Bloch-SIS-PoW — ASIC customization spec

What a custom miner (ASIC/FPGA) must accelerate to mine the Bloch-SIS proof-of-work, and how to build one.

Bloch-SIS-PoW · ownerless protocol · technical note · **UNAUDITED** · parameters are NOT security-validated

> **Read first — honesty baseline.** This is an **engineering description of the current algorithm, not an endorsement to build hardware**. The coin (BLCH) is **not a security and not an asset — no price, no value claim**; there is no economic reason to build an ASIC for it. The PoW parameters below are **explicitly NOT security-validated** — at `β = q/16` the residual gate is structurally weak by the protocol's own admission, the chain is **unaudited**, **51%-attackable**, and the no-shortcut/asymmetry proof for the SIS gate is **outstanding**. Changing any consensus parameter here is a hard fork. Build only for research/education.

1 · The PoW in one page

A valid block carries a **SIS witness**. Mining searches for a nonce/seed such that a short integer vector `s` satisfies a lattice gate AND the block's auxiliary hash clears the difficulty target. Two independent conditions per candidate:

```
candidate(seed, s):
  # 1. deterministically expand the public problem from the seed (SHAKE-256)
  (A, t) = expand_matrix_and_target(seed)      # A ∈ Z_q^{m×n}, t ∈ Z_q^m
  # 2. STRUCTURAL GATE (Module-SIS): the residual must be short on k coeffs
  r = A·s − t   (mod q, centered)              # r ∈ Z_q^m
  gate_ok = ( ||r||_∞ < β  on the k designated coefficients )
  # 3. WORK GATE (hashcash): auxiliary hash must beat the target
  h = SHAKE256( domain_aux || block_header || s )
  work_ok = ( h ≤ target )                     # ASERT-tuned difficulty
  valid = gate_ok AND work_ok
```

Security is **cumulative SHAKE-256 hashcash work**; the SIS gate is a **structural filter** (a candidate must be lattice-valid to even be hashed), not a security claim by itself.

2 · Canonical parameters (consensus — changing them is a hard fork)

Symbol	Value	Meaning
`q`	**8 380 417** = $2^{23} - 2^{13} + 1$	Prime modulus. **Identical to ML-DSA-65 / FIPS 204 (Dilithium)** — deliberately, so an NTT core can be shared with the signature subsystem.
`n`	**256**	Solution-vector dimension. `s ∈ Z^n`, `||s||_∞ ≤ B`.
`m`	**512** (= 2n)	Rows of the public matrix `A ∈ Z_q^{m×n}`.
`β`	**q / 16** ≈ 523 776	Residual ∞-norm bound. **NOT security-validated** — weak by design at this value.
`k`	**8** (mainnet, active) · 4 (testnet)	Number of residual coefficients the gate constrains. Higher k ⇒ exponentially rarer candidates (k=8 is ~4096× rarer than k=4).
hash	**SHAKE-256** (Keccak-f[1600])	Seed expansion, auxiliary work hash, target derivation.

Domain-separation labels: `BLOCH-POW-SEED-V1`, `BLOCH-POW-AUX-V1`, `BLOCH-POW-TARGET-V1`. Field arithmetic is *centered* in `[-(q-1)/2, (q-1)/2]` because the gate measures distance to the nearest multiple of q.

3 · What an ASIC must accelerate (in cost order)

3.1 — Keccak-f[1600] / SHAKE-256 (throughput core)

Every candidate needs SHAKE-256 for (a) expanding `A` (512×256 ≈ 131 072 field entries) and `t` from the seed, and (b) the auxiliary work hash. This dominates gate/s. A standard high-throughput **Keccak permutation array** (unrolled or pipelined round logic, many parallel lanes) is the heart of the chip — the same primitive SHA-3 ASICs already implement.

3.2 — Matrix-vector product `A·s mod q` (the lattice unit)

`A·s` is 512 dot-products of length 256 over Z_q, $q \approx 2^{23}$ (24-bit). Because `q` is the ML-DSA-65 prime, you can **reuse Dilithium/Kyber NTT hardware IP**: butterfly units with Montgomery/Barrett reduction mod 8 380 417. If `A` is treated as a structured module over the ring, an **NTT-based multiply** beats the schoolbook `m·n` MACs; otherwise a wide MAC array with centered reduction works. Reduction is the reusable building block — a 24-bit modular multiply-accumulate cell.

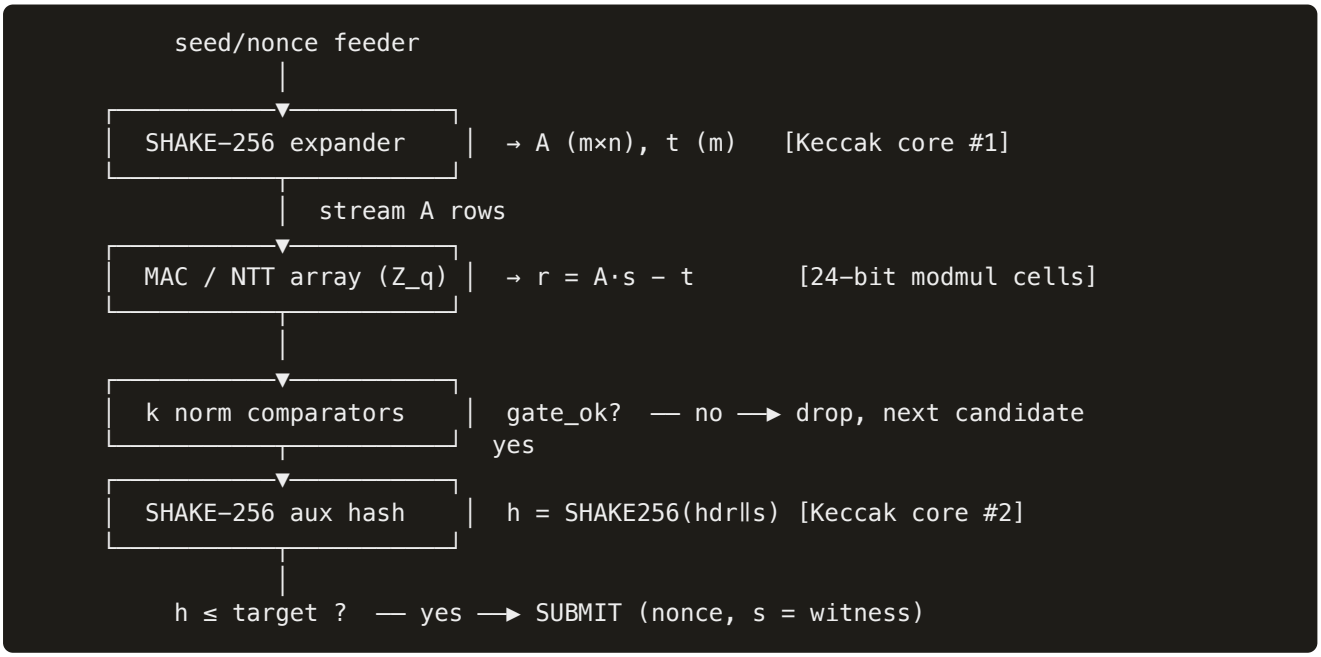
3.3 — Residual norm gate (comparators)

Compute `r = A·s − t` centered, then check `|r_i| < β` on the k designated coefficients. Cheap: k parallel magnitude comparators against `β = q/16`. This gate rejects the vast majority of candidates *before* the work hash, so place it early in the pipeline to save Keccak energy.

3.4 — Difficulty compare (hashcash)

Compare the aux SHAKE-256 output to the ASERT-tuned `target` (256-bit compare). Difficulty retargets per block (max ±4× per step, ~30 s target time).

4 · Reference pipeline



Design levers: (1) generate `A` once per seed and iterate many `s` against it (amortize the expander); (2) the gate is the natural early-reject — size the Keccak-aux stage for the post-gate survival rate only; (3) the

whole chip is **compute-bound on Keccak + 24-bit modmul** and can keep `A` on-die (≈ 384 KiB at 24 bit) to avoid off-chip bandwidth, which is what makes it ASIC-amenable rather than memory-hard.

5 · Practical path

1. **FPGA first.** Prototype the pipeline on an FPGA against the reference solver (`crates/bloch-sis-pow`) — validate that your witnesses are accepted by `verify_sis_pow` before any silicon.
2. **Reuse lattice IP.** The `q = 8 380 417` NTT/reduction cores from open Dilithium/Kyber HW projects drop in directly.
3. **Match the wire.** Emit the witness exactly as the node expects (solution vector `s`, centered representation) — a mismatch fails `validate_structure`.
4. **Pool or solo.** Solo submits full blocks; the reference pool (`postern-pool`) uses a SIS-native share protocol. There is no Stratum path for the lattice witness in the stock share format.

> **Do not over-invest.** `β = q/16` and the current `k` are placeholders pending the concrete-security analysis and IACR ePrint; a future hard fork can change `q`, `n`, `m`, `β`, or `k` and invalidate a fixed-parameter chip. The coin is worthless by design. This document describes an algorithm, not an investment.
