MERCAHASH V2

Technical Overview of Mercatura’s Memory-Hard Proof-of-Work

Authoritative 16 MiB profile • Mercatura Core

Mainnet technical note • 23 November 2026

Executive summary

MercaHash V2 is Mercatura’s mainnet consensus proof-of-work function from genesis. It accepts the canonical 80-byte block header, derives a 1,024-bit working state, fills and mutates a private **16 MiB** scratchpad, performs data-dependent memory mixing with periodic cross-lane and SHA3-512 checkpoint mixing, samples the resulting memory state, and produces a 256-bit SHA3-256 proof-of-work value.

The design goal is not to make specialized hardware impossible. MercaHash combines substantial mutable memory, irregular data-dependent access, serial state evolution, and general-purpose integer execution. CPUs and GPUs can both implement it; actual efficiency and hardware advantage must be measured rather than assumed.

MercaHash is intentionally separated from Mercatura’s block identity. The Bitcoin-style SHA256d header hash remains the block identifier used by indexes, RPCs, locators, previous-block references, and explorer-visible block hashes. MercaHash V2 is used specifically for mining and proof-of-work validation.

Scope. This technical overview describes the mainnet MercaHash V2 design, frozen workload, consensus role, performance implications, and security rationale. It is not a formal cryptographic proof of specialized-hardware resistance.

1. Design goals and consensus architecture

MercaHash V2 is a memory-hard proof-of-work designed for deterministic integration into a Bitcoin Core-derived chain. It avoids a virtual machine, JIT compiler, or randomized instruction program. Instead, it combines a fixed ARX state machine, a 16 MiB mutable scratchpad, data-dependent memory references, and periodic SHA3 checkpoints.

1.1 Separation of block identity and proof of work

Mercatura deliberately gives the 80-byte block header two cryptographic interpretations. The ordinary double-SHA256 header hash remains the block's identity, while MercaHash V2 produces the value compared with the proof-of-work target. This preserves ordinary chain indexing, block locators, disk identity, RPC block hashes, compact-block identity, and previous-block linkage.

Input	Function	Consensus role
80-byte block header	SHA256d GetHash()	Block identity / chain references
80-byte block header	MercaHash V2	Proof-of-work value

1.2 Proof-of-work acceptance

Mining varies the canonical header (normally by changing the nonce and, when needed, the extranonce-derived merkle root). Validation recomputes the same deterministic function. A candidate satisfies proof of work when its MercaHash V2 value is no greater than the target encoded by **nBits**.

$$PoW\ valid \Leftrightarrow uint256(MercaHashV2(header)) \leq Target(nBits)$$

1.3 Why a private mutable scratchpad?

Each active hashing worker owns a **16 MiB** mutable scratchpad. This is not a shared read-only lookup table: the header-derived memory is filled, bound across 4 KiB blocks, and subsequently overwritten as the 16-lane state evolves. Memory capacity, irregular access latency, bandwidth, and serial dependencies therefore contribute to throughput.

Important distinction. A memory-hard design is not ASIC-proof or GPU-proof. Compared with a larger-memory profile, the 16 MiB configuration lowers per-worker memory requirements; it does not establish a particular CPU, GPU, FPGA, or ASIC performance ratio.

2. Frozen V2 workload and algorithm pipeline

2.1 Authoritative parameters

Parameter	MercaHash V2
Input header	80 bytes
Output	32 bytes / 256 bits
Scratchpad	16 MiB private mutable memory
Scratchpad line	64 bytes = 8 × 64-bit words
Scratchpad lines	262,144 = 2^{18}
Binding block	4 KiB = 64 lines
Binding blocks	4,096 = 2^{12}
Binding passes	1
Working state	16 × 64-bit lanes (1,024 bits)
Main mixing steps	16,384 = 2^{14}
Secondary touch	Every 16 steps → 1,024 touches
Cross-lane mix	Every 64 steps → 256 events
SHA3-512 checkpoint	Every 1,024 steps → 16 checkpoints
Final sampling	256 scratchpad reads
Finalization	Domain-separated SHA3-256

$$N_lines = 16{,}777{,}216\ bytes \div 64\ bytes = 262{,}144\ lines$$

2.2 Execution pipeline

Stage	Name	Purpose
1	Initial state	Two domain-separated SHA3-512 digests of the header form sixteen little-endian 64-bit lanes.
2	Scratchpad fill	Generate all 2^{18} lines while evolving state; alternate ARX column and diagonal half-rounds.
3	Binding pass	Rewrite each 4 KiB block from current data, previous-block data and an earlier reference.
4	Main mixing	Execute 2^{14} data-dependent read/modify/write steps and periodic secondary touches.
5	Periodic mixing	Run full-state ARX mixing every 64 steps and SHA3-512 checkpoints every 1,024 steps.
6	Final sample	Perform 256 state-dependent reads and mix their contents back into the state.
7	Final hash	SHA3-256 of V2 final domain, original header and final 16-lane state.

3. State initialization, ARX mixing, fill and binding

3.1 Header-derived state

The working state contains sixteen unsigned 64-bit lanes. MercaHash V2 derives the first eight lanes from **SHA3-512("MercaHash-v2/state/0" || header)** and the second eight from **SHA3-512("MercaHash-v2/state/1" || header)**. Every word is decoded explicitly in little-endian form.

$$S[0..7] = LE64(SHA3\text{-}512(D_0 \parallel header)); S[8..15] = LE64(SHA3\text{-}512(D_1 \parallel header))$$

3.2 The MercaARX primitive

Most non-cryptographic diffusion uses addition-rotation-XOR (ARX) operations on unsigned 64-bit words. A four-word G transform is applied in columns and then diagonals. Unsigned addition wraps modulo 2^{64}, yielding deterministic arithmetic on conforming platforms.

$$a \leftarrow a + b;\ d \leftarrow ROTL(d \oplus a, 32);\ c \leftarrow c + d;\ b \leftarrow ROTL(b \oplus c, 24)$$

$$a \leftarrow a + b;\ d \leftarrow ROTL(d \oplus a, 16);\ c \leftarrow c + d;\ b \leftarrow ROTL(b \oplus c, 63)$$

3.3 Scratchpad fill

Before memory-hard mixing, V2 deterministically fills the entire 16 MiB scratchpad. For each 64-byte line, its index perturbs the state, alternating between ARX column and diagonal half-rounds. Eight words are emitted from XOR combinations of the lower and upper state halves, and each new line feeds back into multiple lanes. Later lines depend on the evolving history.

3.4 One memory-binding pass

V2 then rewrites the scratchpad in **4,096 blocks of 4 KiB** (64 lines per block). For block b, the algorithm derives a reference seed from live state, block index and the block's first line. For blocks after the first, the pseudorandom reference is constrained to an earlier block. The previous block is also read, and each line is recomputed from current, previous and reference material.

$$reference_block(b) < b, \text{ for } b > 0$$

This stage introduces sequential inter-block dependencies before the main random-access loop. The scratchpad is header-specific and mutable, rather than a table that miners could generate once and reuse across different candidates.

> **Why this matters.** Fill creates candidate-specific memory; binding links earlier and later regions; the main mix then repeatedly mutates selected lines. These properties raise the cost of shortcuts but do not prove they are impossible.

4. Main data-dependent mixing

4.1 Primary step: one mutable random touch per step

MercaHash V2 executes **16,384 serial primary steps**. At step t, a live-state-derived 64-bit expression chooses one of 2^{18} scratchpad lines. The power-of-two line count allows a bit mask instead of division.

$$i_p = (S_0 \oplus ROTL(S_5,17) \oplus S_{10} \oplus ROTL(S_{15},41) \oplus t \cdot C_0) \;\&\; (2^{18} - 1)$$

The chosen line is read, mixed into both halves of the sixteen-lane state, transformed, and written back at the same address. The resulting state and written memory influence later addresses, so the main loop is not a set of independent lookups.

4.2 Secondary touch every 16 steps

Every sixteenth primary step triggers a second state-dependent read/modify/write. Its index also incorporates part of the newly written primary line. If the two addresses coincide, V2 flips the low index bit to force distinct accesses. The frozen workload performs **1,024** such secondary touches.

4.3 Cross-lane mixing every 64 steps

After eligible secondary touches, every 64th step injects step-dependent material and forces two complete MercaARX rounds across the state. This couples all sixteen lanes and distributes memory observations into future address generation. V2 performs **256** events.

4.4 SHA3-512 checkpoint every 1,024 steps

Every 1,024 steps, the checkpoint hashes a V2 domain string, step number, primary and secondary line indices, all sixteen state lanes, and the eight words most recently written by each of the two memory operations. Its digest is folded into the state and followed by an ARX round.

$$C_t = SHA3\text{-}512(Dcp \parallel t \parallel i_p \parallel i_s \parallel S \parallel W_p \parallel W_s)$$

V2 performs **16** checkpoints. The periodic cryptographic compression complements the comparatively lightweight ARX and memory operations.

Event	Frequency	V2 count
Primary mutable access	Every step	16,384
Secondary mutable access	Every 16 steps	1,024
Full cross-lane event	Every 64 steps	256
SHA3-512 checkpoint	Every 1,024 steps	16

5. Final sampling, output, and mining behavior

5.1 Final scratchpad sampling

After the main loop, MercaHash V2 performs **256 additional state-dependent reads** from the final, mutated scratchpad. Each index is derived from the current state and sample counter using the 18-bit line mask. Read data updates both halves of the state; a full ARX round follows every sixteenth sample.

$$i_sample = (S_0 \oplus ROTL(S_5,17) \oplus S_{10} \oplus ROTL(S_{15},41) \oplus sample \cdot C_0) \;\&\; (2^{18} - 1)$$

5.2 Domain-separated finalization

The final 256-bit result is SHA3-256 over **"MercaHash-v2/final"**, the original 80-byte header, and all sixteen state lanes serialized as little-endian 64-bit words. The scratchpad is not appended in full: its influence is absorbed through the fill, binding, mixing, checkpoints and sampling stages.

$$H_PoW = SHA3\text{-}256(Dfinal \parallel header \parallel LE64(S_0) \parallel \ldots \parallel LE64(S_{15}))$$

5.3 Memory and throughput implications

A typical hashing worker allocates one 16 MiB scratchpad and reuses it across successive candidate nonces. Allocation occurs outside the consensus hash so memory-management overhead need not be part of each hashing operation. Miners should choose worker counts according to their CPUs, GPUs, memory bandwidth, latency and system responsiveness.

$$Scratchpad\ memory \approx 16\ MiB \times active\ hashing\ workers$$

For example, **six workers need about 96 MiB** in scratchpad allocations, before process, node, graphics runtime and operating-system overhead. Device memory capacity alone does not determine optimal hashrate.

5.4 Reference development performance

A controlled standalone cloud measurement reported **62.935 H/s with one worker** and **119.332 H/s with two workers** over three short five-second cycles. These were synthetic work benchmarks, not integrated wallet or mainnet throughput measurements; other hardware, compilers, worker counts and thermals can produce very different rates. [S3]

5.5 Mining and validation must be identical

MercaHash V2 is a consensus function. Miners and validators must compute the same frozen workload for the same 80-byte header. The public function enforces exact header, scratchpad and output sizes. Header synchronization must verify actual MercaHash V2 proof of work rather than trusting chainwork claims or the SHA256d block identifier.

Operational consequence. Changing scratchpad geometry, mixing counts, rotations, constants, binding, checkpoint schedule, serialization, or finalization would change the proof-of-work consensus rule. A faster miner must preserve the exact V2 digest for every candidate.

6. Security rationale, limitations, and implementation rules

6.1 What MercaHash V2 is trying to make expensive

- **Memory per concurrent worker:** 16 MiB of mutable scratchpad storage per active hash context.
- **Irregular mutable access:** addresses depend on evolving state and selected lines are overwritten.
- **Serial dependence:** later state and addresses incorporate earlier reads, writes and bound blocks.
- **Cross-state coupling:** complete ARX rounds periodically mix all state lanes.
- **Cryptographic checkpoints:** SHA3-512 incorporates indices, state and recent memory writes.
- **Memory recomputation:** per-header fill and inter-block binding discourage a static shared table.

6.2 What MercaHash V2 does not claim

MercaHash V2 is not mathematically ASIC-proof or GPU-proof. Specialized implementations may exploit instruction throughput, memory scheduling, parallel contexts, hardware bandwidth or custom logic. Whether these create centralizing economic advantages must be evaluated through reproducible measurements and real mainnet observation, not the algorithm's label.

6.3 Determinism and consensus safety

Consensus evaluation uses fixed-width unsigned integers, explicit rotations, explicit little-endian serialization, fixed iteration counts and V2-specific SHA3 domain strings. Floating-point arithmetic and undefined or platform-dependent behavior are excluded. Every conforming node must return the identical 256-bit result for the same header.

- Use the frozen **mercahash::HashV2** function and exactly 16 MiB of scratchpad in consensus execution.
- Maintain permanent known-answer vectors, independent CPU/GPU parity checks and cross-platform tests.
- Preserve SHA256d for block identity and MercaHash V2 for proof-of-work comparison.

6.4 Design trade-off

V2 emphasizes deterministic, reviewable memory-and-state mixing without the virtual-machine or JIT complexity of randomized-instruction designs. The 16 MiB choice makes parallel contexts less memory-intensive, while also reducing the capacity burden on specialized hardware. Its security posture depends on the complete algorithm, implementation quality and continuing analysis.

Conclusion

MercaHash V2 combines a header-specific mutable scratchpad, sequential binding, serial data-dependent writes, periodic cross-lane diffusion, SHA3 checkpoints and deterministic final sampling. **For Mercatura mainnet, V2 is the proof-of-work design from genesis.** Release implementations must match the frozen algorithm and validated genesis configuration exactly.

Technical basis

[S1] Mercatura Core src/crypto/mercahash_v2.cpp — authoritative V2 execution and domain separation. **[S2]** src/crypto/mercahash.h and src/pow.cpp — V2 constants, entry point and proof-of-work comparison. **[S3]** doc/mining/MERCAHASH-V2-CLOUD-VALIDATION.md — controlled development benchmark. Mainnet-at-genesis wording expresses the approved launch specification; source configurations and final release artifacts require matching verification.