A Memory-Hard Proof-of-Work Cryptocurrency

Version 1.0 · June 2026

Version 1.0 · June 2026

Mraksoll — Lead Maintainer
contact@dpowcore.org
https://dpowcore.org

Abstract

Dpowcoin Core (DPC) is an open-source cryptocurrency derived from Bitcoin Core 0.26.x, distinguished by two design decisions: replacement of SHA256d proof-of-work with a SHA-512-salted, dual-round Argon2id construction — a memory-hard, ASIC-resistant hashing function — and adoption of the LWMA-3 per-block difficulty algorithm with a five-minute target. From genesis through block 224,999 the network additionally required a second, independent proof-of-work check (Yespower) sharing Argon2id's target; this transitional dual-PoW requirement was dropped from consensus at block 225,000 (Dpowcoin Core v26.3.1), after which Argon2id alone secures the chain. The codebase follows Bitcoin Core's development process including the full functional test suite, reproducible GNU Guix builds, and GPG-signed release attestations. This paper describes the technical parameters, the rationale behind each design choice, and the toolchain used to bootstrap the network.

Table of Contents

---

1. Introduction

Bitcoin's proof-of-work algorithm (SHA256d) was initially mined on consumer CPUs, then GPUs, and eventually displaced entirely by application-specific integrated circuits (ASICs). ASIC manufacturing concentrates hash rate among a small number of hardware producers and large mining operations, reducing the decentralization that proof-of-work is intended to enforce.

Argon2id, winner of the Password Hashing Competition (2015), was designed to be simultaneously time-hard and memory-hard. Its memory requirement makes it resistant to the parallel architectures that give ASICs their advantage over general-purpose hardware, and its sequential memory access pattern penalises GPU implementations relative to CPU ones.

Dpowcoin Core applies a SHA-512-salted, dual-round Argon2id construction (§3.2–3.3) as the proof-of-work challenge function, while retaining SHA256d for all internal chain data structures — the same separation used by Litecoin (Scrypt PoW / SHA256d chain). This preserves full compatibility with Bitcoin's UTXO model, transaction format, and wallet infrastructure while making block production accessible to commodity hardware. From genesis through block 224,999 this Argon2id check ran alongside a second, independent Yespower check under a shared-target dual proof-of-work scheme; §3.6 covers the rationale for consolidating to Argon2id alone at block 225,000.

---

2. Codebase and Development Process

2.1 Upstream Base

Dpowcoin Core is a fork of Bitcoin Core 0.26.x and tracks upstream releases. Protocol changes, wallet improvements, and security fixes from Bitcoin Core are reviewed and merged as they are published. The project commits to staying within two major version series of the Bitcoin Core upstream.

2.2 Development Standards

The development process mirrors Bitcoin Core:

- **Peer review.** All changes are submitted as GitHub pull requests and require review before merging. There is no privileged developer class; contributors earn trust through merit.
- **Functional test suite.** The full Bitcoin Core functional test suite is inherited and extended. Consensus-critical changes require passing all tests, including the regression test framework (`regtest` chain).
- **Reproducible builds.** Release binaries are built using GNU Guix for deterministic output. Builder attestations and signing keys are published at github.com/dpowcore-project/guix.sigs.
- **GPG-signed releases.** Every release is signed by the lead maintainer's GPG key (`23B3 D882 F805 A5D1 F0A1 2A25 35A7 538B 1C81 6E49`). Users are expected to verify signatures before running any binary.
- **Security policy.** Vulnerabilities are reported to security@dpowcore.org under responsible disclosure. The project maintains security updates for the two most recent major release series, following the Bitcoin Core security policy model.

---

3. Proof-of-Work: Argon2id

3.1 Algorithm Selection

Argon2 is defined in RFC 9106 and exists in three variants:

Variant	Parallelism	Data-dependence	Recommended use
Argon2d	Single-pass	Data-dependent	Cryptocurrency PoW (original)
Argon2i	Single-pass	Data-independent	Password hashing
Argon2id	**Hybrid**	**Both**	**General-purpose — RFC recommendation**

Argon2id was chosen over Argon2d since it has wide support on all platforms and programming languages in the form of ready-made libraries, unlike the I and D variants.

3.2 Consensus Parameters

Dpowcoin Core does not call Argon2id once per header — it chains **two** Argon2id passes, each salted independently. The salt for round 1 is derived from a double SHA-512 digest of the header; the salt for round 2 is round 1's output. All parameters below are consensus-critical and are encoded directly in `block.cpp`; any change requires a hard fork.

Parameter	Round 1	Round 2	Notes
Variant	`Argon2_id`	`Argon2_id`	
Time cost (`t`)	`2`	`2`	Passes over memory, per round
Memory cost (`m`)	`4,096 KiB` (4 MiB)	`32,768 KiB` (32 MiB)	Round 2 is 8× round 1
Parallelism (`p`)	`2`	`2`	Lanes per round

Output length	`32 bytes`	`32 bytes`	Intermediate hash / final PoW hash
Password	80-byte serialized header	80-byte serialized header	Identical in both rounds
Salt	`SHA-512(SHA-512(header))`	Round 1 output (`h1`)	64-byte salt in round 1; 32-byte salt in round 2

Round 1's output is not itself a candidate PoW hash — it exists only to serve as round 2's salt. Round 2's output is the value compared against `nBits`.

Doubling the SHA-512 digest before using it as round 1's salt, rather than salting directly with the header, adds a cheap but consensus-binding step ahead of the expensive Argon2id work: any change to the header changes this salt, so no salt can be precomputed independently of a specific header. The 64-byte double-SHA-512 output comfortably exceeds Argon2's 8-byte minimum salt length. The 80-byte header length matches Bitcoin's block header serialization (4 version + 32 prevHash + 32 merkleRoot

- 4 time + 4 nBits + 4 nNonce).

3.3 Hash Construction

```
// From block.cpp — consensus-critical, must not be changed
CDataStream ss(SER_NETWORK, PROTOCOL_VERSION);
ss << *this;                                    // 80-byte serialized header

// Round-1 salt: double SHA-512 over the serialized header
uint8_t salt[CSHA512::OUTPUT_SIZE];
CSHA512 sha512;
sha512.Write(&ss[0], ss.size()).Finalize(salt);
sha512.Reset().Write(salt, sizeof(salt)).Finalize(salt);

// Round 1: header as password, double-SHA-512 digest as salt
uint256 h1;
argon2id_hash_raw(/*t_cost=*/2, /*m_cost=*/4096, /*parallelism=*/2,
                  &ss[0], ss.size(), salt, sizeof(salt), &h1, 32);

// Round 2: header as password, round-1 output as salt
uint256 h2;
argon2id_hash_raw(/*t_cost=*/2, /*m_cost=*/32768, /*parallelism=*/2,
                  &ss[0], ss.size(), &h1, 32, &h2, 32);

return h2;   // compared against nBits
```

3.4 Chain Hash Separation

Block identifiers (used in `hashPrevBlock`, Merkle tree roots, and all chain linkage) are computed with SHA256d, not Argon2id. Argon2id serves exclusively as the proof-of-work challenge. This is the same separation used by Litecoin (Scrypt PoW / SHA256d chain data) and ensures that:

- Chain indexing and validation remain computationally cheap.
- Existing block explorer and wallet infrastructure is compatible without modification.
- The PoW challenge can in principle be changed in a future hard fork without altering chain data structures.

3.5 Memory Hardness and ASIC Resistance

Because round 2 cannot start until round 1's output is known — it is round 2's salt — a single PoW attempt requires filling and reading 4 MiB (round 1) and then 32 MiB (round 2) of working memory, 36 MiB in total. Argon2's internal Blake2b compression rounds create data dependencies between memory blocks, so this memory cannot be computed in parallel sub-units within a round — it must be traversed sequentially. This eliminates the area reuse that makes SHA256d ASICs economically viable: an ASIC targeting these parameters requires the same ~32 MiB SRAM per hashing core (dominated by round 2) as a general-purpose CPU, removing the area advantage ASICs ordinarily exploit.

At `t = 2` passes per round, each round sweeps its own buffer twice. Chaining two independently-sized rounds, rather than one larger round, also means an attacker cannot skip round 1 to save memory: round 1's output is consensus-required as round 2's salt, so both rounds must be computed in full, in order, for every hash attempt — raising the cost of time-memory trade-off (TMTO) attacks relative to a single-round construction of equivalent total work.

3.6 Dual Proof-of-Work Era and the Yespower Drop

From genesis through block 224,999, Dpowcoin Core additionally required every header to pass a second, independent proof-of-work check using Yespower (`N = 2048`, `r = 8`), sharing the same target as Argon2id: a header was valid only if both `Yespower(header) ≤ target(nBits)` and `Argon2id(header) ≤ target(nBits)`. Yespower was checked first as a cheap filter; the more expensive Argon2id check (§3.2–3.3) only ran if Yespower passed.

Because both algorithms were held to the same target, Yespower added no security beyond what Argon2id already provided on its own, while doubling the work the LWMA-3 retargeting algorithm (§5) had to account for and adding consensus-code complexity for no measurable benefit. The Yespower check was independent of difficulty adjustment and chain-weight accounting, so dropping it changes neither. The dual-PoW design was worth trying; in practice it added cost without adding security. The "Dual PoW" name is retained in project documentation for historical reference only — to be explicit, this is unrelated to, and does not use, "delayed proof-of-work" (dPoW) technology, and Dpowcoin Core is not affiliated with Komodo or its projects.

Era	Block range	PoW check(s) required
Dual PoW Era	0 – 224,999	Yespower **and** Argon2id (shared target)
Transition	225,000	Yespower requirement removed from consensus code
Argon2id-Only Era	225,000+	Argon2id only

The Yespower requirement was removed from consensus starting with Dpowcoin Core v26.3.1. No action is required from miners or SPV wallets: since Yespower and Argon2id always shared the same target, mining software already built on Argon2id continues to work unchanged across the transition, and light clients can rely on Argon2id alone for header verification both before and after block 225,000.

---

4. Implementation: ISA Dispatcher

The internal fill-segment function — the innermost loop of Argon2's memory-filling phase — is performance-critical. Dpowcoin Core implements the same ISA-dispatch pattern used by Bitcoin Core for SHA256: a runtime CPU feature detector selects the fastest available implementation.

4.1 Available Backends

Backend	Compiled with	Condition
`fill_segment_ref`	(always)	Pure-C reference; safe fallback
`fill_segment_sse2`	`-msse2`	x86: SSE2 detected at runtime
`fill_segment_ssse3`	`-mssse3`	x86: SSSE3 detected at runtime
`fill_segment_avx2`	`-mavx2`	x86: AVX2 + OS XSAVE enabled
`fill_segment_avx512`	`-mavx512f`	x86: AVX-512F + OS XSAVE enabled
`fill_segment_neon`	`-mfpu=neon`	AArch64: NEON flag in `use_implementation`

4.2 Dispatcher Mechanism

A global function pointer `argon2_fill_segment` defaults to `fill_segment_ref` at startup. `Argon2AutoDetectImpl()` is called once at node startup. It performs CPUID detection and, on x86, verifies OS XSAVE support before upgrading the pointer to the fastest available SIMD backend. On AArch64, NEON is selected when available.

This design means every node automatically uses the most efficient implementation available on its hardware without requiring separate binary distributions per CPU microarchitecture.

4.3 Header Proof-of-Work Cache

Because Argon2id is deliberately expensive (§3.5), and the same header's proof-of-work is legitimately re-checked at several independent points during normal operation — during the headers-presync anti-DoS pass, again in `CheckBlockHeader()` on header acceptance, again on full block acceptance, and again on any later disk re-read — Dpowcoin Core caches the *result* of each successful check rather than repeating the double Argon2id computation (§3.2–3.3) every time.

The cache (`HeaderPoWCache`) is a `CuckooCache::cache<uint256, ...>` — the same cuckoo-filter-

based cache structure Bitcoin Core uses for its signature cache — keyed on the header's cheap SHA256d hash (`GetHash()`), not on the expensive Argon2id result:

Property	Value
Backing structure	`CuckooCache::cache`
Cache key	`header.GetHash()` (SHA256d)
Cached entries	Passing headers only — failed checks are never cached
Default size	64 MiB (2,097,152 entries at 32 bytes/entry)
Override	`-headerpowcachesize=<MiB>`
Concurrency	`shared_mutex` — concurrent reads; writes are exclusive
Scope	One process-lifetime instance, shared by every call site

A cache **miss** always falls back to a full, honest Argon2id recomputation — behaviour on miss is identical to having no cache at all — so the cache can only remove redundant work; it can never weaken validation. Because only positive results are ever cached, a hit can only mean "this exact header content already passed proof-of-work verification at least once in this process." This also makes the cache bidirectional across sync phases: a header verified during the presync anti-DoS pass is already cached by the time the same header is re-sent in full during redownload, so the second check hits the cache instead of recomputing both Argon2id rounds.

4.4 Parallel Header Validation

Header proof-of-work checks are independent of one another — validating header *N* does not require knowing whether header *N-1* passed — so a batch of headers received from a peer can have its PoW checked across multiple threads at once, ahead of the necessarily-sequential chain-linkage and chainwork accounting that follows.

Parameter	Value
Parallel-dispatch threshold	32 headers
Max worker threads	6
Worker thread count	`clamp(cores > 1 ? cores - 1 : cores, 1, 6)`, minus the calling thread
Dispatch mechanism	`CCheckQueue<CHeaderPoWCheck>` — the same queue pattern used for parallel script verification

Batches smaller than the threshold are checked sequentially on the calling thread, since dispatching to a queue has overhead that isn't worth paying for a handful of headers. At or above the threshold, each header's check is wrapped in a `CHeaderPoWCheck` (which itself goes through the cache described in §4.3) and handed to the queue. One CPU core is always left free for the rest of the node when more than one core is available, so this never starves a small box such as a single-board computer. This path is exercised most heavily during the headers-presync stage, where a new peer may send tens of thousands of headers in a single anti-DoS batch before any block data is downloaded.

This mechanism is adapted from the reference dual-threaded header-validation implementation and is slated to ship as part of Dpowcoin Core's transition to the Bitcoin Core 31.2 base (CMake build system).

---

5. Difficulty Algorithm: LWMA-3

5.1 Rationale for Replacing Bitcoin's Retargeting

Bitcoin adjusts difficulty once every 2,016 blocks (approximately two weeks). This large window was appropriate for a network with stable hash rate, but creates significant instability when hash rate changes rapidly — a common occurrence for smaller networks where a single pool can represent a significant fraction of total hash rate. A 2,016-block window also means the network is effectively unable to respond to hash rate changes for up to two weeks.

Dpowcoin Core uses LWMA-3 (Linearly Weighted Moving Average, revision 3), designed by Zawy with contributions from the Bitcoin Gold and Zcash communities. LWMA-3 retargets on every block, using a weighted average of the N most recent solve times where more recent blocks receive linearly higher weight. This allows the network to track hash rate changes within hours rather than weeks.

5.2 Parameters

Parameter	Value	Notes
Target block time T	300 s (5 min)	Between LTC (150 s) and BTC (600 s)
Averaging window N	576 blocks	
Retarget	Every block	

The 5-minute block time was chosen to place Dpowcoin between Litecoin's 2.5 minutes and Bitcoin's 10 minutes. Confirmation latency is half that of Bitcoin while the longer interval reduces orphan rate on variable-latency network paths.

6. Timing and Anti-Manipulation Parameters

6.1 Future Time Limit (FTL)

```
// chain.h
static constexpr int64_t MAX_FUTURE_BLOCK_TIME = 600; // 2 × target block time
```

A block is rejected if its timestamp exceeds the node's current time by more than 600 seconds. Setting FTL to $2T$ (two block intervals) is standard practice for time-warp attack mitigation. The original Bitcoin Core value was 7200 s (two hours); Dpowcoin Core reduces this to limit the window in which a miner can manipulate timestamps to artificially lower difficulty.

6.2 Timestamp Window

```
        static constexpr int64_t TIMESTAMP_WINDOW = 2 * 60 * 60; // 2 hours
```

The grace period used when validating external timestamps (RPC parameters, wallet key creation times) against block timestamps. This remains at two hours to preserve compatibility with wallet software that computes key creation time from block timestamps.

6.3 Catching-Up Gap

```
        static constexpr int64_t MAX_BLOCK_TIME_GAP = 90 * 60; // 90 minutes
```

The threshold at which the GUI switches to "Catching up..." mode. At a 5-minute block time, 90 minutes represents 18 blocks — enough to distinguish a genuine sync lag from normal network variance.

6.4 Summary

Constant	Value	Ratio to T
`MAX_FUTURE_BLOCK_TIME`	600 s	2T
`TIMESTAMP_WINDOW`	7200 s	24T
`MAX_BLOCK_TIME_GAP`	5400 s	18T

7. Monetary Policy

7.1 Supply Cap

The maximum supply is capped at `42,000,000 DPC` in the consensus parameters. The subsidy function uses integer right-shift identical to Bitcoin Core's `GetBlockSubsidy()` implementation. Actual issuable supply across all 33 halving eras is **41,999,999.9538 DPC** (4,199,999,995,380,000 satoshis — the C++ verifiable nSum value).

7.2 Block Reward and Halving

```
Initial block reward : 50 DPC  (5,000,000,000 satoshis)
Halving interval     : 420,000 blocks
Halving period       : ≈ 4.0 years at 300 s/block
```

The reward for block `n` in satoshis:

```
subsidy(n) = (50 × COIN) >> floor(n / 420000)
COIN = 100,000,000
```

This mirrors Bitcoin Core exactly. The right-shift produces precise fractional DPC values

(12.5, 6.25, 3.125…) unlike floor division of integer DPC amounts.

7.3 Halving Schedule

Era	Blocks	Reward (DPC)	Era Supply (DPC)	Cumulative (DPC)	Approx. Year
0	0 – 419,999	50	21,000,000	21,000,000	2026–2030
1	420,000 – 839,999	25	10,500,000	31,500,000	2030–2034
2	840,000 – 1,259,999	12.5	5,250,000	36,750,000	2034–2038
3	1,260,000 – 1,679,999	6.25	2,625,000	39,375,000	2038–2042
4	1,680,000 – 2,099,999	3.125	1,312,500	40,687,500	2042–2046
5	2,100,000 – 2,519,999	1.5625	656,250	41,343,750	2046–2050

Series continues for 27 additional eras. Total converges to **41,999,999.9538 DPC** (verifiable nSum: 4,199,999,995,380,000 satoshis).

7.4 Coinbase Maturity

A standard coinbase output becomes spendable after `100` blocks (approximately 8.3 hours).

---

8. Network Parameters

8.1 Ports and Magic

Parameter	Value
P2P Port (mainnet)	`42003`
RPC Port (mainnet)	`42002`
Network magic	`0xf29f4afb`
BIP-324 shared secret salt	`dpowcoin_v2_shared_secret`
BIPs active from	Block 1

8.2 Address Formats

Type	Prefix	Version byte	Encoding
P2PKH	`P…`	`55` (0x37)	Base58Check
P2SH	`C…`	`28` (0x1c)	Base58Check
P2WPKH / P2WSH	`dpc1q…`	—	Bech32
P2TR (Taproot)	`dpc1p…`	—	Bech32m

All SegWit and Taproot address types inherited from Bitcoin Core are fully supported and active from block 2.

---

9. Consensus-Level Security Patches

Dpowcoin Core incorporates several security patches applied at the consensus layer, beyond what was present in the inherited Bitcoin Core base.

9.1 BIP53 — 64-Byte Transaction Rejection

```
// tx_check.cpp
if (!tx.IsCoinBase() && ::GetSerializeSize(TX_NO_WITNESS(tx)) == 64)
    return state.Invalid(TxValidationResult::TX_CONSENSUS, "bad-txns-64byte",
                         "tx serialized size is exactly 64 bytes (BIP53)");
```

Transactions serialized to exactly 64 bytes (without witness) are rejected at the consensus level. Such transactions create ambiguity in the Merkle tree: a 64-byte transaction serialization can be confused with an internal Merkle node, enabling certain Merkle proof forgeries. Coinbase transactions are exempt because a 64-byte coinbase would require approximately 224 bits of work to construct and is therefore practically impossible to produce maliciously.

9.2 CVE-2018-17144 — Duplicate Input Detection

Transactions are checked for duplicate inputs before any UTXO lookup. The original vulnerability could result in coin inflation if a transaction spending the same output twice passed validation in certain caching conditions.

---

10. Conclusion

Dpowcoin Core applies two targeted modifications to Bitcoin Core: Argon2id as a memory-hard proof-of-work function, and LWMA-3 per-block difficulty retargeting calibrated to a five-minute block interval. All other Bitcoin Core behaviour — transaction format, UTXO model, SegWit, Taproot, BIP-324 transport encryption, reproducible builds, and the full test suite — is preserved.

The design prioritises verifiability. Every consensus-critical parameter is embedded directly in source code and documented here. Every release binary is independently reproducible via GNU Guix and GPG-signed. The development process is open: there is no privileged developer class, and all changes go through public peer review.

Source: github.com/dpowcore-project/dpowcoin
Website: dpowcore.org

Appendix A — Network Parameter Reference

Parameter	Value
Ticker	DPC
Max supply (params)	42,000,000 DPC
Issuable supply (satoshi precision)	41,999,999.9538 DPC
nSum (C++ verifiable)	4,199,999,995,380,000 sat
Initial block reward	50 DPC
Halving interval	420,000 blocks
Halving period	≈ 4.0 years
Target block time	300 s (5 min)
Coinbase maturity (standard)	100 blocks (~8.3 h)
PoW algorithm	Argon2id (dual-round, SHA-512-salted)
Argon2id round 1 `t` / `m` / `p`	2 / 4,096 KiB / 2
Argon2id round 2 `t` / `m` / `p`	2 / 32,768 KiB / 2
Dual-PoW era (Yespower + Argon2id)	Blocks 0 – 224,999
Yespower dropped from consensus	Block 225,000 (v26.3.1)
Header PoW cache size (default)	64 MiB (`-headerpowcachesize`)
Header PoW parallel-check threshold	32 headers
Header PoW check worker threads (max)	6
Difficulty algorithm	LWMA-3
LWMA window N	576 blocks
Retarget	Every block
Max future block time (FTL)	600 s
PoW limit	`001fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff` or `0x1f1fffff`
Genesis block hash	`d86f8a0582e779830f182befeaaabc8c73a159b6b06530910758daf17ce31e36`
P2P port (mainnet)	42003
RPC port (mainnet)	42002
Network magic	0xf29f4afb
BIP-324 salt	`dpowcoin_v2_shared_secret`
P2PKH prefix	P (dec 55, 0x37)

P2SH prefix	C (dec 28, 0x1C)
Bech32 HRP	`dpc`
SegWit / Taproot	Active from block 2
BIPs	Active from block 2
License	MIT

Appendix C — Halving Schedule

Era	Block range	Reward (DPC)	Era supply (DPC)	Cumulative (DPC)	Target year
0	0 – 419,999	50	21,000,000	21,000,000	2026–2030
1	420,000 – 839,999	25	10,500,000	31,500,000	2030–2034
2	840,000 – 1,259,999	12.5	5,250,000	36,750,000	2034–2038
3	1,260,000 – 1,679,999	6.25	2,625,000	39,375,000	2038–2042
4	1,680,000 – 2,099,999	3.125	1,312,500	40,687,500	2042–2046
5	2,100,000 – 2,519,999	1.5625	656,250	41,343,750	2046–2050
6	2,520,000 – 2,939,999	0.78125	328,125	41,671,875	2050–2054
7	2,940,000 – 3,359,999	0.390625	164,062.5	41,835,937.5	2054–2058
…	…	…	…	…	…
33	13,440,000 – 13,859,999	0.00000001	0.0042	41,999,999.9538	~2162

Reward halves every 420,000 blocks. Series runs 33 eras until reward = 1 satoshi (0.00000001 DPC). Total issuable: **41,999,999.9538 DPC** (nSum: 4,199,999,995,380,000 satoshis).