CONTENTS

NEXON

A Mined ERC-20 with a Self-Hook

V1.0 / MAY 2026 / BASE · CHAINID 8453

ABSTRACT

EXON is an ERC-20 token released entirely through proof of work, with no team allocation, no insider presale, no admin keys, and no upgrade path. The token contract

is also its own Uniswap V4 swap hook and its own miner. **One bytecode, one address. Once deployed, the rules cannot change.**

The supply is 21 million tokens, identical to Bitcoin in scale. Five percent funds an open genesis sale, five percent seeds a Uniswap V4 NEXON/ETH pool whose liquidity is locked forever by the same hook that collects a 1% swap fee. The remaining ninety percent is mined: anyone with a browser can solve a keccak256 puzzle bound to their wallet address and call `mine()` to receive the current era's reward.

This document describes why the design exists, how each piece works, and what limits remain.

WHY THIS EXISTS

Most ERC-20 launches today are pre-mints. A team deploys a contract, mints the supply to a multisig, and decides how to distribute it later. The "fair launch" variants typically replace the multisig with a vesting contract, which is just a slower version of the same problem.

A different model has existed since 0xBitcoin in 2018: **the contract refuses to mint to anyone.** New tokens emerge only when someone solves a hash puzzle on chain. The supply schedule is mechanical, the distribution is permissionless, and the team has no special power because there is no team-controlled state to abuse.

NEXON takes that model and adds two pieces that 0xBitcoin and its imitators could not have. The first is **address-bound proofs of work**: a mined nonce only works for the wallet that found it, so solutions cannot be copied from the mempool. The second is a **self-hook**, made possible by Uniswap V4 hooks shipping to mainnet in January 2025. The token contract is also the swap hook for its own liquidity pool, which lets the pool reject every liquidity-modification call and effectively lock the LP without any external service.

ADDRESS-BINDING KILLS FRONT-RUNNING AT THE CRYPTOGRAPHIC LEVEL. THE SELF-HOOK REMOVES THE NEED TO TRUST.

Together they remove the two most common forms of soft-rug that survive even in supposedly fair launches.

HOW THE CONTRACT IS ORGANIZED

The contract moves through four phases in its lifetime.

Phase 0, Empty. The contract is deployed and holds nothing. Constructor parameters set the addresses of the Uniswap V4 PoolManager, PositionManager, and Permit2 for the target chain. The deployer becomes the `controller`, the only address with permission to claim accumulated swap fees later. No other state is mutable from outside.

Phase 1, Genesis. Anyone can call `mintGenesis(units)` with `units * 0.01 ETH`. Each unit yields 1,000 NEXON, capped at five units per transaction. The transaction cap exists so a single mempool slot cannot sweep the entire allocation in one go. Excess ETH is refunded in the same call.

Phase 2, Seeding. Once the genesis cap of 1,050,000 NEXON is sold out, anyone can call `seedPool()`. The function mints the remaining 19,950,000 NEXON to the contract itself, creates the V4 NEXON/ETH pool with the contract as its hook, deposits all genesis ETH plus 1,050,000 NEXON as liquidity, and sets the initial mining difficulty. The LP position NFT is minted to the controller. A fallback `partialSeed()` exists for the case where genesis stalls below the cap.

Phase 3, Mining. The remaining 18,900,000 NEXON is released through proof of work. Each successful `mine(nonce)` call pays the current era's reward, starting at 100 NEXON and halving every 100,000 mints. Mining ends when the cap is reached.

Phase transitions are gated by storage flags. **No admin function exists to skip a phase, rewind state, or change parameters.**

THE MINING PROTOCOL

A mined nonce is valid for a wallet at a given epoch when

```sol
keccak256(abi.encode(challenge, nonce)) < currentDifficulty
```

where

```sol
challenge = keccak256(abi.encode(chainId, contractAddress, miner, epoch))
epoch     = blockNumber / 600
```

The challenge changes every six hundred blocks, roughly twenty minutes on Base. A miner who fails to find a solution within one epoch cannot carry the work forward. The seed shifts and the search starts over.

The `miner` field is `msg.sender`. Bob's nonce search is for a different challenge than Alice's. Even if Bob copies Alice's pending transaction from the mempool and rebroadcasts it under his own address, his version computes a different hash and almost certainly fails the difficulty check. **Front-running mined solutions is impossible at the cryptographic level, not at the transaction-ordering level.**

Replay protection lives in a `usedProofs` mapping keyed on `keccak256(miner, nonce, epoch)`. The same triple cannot be claimed twice. Storage grows by one slot per successful mint.

Difficulty retargets every 2,016 mints, the same cadence as Bitcoin. The retarget is clamped to a factor of four in either direction per period. A hard cap of ten mints per block prevents one party with surplus hashpower from sweeping every block.

Starting difficulty after seeding is `type(uint256).max >> 32`, which means roughly one in four billion hashes qualifies. A modern laptop finds a solution in seconds.

THE SELF-HOOK

A Uniswap V4 hook is a contract whose address contains specific bits in its lower fourteen bits. Those bits encode which lifecycle functions the hook actually implements. The same contract

that implements `transfer` and `balanceOf` for the ERC-20 also implements `beforeSwap`, `afterSwap`, and `beforeInitialize` for the pool. To make this work, the contract is deployed via CREATE2 with a salt chosen so the resulting address has the correct permission bits.

After seeding, the pool exists at coordinates: currency0 is native ETH, currency1 is NEXON, tickSpacing is 200, hooks is the NEXON contract itself. The hook intercepts swaps and routes 1% of the ETH side of every swap to the contract's own balance.

The hook also denies every liquidity modification. `beforeAddLiquidity` and `beforeRemoveLiquidity` revert unconditionally. The pool starts with the seed liquidity and stays at that exact shape forever. **Nobody, including the deployer who holds the LP NFT, can move the liquidity out or add more on top of it.**

Accumulated ETH fees sit on the contract until the controller calls `claimFees()`, which transfers the contract's entire ETH balance to the controller. This is the only privileged function in the contract.

THE LOCK IS THE BYTECODE.

TOKENOMICS

ALLOCATION	AMOUNT	SHARE
Genesis sale	1,050,000 NEXON	5%
Locked LP	1,050,000 NEXON	5%
Mining	18,900,000 NEXON	90%
Total	**21,000,000 NEXON**	**100%**

Genesis is priced at 0.01 ETH per 1,000 NEXON, fixed. A fully subscribed genesis raises 10.5 ETH, every unit of which goes directly into the V4 pool as the ETH side of seed liquidity.

Nothing reaches the deployer at genesis.

The mining schedule halves every 100,000 successful mints. Era zero pays 100 NEXON per mint, era one pays 50, era two pays 25, and so on. In practice the schedule terminates at era four because the cumulative reward up to that point reaches the 18.9M mining cap.

SEEDING STRATEGY

Seeding the pool is the only timed decision the controller has to make. Two functions enable it:

```sol
seedPool()    — permissionless, requires genesisMinted == GENESIS_CAP
partialSeed() — controller only, requires block.timestamp ⩾ deployedAt + 30 minutes
```

The unsold genesis NEXON is never minted. Not burned, not held by the controller, not stuck in storage. It simply does not exist.

Three strategies are reasonable: **strict sell-out** (wait for the full cap), **threshold seed** (commit to a public threshold), or **aggressive early seed** (seed within hours regardless). The contract enforces none; the controller's discretion within the time gate is total.

REFUND ESCAPE HATCH

The genesis sale collects ETH on the contract and trusts that one of the seed functions will eventually be called. If neither succeeds, the ETH would otherwise be locked forever. `refundGenesis` exists to cover both: a controller who disappears, and a controller who tries to seed but cannot.

```sol
function refundGenesis(uint256 tokenAmount) external nonReentrant {
    if (genesisComplete)                                revert GenesisAlreadyComplete();
    if (block.timestamp < deployedAt + REFUND_GRACE)    revert RefundGraceNotPassed();
    if (tokenAmount == 0 || tokenAmount % GENESIS_UNIT ≠ 0) revert MustBeUnitMultiple();

    uint256 units   = tokenAmount / GENESIS_UNIT;
    uint256 ethBack = units * GENESIS_PRICE;
```

```
        _burn(msg.sender, tokenAmount);
        genesisMinted    -= tokenAmount;
        genesisEthRaised -= ethBack;

        (bool ok,) = msg.sender.call{value: ethBack}("");
        if (!ok) revert EthTransferFailed();

        emit GenesisRefund(msg.sender, ethBack, tokenAmount);
    }
```

`REFUND_GRACE` is fixed at 3 days. After three days, if seeding has not succeeded, the system effectively becomes a no-op refund market.

VERIFICATION

The contract source is published under the MIT license. The deployed instance on Base mainnet (`0xNEX001…89ABc`) is verified on Basescan. The compiled bytecode, the constructor arguments, and the CREATE2 salt are reproducible from the source.

The mining algorithm is implemented in Rust, compiled to WebAssembly, and runs entirely in the user's browser.

A user who wants to confirm the contract does what this document claims can do three things independently: **first**, check the V4 hook permission bits in the contract address. **Second**, read the constants directly from chain. **Third**, try to remove liquidity from the NEXON/ETH pool through any V4 router; the call reverts with `InvalidAction()`.

AGENT ALIGNMENT

The NEXON contract maps cleanly onto the ERC-8004 agent registry model. Three properties of a "trustless agent" are spelled out in the EIP: **a stable identity, observable reputation, and verifiable behavior.** NEXON satisfies all three without any wrapper layer.

The **identity** is the contract's own address. Fixed at deploy time, derived deterministically from the CREATE2 init-code hash. The address is the agent.

The **reputation** is the contract's on-chain state. Every metric that matters is queryable directly from chain by any indexer.

The **behavior** is verified by the source. The bytecode is the source, the source is the spec, and both are immutable.

NEXON is registered as Agent #74832 on the canonical ERC-8004 Identity Registry on Base mainnet.

OPERATOR SIGIL NFTS

A separate ERC-721 collection, **OperatorSigil**, gives each NEXON participant an on-chain identity in the ERC-8004 sense without modifying the core token contract. The mapping is one NFT per address, claimable once, soulbound after mint.

The artwork is a curated set of ten 1-of-1 pieces, organised as five tiers times two variants, each piece named after a state in a transaction lifecycle: *Genesis Signal, Pending State, Ordered Execution, Verified State, Replay Barrier, Finalized State, Archived State, Echo State, Transition State,* and *Confirmation State.*

The tier scales dynamically with the holder's live NEXON balance — Initiate (under 1,000 NEXON), Bronze (1k–10k), Silver (10k–100k), Gold (100k–1M), Platinum (1M and above) — so a wallet that grows from Silver to Gold visibly upgrades its badge without any on-chain action.

The PNGs are pinned to IPFS as a single CIDv1 dag-pb folder, content-addressed and provably immutable. **Transfers between EOAs revert with `Soulbound()`.**

LIMITS AND CAVEATS

The contract has **no upgrade path**. If a critical bug is discovered post-launch, there is no fix. The mitigation is the same as Bitcoin's: keep the surface small, audit before deploy.

Mining costs gas. On Base, a `mine()` transaction costs cents. As the price of NEXON in the AMM falls below the mining cost, hashrate drops, the network produces fewer than one mint

per minute, and difficulty retargets downward. **Equilibrium price is whatever clears that condition.**

Address-binding makes solutions unstealable from the mempool but does nothing against a miner who controls multiple wallets. This is the same property Bitcoin has.

The 1% swap fee paid to the controller represents an ongoing extraction from swap volume. The fee can never be raised, lowered, or rerouted because the contract is immutable.

NEXON IS NOT A PROMISE. THERE IS NO ROADMAP TO DELIVER AGAINST. THE CONTRACT DOES WHAT ITS BYTECODE DOES, AND NOTHING MORE.

◊ ◈ ◊ END OF DOCUMENT ◊ ◈ ◊
