf, which forces every verifier to check an inclusion path of depth $\lceil \log_2 n \rceil$. Depth, however, is an implementation choice, not a consensus rule. A PoW blockchain optimised for lightweight verifiers could instead (i) place the coinbase as an immediate child of the root ($d = 1$) or (ii) embed the coinbase's hash directly in the block header ($d = 0$). Either change leaves consensus security intact: the header is still double-SHA-256-hashed and must satisfy the global difficulty target.

Example 6 (Best-Case $n = 4000$)**.** ***Near-header commitment*** **(**$d = 1$**).** *The verifier recomputes one double SHA-256 for the block header, one double SHA-256 for the single-level Merkle path, and the coinbase hash (*3 *double hashes* $= 6$ *single SHA-256 calls). This pushes the cost ratio from* $\rho = 310$ *in Example 5 to* $310 \times 28/6 \approx 1\,446$*.*

Header-level commitment **(**$d = 0$**).** *The verifier computes* two *double SHA-256 operations: one for the coinbase transaction itself and one for the block header (*2 *double hashes* $= 4$ *single SHA-256 calls). This layout is therefore* $\approx 1.5\times$ *faster than the near-header option and achieves* $\rho \approx 2\,170$*.*

Proof size. *The near-header proof includes the 80-byte block header, the 100-byte coinbase stub, and one 32-byte sibling hash, for* $\approx 212\,B$*. With header-level commitment the proof shrinks to the 80-byte header plus the 100-byte coinbase stub,* $\approx 180\,B$*. A further cut is possible: several coinbase fields (version, null out-point, sequence, locktime) are constant across all blocks. Omitting these fixed values would save about* $40\,B$*, trimming the proof to about* $140\,B$*.*

VII. EXTENSIONS

Overview. The baseline protocol makes three simplifying assumptions: (i) the miner serves one request at a time, (ii) accountability mechanisms are not enabled, and (iii) clients are benign. This section relaxes these assumptions via three orthogonal extensions: (A) batching for concurrent requests, (B) publicly verifiable and unframeable miner accountability, and (C) lightweight client-abuse mitigation. Each extension is optional and can be enabled independently.

A. Request Batching

In the baseline protocol a miner works *sequentially*: it accepts a single request, mines the corresponding block, and only then moves on to the next request. When dozens of clients compete for service, this one-at-a-time loop leaves hardware idle and drives confirmation latency up.

Batching idea. We allow the miner to aggregate the b requests that accumulate while it searches for a nonce. Each request contributes $c_i = H(h_i \| \mathsf{pk}_{S_i} \| \sigma_{S_i})$ to an *incremental* Merkle tree T whose root $\mathsf{root}(T)$ is placed in the coinbase. If a new request arrives during the current PoW search, the miner inserts the leaf, recomputes the path to the root, and updates $\mathsf{root}(T)$ the next time it tweaks the coinbase anyway (e.g. after nonce exhaustion). No additional block space is needed.

Proof size. A client now receives its Merkle authentication path in T. This path adds $32\lceil \log_2 b \rceil$ bytes to the proof

package. For $b = 256$ concurrent requests the increase is $32 \times 8 = 256$ bytes, bringing the 596-byte package of Example 4 to about 852 bytes.

Verification cost. The verifier hashes the same $\lceil \log_2 b \rceil$ siblings, a few microseconds on an ESP32. The miner pays $b - 1$ extra SHA-256 calls per batch—negligible beside the billions of header hashes it already computes.

Scalability. Both the communication overhead and the hashing work grow only logarithmically in b, thus throughput can rise by two orders of magnitude before the added cost becomes noticeable. We revisit this mechanism in Section VII-C, where a malicious client may try to inflate b to burden the system.

Takeaway: batching increases throughput while keeping verifier overhead logarithmic in batch size.

B. Proofs of Miner Misbehavior

Designated miners are maintained on a policy-driven allow-list $\mathbf{M}_{\text{desg}}$ that can be updated over time. We place no correctness trust in these miners: if a miner misbehaves, the event should be attributable and publicly verifiable thus that nodes can evict its identifier. At the same time, eviction must be *unframeable*: since any party can grind PoW while paying a coinbase output to a designated miner address, PoW binding alone is insufficient to blame (and evict) a miner.

Intuition. A key challenge is to enable eviction without enabling *framing*: since any party can mine PoW while directing the coinbase payout to a designated miner address, PoW binding alone cannot attribute misbehavior. We therefore attach a lightweight, transaction-based attestation that proves knowledge of a miner-specific attestation key and binds the proof to the sender's request. The verifier performs this extra validation only in the failure path (when σ_S is invalid); otherwise, the protocol proceeds unchanged.

Accountability rule (transaction-based, unframeable). For each designated miner $M \in \mathbf{M}_{\text{desg}}$, the verifier stores a fixed *attestation identifier* a_M corresponding to a legacy P2PKH public-key hash (i.e., $a_M = \mathsf{Hash160}(\mathsf{pk}_M)$ for some secp256k1 public key pk_M).[3] For each sender request $(m, \mathsf{pk}_S, \sigma_S)$, define $h := H(m)$ and the request commitment

$$c \ := \ H\big(h \,\|\, \mathsf{pk}_S \,\|\, \sigma_S\big). \tag{9}$$

The miner commits to c in the coinbase transaction (in place of committing only h), and additionally inserts two auxiliary transactions tx_A, tx_B *immediately after* the coinbase transaction tx_{cb} in the block template:

- tx_A creates an output paying to the P2PKH hash a_M (i.e., to $\mathsf{Hash160}(\mathsf{pk}_M)$).
- tx_B spends that output using *legacy* P2PKH, thereby revealing pk_M and a Bitcoin ECDSA signature, and it includes c in an `OP_RETURN` output.[4]

[3]This attestation key can be distinct from the miner's reward payout key; it is used only for accountability.

[4]We use legacy P2PKH for tx_B thus that its signature and public key are committed by the block's transaction Merkle tree (unlike SegWit witness data).

The miner returns a proof package containing $(h, \mathsf{pk}_S, \sigma_S, \mathsf{proof}, tx_A, tx_B)$, where proof includes the block header, tx_{cb}, and a Merkle (multi-)proof authenticating inclusion of $\{tx_{cb}, tx_A, tx_B\}$ under the header's transaction Merkle root.

Establishing miner misbehavior. Given such a package, any verifier (or auditor) establishes miner misbehavior if: (i) all PoW checks pass (header difficulty, miner binding, Merkle inclusion, and the commitment check $c = H(h \| \mathsf{pk}_S \| \sigma_S)$); (ii) the sender's signature σ_S fails to verify under pk_S; and (iii) the miner attestation validates, namely:

1) tx_B spends the designated output of tx_A;
2) the public key pk_M revealed in tx_B satisfies $\mathsf{Hash160}(\mathsf{pk}_M) = a_M$ (matching the miner identity on the allow-list); and
3) Bitcoin script validation for tx_B succeeds (i.e., the ECDSA signature in tx_B verifies under pk_M and covers the output carrying c).

In this case the tuple $(h, \mathsf{pk}_S, \sigma_S, \mathsf{proof}, tx_A, tx_B)$ is a short, publicly verifiable proof that the designated miner produced a proof package bound to an invalid sender signature, and it justifies evicting the miner's identifier from $\mathbf{M}_{\text{desg}}$. If the sender signature is invalid but the miner attestation does *not* validate, the verifier drops the message but does not evict any miner, since the evidence is not attributable (it could be a framing attempt by an adversary mining to a designated miner address).

Verification path. The lightweight verifier proceeds as follows:

- **Cheap checks first.** The verifier first checks that pk_S is allow-listed, then performs the hash-only PoW validation (header threshold and miner binding), and verifies the Merkle (multi-)proof authenticating inclusion of (tx_{cb}, tx_A, tx_B). It also checks the basic structure of the accountability transactions (e.g., that tx_B spends the designated output of tx_A and carries the commitment c).
- **Sender signature.** It then verifies σ_S under pk_S. If σ_S is valid, the protocol proceeds unchanged; the verifier need not perform Bitcoin-script validation for tx_B.
- **Failure path (accountability).** If σ_S is *invalid*, the verifier checks the miner attestation via (tx_A, tx_B). If the attestation validates, the verifier evicts the implicated miner; otherwise, it drops the message without blame.

Overhead. This extension adds only the bytes for tx_A, tx_B and a Merkle (multi-)proof authenticating their inclusion. Because tx_A and tx_B are included only in the short window while producing a recycled-header proof, the probability that they appear on-chain in an eventual winning Bitcoin block is negligible in practice; hence their expected on-chain footprint is near zero. On the verifier, the additional hashing work consists of computing two extra transaction hashes and a small constant-factor increase in Merkle-proof processing (since $\{tx_{cb}, tx_A, tx_B\}$ are consecutive and placed immediately after the coinbase). The only additional public-key cost is a single secp256k1 verification for tx_B, and it is performed only in the failure path where σ_S is invalid.

Soundness (unframeability).

Proposition 7 (Unframeable miner misbehavior proofs)**.** *Assume ECDSA is EUF-CMA secure and $H(\cdot)$ is collision resistant. Then no PPT adversary can produce a proof package that causes a verifier to evict an* honest *designated miner M unless it forges a valid ECDSA signature under pk_M. In particular, mining PoW while paying a coinbase output to M (without knowing sk_M) cannot frame M into eviction.*

Proof sketch. A verifier evicts M only if Bitcoin-script validation for tx_B succeeds and tx_B reveals a public key pk_M whose hash matches the allow-listed identifier a_M. Producing such a tx_B without knowledge of sk_M constitutes an ECDSA forgery under pk_M, which is infeasible by EUF-CMA security. Collision resistance of $H(\cdot)$ ensures that the request commitment c uniquely binds the tuple $(h, \mathsf{pk}_S, \sigma_S)$ to the proof package. □

Takeaway: misbehavior is attributable without framing, enabling safe allow-list updates.

C. Client Abuse Mitigation

Batching (Section VII-A) makes the miner's load grow only $O(\log b)$ with the number of in-flight requests b, yet an attacker that can flood the miner for free still causes needless work and network traffic. We curb this risk with a light registration layer that targets *only partially trusted* clients.

Selective registration and prepaid quota. Trusted clients—operator-controlled nodes or long-standing partners—may submit requests freely. All other clients must *register* by pre-purchasing a quota of k future requests with a single lump-sum payment. The payment is made off-chain, either through a Lightning-network transfer or a direct one-way channel, thus the transaction incurs negligible cost and adds no latency to the service itself.

A possible concern is that a miner might pocket the payment and refuse service. In practice, this threat is negligible: the reputation damage a Bitcoin-scale miner would incur by cheating on a small prepaid quota far outweighs the nominal revenue it could gain.

Economic throttling. Because the quota is paid *up front*, a malicious client must spend real funds for every request it injects. Doubling the number of concurrent requests b doubles the attacker's monetary outlay, whereas the miner's extra work and the proof size grow only $O(\log b)$ (one additional SHA-256 step and 32 bytes).

Extra miner incentive. Although many miners would recycle idle PoW capacity for reputational or community reasons, the prepaid quota provides a direct revenue stream with practically zero marginal cost. This incentive attracts more miners to the service and boosts overall throughput, all while keeping honest users' costs minuscule.

Takeaway: prepaid quotas shift miner-side abuse costs to adversarial clients without increasing verifier cost.

VIII. Implementation

To demonstrate the real-world feasibility of our proposed framework, we implemented it on a physical Bitcoin mining platform. The mining platform uses Pooler's `cpuminer` implementation [19]. The testbed is controlled by our custom Stratum server, which emulates the operational behavior of a commercial mining pool. For independent verification, this implementation is publicly accessible via a dedicated IP address and port. The empirical results of our network latency measurements are presented in section VIII-B.

Operation	Current (mA)	Power (mW)	Time (ms)	Energy (mJ)
Proof verification	44.5	147	0.84	0.12
ECDSA verification	52.5	173	260	45.0

TABLE III
Power measurements.

Location	Distance (km)	Latency (ms)
University of Alberta	≤1	16
Downtown Edmonton	5	25
Calgary	300	40
Ottawa	2,800	65

TABLE IV
Delay measurements

In addition to performance, we characterized the energy consumption of the microcontrollers (MCUs) to evaluate the system's power efficiency.

Figure 3 summarizes the full testbed: the upper path is used for network-latency measurements, while the lower path is used for verifier-side timing and energy measurements.

The following section details our measurement methodology.

A. Energy Consumption

As mentioned in Example 5, ECDSA verification is far slower than verifying a proof package on ESP32 hardware. To compare *energy* rather than just time, we measured the instantaneous power draw of each operation using a Nordic Power Profiler Kit II (PPK2), shown in Fig. 4. The PPK2 supplied a regulated $3.0\,\mathrm{V}$ to the device, consistent with the datasheet. To obtain stable traces for individual operations, we executed each operation in a tight `while()` loop and averaged over thousands of iterations [25].

Average execution time for each operation was measured in firmware; average power was read from the PPK2 trace. Energy per operation was then computed as $E = \bar{P} \times \bar{t}$. Table III summarizes the results. ECDSA verification takes $260\,\mathrm{ms}$ at $173\,\mathrm{mW}$, i.e., $\approx 45\,\mathrm{mJ}$, whereas proof-package verification takes $0.84\,\mathrm{ms}$ at $147\,\mathrm{mW}$, i.e., $\approx 0.12\,\mathrm{mJ}$. Thus, the proof-package path uses about $45/0.12 \approx 375\times$ *less* energy than ECDSA verification.

B. Latency

This section details the empirical measurement of the system's end-to-end latency. We define this latency as the interval from the moment a client transmits its message to the point at which our Stratum server dispatches an updated mining job, incorporating that message, to the miner and back to the client.

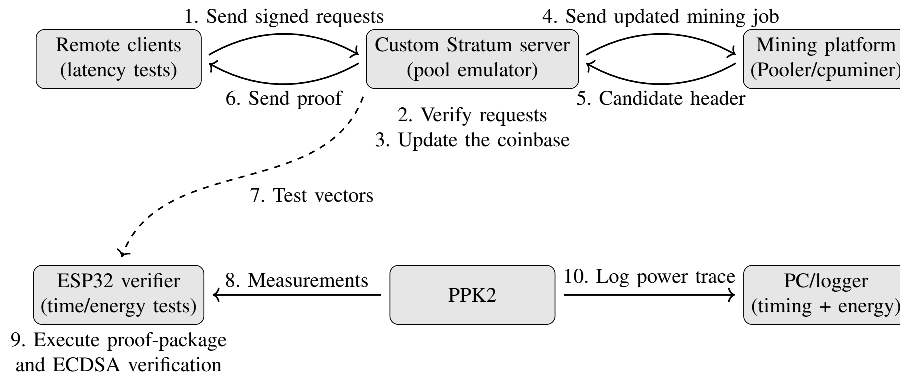

Fig. 3. Illustration of the full testbed and deployment setup. Remote clients send signed requests to the custom Stratum server, which emulates a mining pool: it validates and batches requests, embeds the resulting commitment in the coinbase template, and issues an updated mining job to the mining platform. The network-latency measurements use this request–mining–response path. Separately, proof-package and ECDSA verification are executed on the ESP32 verifier while the PPK2 supplies regulated power and records the power trace.

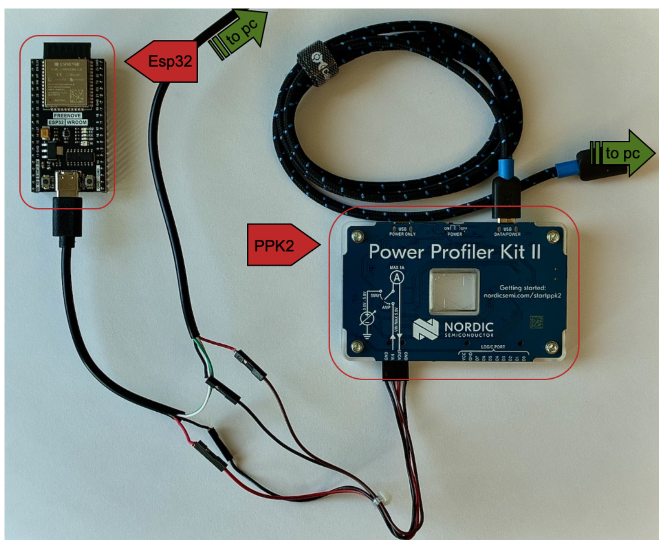

Fig. 4. Power-measurement setup.

Upon receiving a client request, our Stratum server executes the following sequence: 1) it validates the message's signature and associated payment; 2) it aggregates the message with others into a Merkle tree; 3) it embeds the resulting 32-byte Merkle root into the coinbase transaction of a new block template; and 4) it updates the miner with a new block header for processing.

To ensure minimal latency, a critical requirement for our scheme, we configured the server to issue an urgent header update. This is achieved by setting the clean_jobs = True flag in the mining.notify() function call. This flag compels the miner to immediately cease its current work and begin hashing the new block header. This approach is justified by the memoryless property of the PoW algorithm; because each hash attempt is an independent probabilistic event, abandoning the previous search does not introduce any computational inefficiency or negatively impact the overall mining outcome.

While an estimated delay was calculated in Section VI, the results presented here are empirical measurements from thousands of transactions conducted on a live deployment. These tests were performed from various clients across the globe, all communicating with our mining server located in Edmonton, Canada. A summary of the measured latencies from different geographical locations is presented in Table IV.

C. Comparison against a symmetric PoW baseline

A natural symmetric baseline is a Hashcash-style client puzzle in which every sender must compute a nonce r per message such that

$$H_2(c \parallel r) \le T_{\text{pow}}, \quad \text{where } c = H(h \| \mathsf{pk}_S \| \sigma_S),$$

and the verifier checks this condition (one double-SHA-256) before attempting the expensive signature verification. This baseline preserves the same "cheap-first" verification order as SPARE, but it is *symmetric*: honest senders incur the same expected PoW cost as attackers.

To make the comparison fair, we set $T_{\text{pow}} = T_{\text{app}}$, i.e., the same acceptance probability $p = T_{\text{app}}/2^{256}$ that SPARE uses for recycled headers. The expected number of trials is then $\mathbb{E}[\#\text{trials}] = 1/p$. Under the parameters used in Example 2 (designated miner hash rate $\mathcal{R} \approx 470$ TH/s and $\mathbb{E}[\tau_{\text{mine}}] \approx 33$ ms), this corresponds to

$$\mathbb{E}[\#\text{trials}] = \frac{1}{p} = \mathcal{R} \cdot \mathbb{E}[\tau_{\mathsf{mine}}] \approx 1.6 \times 10^{13} \approx 2^{44}.$$

To estimate the *honest-sender* overhead in this symmetric baseline, we use published SHA2-256 benchmark rates from hashcat (Hash-Mode 1400). A desktop-class CPU (Intel Core i7-9700K) achieves about 299.3 MH/s [1], while a high-end consumer GPU (RTX 4090) achieves about 22,138.4 MH/s [2]. Approximating one $H_2(\cdot)$ trial as two SHA2-256 evaluations, the expected solve time is roughly

$$\mathbb{E}[\tau_{\text{solve}}] \approx \frac{1/p}{(\mathcal{R}_{\mathsf{SHA}}/2)} = \frac{2}{p\,\mathcal{R}_{\mathsf{SHA}}}.$$

At the matched difficulty above ($1/p \approx 1.6 \times 10^{13}$), this yields an expected per-message PoW time of roughly ~ 29 hours on the i7-9700K and ~ 25 minutes on an RTX 4090. In contrast, SPARE imposes *no* per-message PoW solving cost on honest senders (beyond the ordinary signature), because

Scheme	Verifier precheck (before signature)	Honest-sender PoW cost	Attacker PoW cost
Symmetric client puzzle	Verify $H_2(c \parallel r) \leq T_{\mathsf{pow}}$ (1 double-SHA-256; ≈ 0.06 ms on ESP32)	$\mathbb{E}[\#\text{trials}] = 1/p \approx 2^{44}$ at matched difficulty; ~ 29 h on i7-9700K [1], ~ 25 min on RTX 4090 [2]	$\mathbb{E}[\#\text{trials}] = 1/p \approx 2^{44}$ trials
SPARE (ours)	Verify recycled header + Merkle inclusion proof (≈ 0.84 ms on ESP32)	No per-message PoW solving (beyond signature); obtain proof from miner	$\mathbb{E}[\#\text{trials}] = 1/p \approx 2^{44}$ trials to fabricate a valid proof

TABLE V

COMPARISON TO A SYMMETRIC POW BASELINE. BOTH SCHEMES GATE SIGNATURE VERIFICATION WITH CHEAP HASH CHECKS, BUT ONLY SPARE AVOIDS IMPOSING PER-MESSAGE PUZZLE SOLVING ON HONEST SENDERS BY REUSING MINERS' DISCARDED WORK.

proofs are obtained by reusing miners' discarded headers; the sender's added overhead is dominated by network RTT and miner responsiveness (Table IV).

Conversely, if the symmetric baseline is tuned to keep sender-side delay small on non-mining hardware, then attacker work decreases proportionally. For instance, to target ≈ 100 ms solve time on the i7-9700K, the expected trials drop to about $(\mathcal{R}_{\mathsf{SHA}}/2)\cdot 0.1\text{s} \approx 1.5\times 10^7 \approx 2^{24}$, i.e., about a $10^6\times$ reduction relative to 2^{44}. This illustrates the core advantage of SPARE: it retains the DoS-resistance benefits of PoW-gating at the verifier, while avoiding the blanket per-message puzzle-solving burden that symmetric client puzzles impose on honest senders.

IX. RELATED WORK

Classic Hashcash-style email stamps [4], Juels and Brainard's client puzzles [11], and more recent non-interactive, VDF-based puzzles for web services [20] require each sender to solve a fresh puzzle per message, thus legitimate traffic bears the same computational burden as an attacker. In resource-constrained sensor/IoT settings, prior work documented that adversaries can exploit the high cost of signature verification to mount DoS attacks by flooding devices with well-formed but invalid messages [15]. To preserve *public-key* authentication and non-repudiation while throttling such floods, researchers proposed *message-specific puzzles* and *pre-authentication filters* that a node can verify quickly *before* attempting ECDSA/RSA. Examples include weak-authenticator puzzles for broadcast authentication [15] and DoS-resilient code dissemination protocols (e.g., Seluge) that gate signature verification with lightweight checks and Merkle constructions [9]. Later variants make puzzle difficulty dynamic to bound sender-side delay while keeping attackers' workload high [3]. These approaches retain end-to-end digital signatures but still require honest senders to solve a new puzzle for each message and to tune difficulty against adversaries.

Blockchain has also been used to decentralize *authentication* and *key management* in IoT deployments (e.g., Bubbles of Trust [8], a lightweight blockchain-based authentication mechanism [12], and blockchain-based authentication/key management using hash-chain techniques [18]). These systems primarily use the blockchain as a distributed trust substrate (e.g., to record, distribute, or audit authentication and keying state). In contrast, our focus is orthogonal: we target *signature-verification* DoS on constrained verifiers, and we use the blockchain's *mining effort* (rather than its ledger semantics) to create a cheap, message-level pre-authentication filter.

Our scheme differs in how it creates *asymmetry*: rather than using a blockchain primarily as a ledger for authentication state [8], [12], [18] or asking honest senders to compute puzzles, it reuses *non-winning Bitcoin headers* as a pre-authentication stamp. Legitimate senders attach a compact header + Merkle proof that verifiers can check with a few hashes, while adversaries cannot piggy-back on recycled work and must grind fresh headers (or effectively mine on the designated miner's behalf). Thus we shift solving cost off honest parties without weakening public-key authenticity.

Beyond DoS mitigation, others have recycled blockchain PoW for different purposes. *Merged mining* (auxiliary PoW) lets a child chain accept PoW performed for a parent chain; *Namecoin* is the canonical example of reusing Bitcoin's PoW to secure a separate naming blockchain, with similar arrangements later adopted elsewhere (e.g., Dogecoin–Litecoin). These recycle effort at the *consensus* layer to secure additional blockchains, not to create message-level cost asymmetry in a DoS filter, and they require protocol support on the auxiliary chain. Jakobsson and Juels' "bread-pudding" protocols restructure work into many low-difficulty mini-puzzles to mint MicroMint coins [10]; related "useful PoW" systems redesign puzzles entirely (e.g., Permacoin, Primecoin) [13], [16]. In contrast, our mechanism keeps Bitcoin's puzzle unchanged and reuses discarded headers *at the message layer*.

Our distinction. To the best of our knowledge, we are the first to *combine* (i) reuse of discarded Bitcoin PoW at the message layer with (ii) a strong attacker–honest asymmetry in solving vs. verification cost, *without* modifying the underlying blockchain. We bind commitments into a designated miner's coinbase and let honest senders attach the resulting non-winning headers and Merkle proofs; attackers, by contrast, must still grind fresh work to meet the receiver's threshold.

X. CONCLUSION

We presented an *asymmetric*, proof-of-work–based mitigation for signature-amplification attacks on IoT devices. Our design recycles non-winning Bitcoin block headers: a designated miner embeds the request commitment in its coinbase, and any header below an application-chosen threshold serves as a lightweight pre-authentication proof. Because proofs are bound to the miner's payout address, adversaries cannot piggy-back on recycled work; to pass the filter they must grind fresh headers (or effectively mine on the designated miner's behalf), creating a durable resource gap that spares honest senders and verifiers.

A prototype on an ESP32 MCU shows that PoW validation completes in $\mathbf{0.8\,ms}$ versus $\mathbf{260\,ms}$ for ECDSA

($> \mathbf{300}\times$ speed-up); proof packages remain $< \mathbf{1\,kB}$, end-to-end latency is dominated by sender–miner RTT, and power measurements confirm orders-of-magnitude lower energy than signature checks. The mechanism requires no blockchain modifications, scales to thousands of devices per miner, and immediately hardens firmware updates, certificate rotation, and other unsolicited IoT traffic. It also supports batching (one Merkle root for many messages) and *publicly verifiable* proofs of miner misbehavior. When a miner is unavailable or misbehaves, the verifier safely falls back to direct signature verification, thus authenticity never depends on miner behavior.

These properties make the protocol a practical, drop-in filter that leverages existing mining effort to shift costs off honest parties. Future work includes larger-scale deployments with external miners, broader hardware coverage, and field measurements across diverse networks.

REFERENCES

[1] Is any benchmark of intel cpu s ? (hashcat forum post with sha2-256 results). https://hashcat.net/forum/thread-9042-post-47927.html, 2020. Reports hashcat benchmark for Hash-Mode 1400 (SHA2-256) on Intel Core i7-9700K: 299.3 MH/s; accessed 2026-02-02.

[2] Nvidia rtx 4090 benchmark hc 7.0.0 (hashcat forum archive with sha2-256 results). https://hashcat.net/forum/archive/index.php?thread-13338.html=, 2025. Reports hashcat benchmark for Hash-Mode 1400 (SHA2-256) on RTX 4090: 22138.4 MH/s; accessed 2026-02-02.

[3] Farah Afianti, Wirawan, and Titiek Suryani. Dynamic cipher puzzle for efficient broadcast authentication in wireless sensor networks. *Sensors*, 18(11):4021, 2018. doi:10.3390/s18114021.

[4] Adam Back. Hashcash – a denial of service counter-measure. Technical report, hashcash.org, 2002. URL: www.hashcash.org/hashcash.pdf.

[5] BITMAIN Technologies. Bitcoin miner s21 XP hyd., 2025. URL: https://shop.bitmain.com/product/detail?pid=00020250312184146889fF0fd5720644.

[6] Marcel Deer. What is a coinbase transaction?, 2023. URL: https://cointelegraph.com/explained/what-is-a-coinbase-transaction.

[7] Michael Fagan, Katerina N. Megas, Karen Scarfone, and Matthew Smith. IoT device cybersecurity capability core baseline. NIST Interagency/Internal Report 8259A, National Institute of Standards and Technology, Gaithersburg, MD, USA, May 2020. doi:10.6028/NIST.IR.8259A.

[8] Mohamed Tahar Hammi, Badis Hammi, Patrick Bellot, and Ahmed Serhrouchni. Bubbles of trust: A decentralized blockchain-based authentication system for IoT. *Computers & Security*, 78:126–142, September 2018. doi:10.1016/j.cose.2018.06.004.

[9] Sangwon Hyun, Peng Ning, An Liu, and Wenliang Du. Seluge: Secure and dos-resistant code dissemination in wireless sensor networks. In *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN '08)*, pages 445–456. IEEE, 2008. doi:10.1109/IPSN.2008.12.

[10] Markus Jakobsson and Ari Juels. Proofs of work and bread pudding protocols. In *Secure Information Networks: Communications and Multimedia Security*, IFIP Conference Proceedings, pages 258–272. Springer, 1999. doi:10.1007/978-0-387-35568-9_18.

[11] Ari Juels and John Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *Network and Distributed System Security Symposium (NDSS)*, pages 151–165, 1999. URL: https://www.ndss-symposium.org/ndss1999/cryptographic-defense-against-connection-depletion-attacks/.

[12] Umair Khalid, Muhammad Asim, Thar Baker, Patrick C. K. Hung, Muhammad Adnan Tariq, and Laura Rafferty. A decentralized lightweight blockchain-based authentication mechanism for IoT systems. *Cluster Computing*, 23(3):2067–2087, February 2020. doi:10.1007/s10586-020-03058-6.

[13] Sunny King. Primecoin: Cryptocurrency with prime number proof-of-work. https://primecoin.io/primecoin-paper.pdf, 2013.

[14] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987. doi:10.1090/S0025-5718-1987-0866109-5.

[15] An Liu, Peng Ning, and Wenliang Du. Mitigating DoS attacks against broadcast authentication in wireless sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 4(1):1–35, 2008. doi:10.1145/1325651.1325652.

[16] Andrew Miller, Ari Juels, Elaine Shi, Bryan Parno, and Jonathan Katz. Permacoin: Repurposing Bitcoin work for data preservation. In *2014 IEEE Symposium on Security and Privacy (S&P)*, pages 475–490. IEEE, 2014. doi:10.1109/SP.2014.37.

[17] Brendan Moran, Hannes Tschofenig, and Henk Birkholz. A manifest information model for firmware updates in internet of things (IoT) devices. RFC 9124, January 2022. URL: https://www.rfc-editor.org/info/rfc9124, doi:10.17487/RFC9124.

[18] Soumyashree S. Panda, Debasish Jena, Bhabendu Kumar Mohanta, Somula Ramasubbareddy, Mahmoud Daneshmand, and Amir H. Gandomi. Authentication and key management in distributed IoT using blockchain technology. *IEEE Internet of Things Journal*, 8(16):12947–12954, August 2021. Article 9369319. doi:10.1109/JIOT.2021.3063806.

[19] Pooler. This is a multi-threaded CPU miner for Litecoin and Bitcoin, fork of jeff garzik's reference cpuminer, 2020. URL: https://github.com/pooler/cpuminer.

[20] Mayank Raikwar and Danilo Gligoroski. Non-interactive VDF client puzzle for DoS mitigation. In *Proceedings of the 2021 European Interdisciplinary Cybersecurity Conference (EICC '21)*, 2021. doi:10.1145/3487405.3487406.

[21] Ivan Ristić. TLS renegotiation and denial of service attacks. https://blog.qualys.com/product-tech/2011/10/31/tls-renegotiation-and-denial-of-service-attacks, 2011. Describes handshake-exhaustion attacks on TLS servers.

[22] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978. doi:10.1145/359340.359342.

[23] Verizon Business. IP latency statistics, 2025. URL: https://www.verizon.com/business/terms/latency/.

[24] wolfSSL Inc. Benchmarking wolfssl and wolfcrypt, 2025. URL: https://www.wolfssl.com/docs/benchmarks/.

[25] Amirhosein Yari. Recycling discarded proof-of-work. https://github.com/AmirhoseinYari/Recycling-Discarded-Proof-of-Work, 2025. Accessed: September 2025.

[26] YCharts. Bitcoin average difficulty, 2025. URL: https://ycharts.com/indicators/bitcoin_average_difficulty.