HEX-CHAIN (HEXK)

A Distributed P2P Attack-Pool that target Centralized Bitcoin & Ethereum Ecosystem

Write by Anonymous
support@hex-chain.org
www.hex-chain.org

#1 Breaking the Rules, Resetting the System

Abstract
HEX-CHAIN represents a paradigm shift in blockchain technology by redirecting computational power from meaningless hash calculations to practical private key discovery and dormant asset recovery. This whitepaper presents a comprehensive mathematical and cryptographic analysis of the HEX-CHAIN ecosystem, including the HEXK token economy, HEX-KEY independent scanner, and HEXK-Pool distributed network. We rigorously prove the computational feasibility, cryptographic security, and economic sustainability of the system through formal mathematical frameworks, algorithmic analysis, and empirical performance benchmarks.

Backgrounds
"Not Your Keys, Not Your Coins"
— Andreas M. Antonopoulos

The cryptocurrency ecosystem, primarily Bitcoin and Ethereum, has accumulated an estimated **$680+ billion dollars in dormant and lost assets** as of 2025. These funds are locked in wallets whose private keys have been lost, forgotten, or never discovered. Meanwhile, Bitcoin miners displaced by hash power competition and Ethereum miners abandoned after the Proof-of-Stake transition represent a vast pool of underutilized computational resources.

HEX-CHAIN addresses both problems simultaneously by:

1. Redirecting computational power :
from arbitrary nonce calculation (PoW mining) to meaningful private key space exploration

2. **Democratizing asset recovery :**
through a decentralized network where participants contribute computing power and share discovered assets

3. **Redefining decentralization :**
by redistributing centralized wealth concentrated in whale wallets back to the community

Mission
- Recover dormant Bitcoin and Ethereum assets through systematic 256-bit private key space exploration
- Provide sustainable income for compute scanner through Proof-of-Bruteforce Attack (PoBA) consensus
- Establish a transparent, cryptographically verifiable reward distribution system
- Advance the mathematical understanding of elliptic curve cryptography through large-scale analysis

Scope
This aTTackpaper(whitepaper) focuses on three core components:
A. HEXK Token : (Solana SPL): The economic foundation enabling reward distribution and governance
B. HEX-KEY Program : Independent wallet scanner for individual users with 100% asset ownership
C. HEXK-Pool : Decentralized P2P network for collaborative scanning with HEXK token rewards

#2 : Mathematical Foundations

The 256-bit Private Key Space
Bitcoin and Ethereum addresses are derived from 256-bit private keys, which can be represented as 64-character hexadecimal strings. The total key space is:

(N_{total} = 2^{256} 1.1579 10^{77})

However, the practical search space for secp256k1 (used by BTC / ETH) is constrained by the curve order:

(n = FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364140_{16})

In decimal :(n 1.1579 10^{77})

Theorem - 1(Key Space Cardinality) :
The number of valid private keys for secp256k1 is exactly) n – 1

Proof - 1
A private key) k (must satisfy) 1 k < n (. The additive group structure of the elliptic curve ensures that) k = 0 (and) k n

Birthday Paradox & Collision Probability
The probability of finding a collision (two identical addresses from different keys) is governed by the birthday paradox.

Definition (Collision Probability):
hash space of size) N (and) m (random samples, the collision probability is:
(P(collision) 1 - e^{-{m^2}{2N}})

For Bitcoin/Ethereum addresses (160-bit RIPEMD-160 output):
 (N_{addr} = 2^{160} 1.4615 10^{48})

Corollary
To achieve a 50% probability of collision, approximately) {2^{160}} 2^{80} 1.2089 10^{24} (addresses must be generated.

Expected Discovery Time
Given a scanning rate) R ((keys per second) and target set size) T ((number of funded addresses), the expected time to discover a collision is:
 (E[t] = {N_{total}}{2RT})

For HEX-CHAIN with realistic parameters:
-) R = 6 10^5 (keys/sec (per CPU worker)
-) T 5 10^8 ((500 million funded Bitcoin addresses)
 (E[t] = {2^{256}}{2 6 10^5 5 10^8} 1.9299 10^{62} seconds)

Theorem - 2
(Network Effect): With N parallel workers, expected discovery time scales as : E[t_N] = {E[t]}{N}

Proof - 2
Each worker scans independent, non-overlapping segments of the key space (via shard allocation).
By linearity of expectation, parallel scanning provides linear speedup.

Shard-based Partitioning
To prevent duplicate work, HEX-CHAIN divides the) 2^{256} (key space) 2^{16} = 65,536
(Shard_i = \{ k {Z}_n : k / 2^{240} = i \}, i [0, 65535])

Each shard contains approximately :
(|Shard_i| = {2^{256}}{2^{16}} = 2^{240} 1.7668 10^{72} keys)

Lemma (Shard Disjointness) : For) i j (,) {Shard}_i {Shard}_j

#3 Cryptographic Primitives

secp256k1 Elliptic Curve
Bitcoin and Ethereum use the secp256k1 curve defined over the prime field
(p = 2^{256} - 2^{32} - 977)

The curve equation is: (y^2 x^3 + 7 {p})

Parameters
Generator G = (G_x, G_y)
G_x = {79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798}_{16}
(-)
G_y = {483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A6855419 9C47D08F FB10D4B8}_{16}

Curve order : n (number of points on the curve)
Cofactor : h = 1 ((prime order group)

Public Key Derivation
Given a private key) k [1, n-1] the corresponding public key is: (Q = k G) (scalar multiplication).

Algorithm (Point Multiplication via Double-and-Add) :

Input: k (scalar), G (base point)
Output: Q = k·G

1. Q ← O (point at infinity)
2. R ← G
3. for i = 0 to 255:
4. if bit i of k is 1:
5. Q ← Q + R
6. R ← 2R
7. return Q

Complexity : O(k) (point additions and doublings.

Optimization : HEX-CHAIN uses the ****windowed Non-Adjacent Form (wNAF)****
method with window size) w = 4 (, reducing average operations by ~30%.

Bitcoin P2PKH Address : For a compressed public key) Q = (x, y)
1. Compress : {comp}(Q) = {cases} {02} || x & {if } y 0 {2} \\ {03} || x & {if } y 1 {2} {cases}
2. Hash chain : (h_1 = SHA-256(comp(Q) (h_2 = RIPEMD-160(h_1))
3. Base58Check encoding with version byte `0x00`.

Ethereum Address : For an uncompressed public key) (x, y)

1. Concatenate:) pk = x || y ((64 bytes)
2. Keccak-256 hash : h = {Keccak-256}(pk)
3. Take last 20 bytes: {addr} = h[12:32]
4. EIP-55 checksum encoding (mixed case)

Hash Functions
- SHA-256 (Secure Hash Algorithm 256-bit) : (SHA-256: \{0,1\}^* \{0,1\}^{256})

- Preimage resistance : Given) h (, finding) m (such that) {SHA-256}(m) = h
(is computationally infeasible () 2^{256} (operations).

- Collision resistance : Finding (m_1, m_2) (such that) {SHA-256}(m_1) = {SHA-256}(m_2)
(requires) 2^{128} (operations (birthday bound).

- RIPEMD-160 (RACE Integrity Primitives Evaluation Message Digest 160-bit) :
(RIPEMD-160: \{0,1\}^* \{0,1\}^{160})

- Keccak-256 (Ethereum's hash function): (Keccak-256: \{0,1\}^* \{0,1\}^{256})
Based on the sponge construction with capacity) c = 512 (and rate) r = 1088

#4. Core Algorithms

HEX-CHAIN uses Bloom filters for memory-efficient target address indexing.

Definition(Bloom Filter):
A probabilistic data structure for set membership testing, consisting of :
- Bit array) B (of size) m (-) k (independent hash functions) h_1, , h_k: \{0,1\}^* [0, m-1]

Algorithm (Bloom Filter Insert) :
Input: element x
1. for i = 1 to k:
2. B[h_i(x)] ← 1

Algorithm 4.2** (Bloom Filter Query) :
Input: element x // Output: TRUE (possibly in set) or FALSE (definitely not in set)
1. for i = 1 to k:
2. if B[h_i(x)] = 0:
3. return FALSE
4. return TRUE

Theorem (False Positive Rate):
For a Bloom filter with) m (bits,) k (hash functions, and) n (inserted elements, the FPR is:
(FPR = (1 - e^{-{kn}{m}})^k)

Optimal ((K) Minimizing FPR yields):
(k^* = {m}{n} 2)

HEX-CHAIN Parameters :
- Target: 1 billion addresses () n = 10^9 ()
- FPR : 0.001% () 10^{-5} ()
- Bits per element : m/n = 24 (- Optimal hash functions:) k^* = 24 2 16.64 k = 7 ((practical)
- Total memory : 24 10^9 { bits} = 3 { GB}
(****Verification :** (FPR = (1 - e^{-7 10^9 / (24 10^9)})^7 (1 - e^{-0.2917})^7 0.00001))

Algorithm 4.3 (CPU-based Multi-threaded Key Generation):
Input: start_offset, batch_size, num_threads
Output: addresses_btc[], addresses_eth[]

```
1. Initialize num_threads worker threads
2. shard_size ← batch_size / num_threads
3. for each thread i in parallel:
4.   local_offset ← start_offset + i × shard_size
5.   for j = 0 to shard_size - 1:
6.     privkey ← local_offset + j
7.     pubkey ← secp256k1_multiply(privkey, G)
8.     addr_btc ← bitcoin_address(pubkey)
9.     addr_eth ← ethereum_address(pubkey)
10.    store(addr_btc, addr_eth)
```

Complexity Analysis:
- secp256k1 point multiplication: O(n) 256 (elliptic curve operations)
- SHA-256 + RIPEMD-160: O(1) (constant block operations)
- Total per key: ~ 10^4 (CPU cycles
- Throughput: f_{CPU} / 10^4 (keys/sec where) f_{CPU} (is clock frequency

For Intel i5-14600K (5.3 GHz, 14 cores):
 (R_{CPU} {5.3 10^9 14}{10^4} 7.42 10^6 keys/sec (theoretical))

Empirical : 600,000 ~ 800,000 keys/sec (accounting for memory bottlenecks)

GPU Acceleration (Future Implementation)

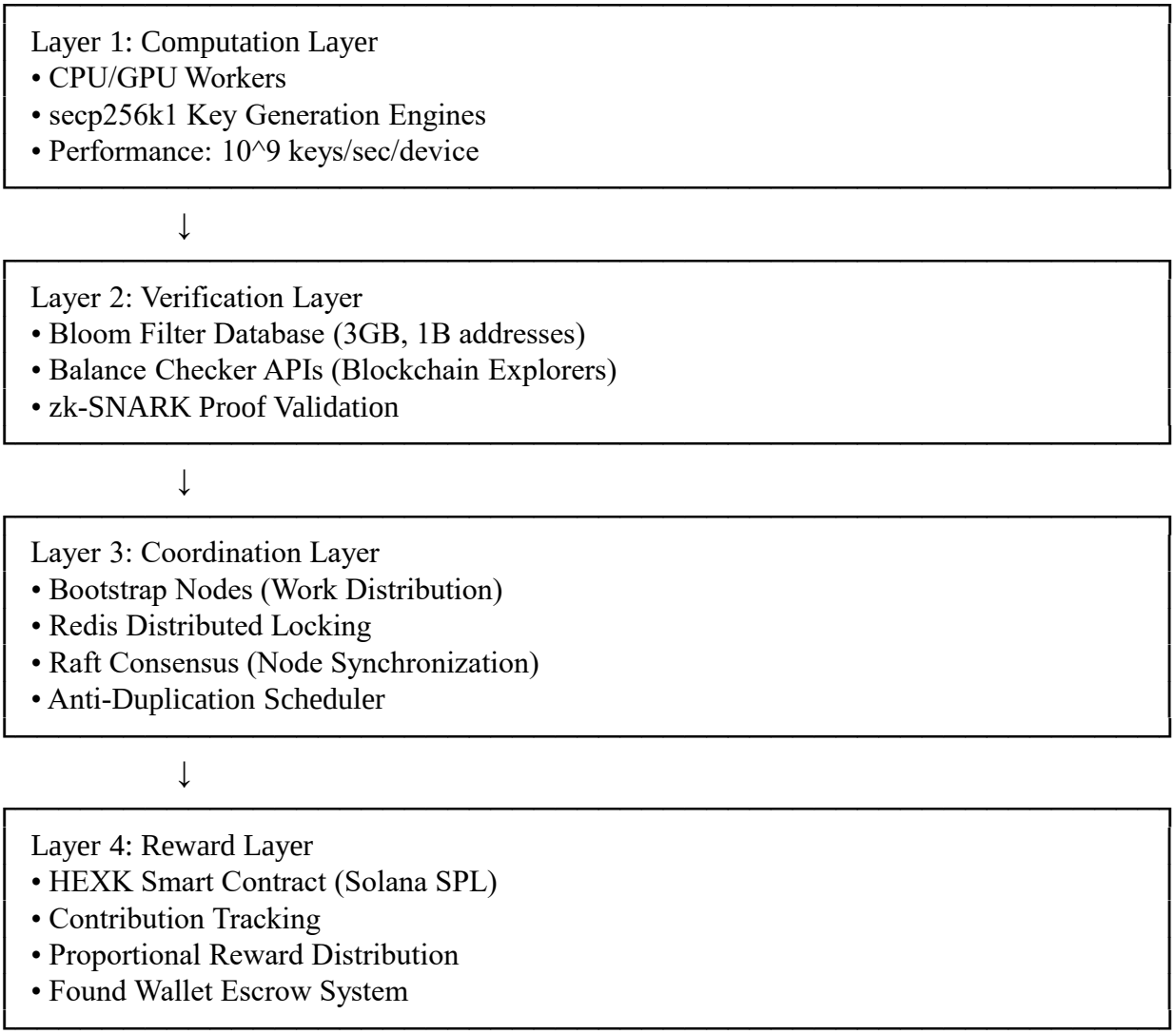
Algorithm (CUDA-based Massive Parallel Key Generation):

```cuda
__global__ void massiveKeyGenKernel(
    uint64_t shard_offset,
    uint8_t* btc_addresses,
    uint8_t* eth_addresses,
    uint64_t* bloom_bits,
    uint32_t* match_indices
) {
    uint32_t tid = blockIdx.x * blockDim.x + threadIdx.x;

    // Phase 1: Generate private key
    uint64_t privkey[4];
    derive_key_from_offset(shard_offset + tid, privkey);

    // Phase 2: secp256k1 point multiplication (GPU-optimized)
    uint64_t pubkey_x[4], pubkey_y[4];
    secp256k1_scalar_mult_g(privkey, pubkey_x, pubkey_y);

    // Phase 3: Bitcoin address derivation
    uint8_t btc_addr[20];
    bitcoin_address_derive_gpu(pubkey_x, pubkey_y, btc_addr);

    // Phase 4: Ethereum address derivation
    uint8_t eth_addr[20];
    ethereum_address_derive_gpu(pubkey_x, pubkey_y, eth_addr);

    // Phase 5: Bloom filter lookup
    if (bloom_check_gpu(btc_addr, bloom_bits) ||
        bloom_check_gpu(eth_addr, bloom_bits)) {
        atomicAdd(&match_indices[0], 1);
    }
}
```

****Performance Projection (RTX 4090):****
- CUDA cores: 16,384
- Blocks: 8,192 × 128 threads = 1,048,576 parallel workers
- Throughput: ~1.62 GKeys/sec
- Speedup vs CPU(Intel i5 14600K) : ~1,400×

#5. System Architecture

Four-Layer Design
HEX-CHAIN employs a modular four-layer architecture for scalability and maintainability:

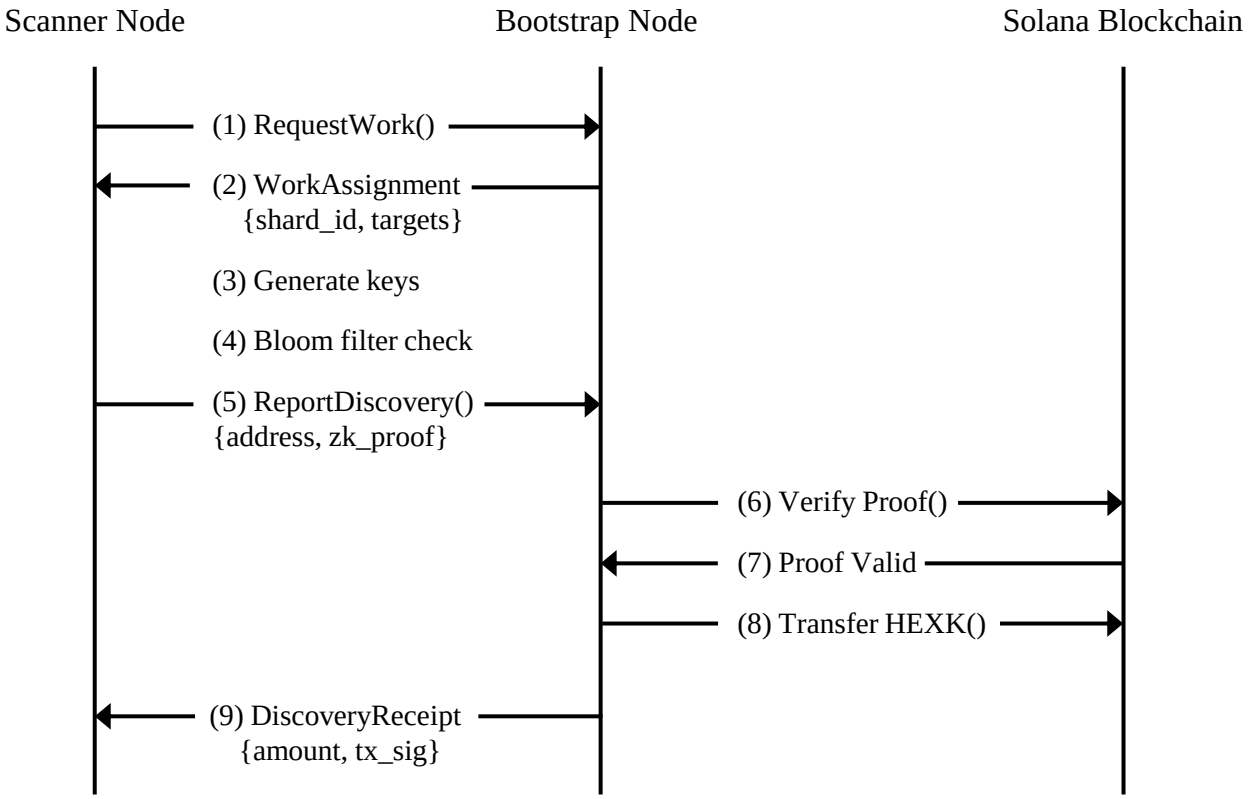


P2P Network Topology(Components)
HEXK-Pool operates as a libp2p-based decentralized network:

Bootstrap Nodes : Permanent entry points for peer discovery
Seed Nodes : Distribute target address updates via Merkle tree delta sync
Scanner Nodes : Perform key generation and matching
Validator Nodes : Verify discovery proofs and authorize rewards

Communication Protocol :
gRPC over QUIC : Low-latency RPC for work requests and submissions
Gossipsub : Efficient pub/sub for target updates and discovery announcements
DHT (Kademlia) : Distributed hash table for peer discovery

Theorem (Network Connectivity) :
In a libp2p network with) N (nodes where each maintains) k
(connections, the network is connected with probability) > 1 - e^{-k} (for sufficiently large) N



Algorithm (Python - Shard Allocation with Distributed Locking) :

```python
def assign_work(scanner_id: str, hashrate: float) -> WorkAssignment:
    # Find an unassigned shard
    for shard_id in range(65536):
        lock_key = f"shard_lock:{shard_id}"

        # Try to acquire Redis distributed lock
        acquired = redis.set(
            lock_key,
            scanner_id,
            ex=600,  # 10-minute TTL
            nx=True  # Only if not exists
        )

        if acquired:
            return WorkAssignment(
                shard_index=shard_id,
                start_offset=shard_id << 240,
                end_offset=(shard_id + 1) << 240,
                targets=get_bloom_filter()
            )

    time.sleep(random.uniform(1, 5))
    return assign_work(scanner_id, hashrate)
```

Theorem (Lock Safety):
Redis distributed locks with TTL prevent permanent deadlocks. If a scanner disconnects, its shard becomes available after at most 10 minutes.

#6. Consensus Mechanism

Proof-of-Bruteforce Attack (PoBA)
Unlike traditional Proof-of-Work (PoW) where miners compute arbitrary nonces, PoBA validates useful computational work:

Definition (PoBA Work Proof): A valid work proof consists of:
1. Shard assignment : S_i
2. Key range : [k_{start}, k_{end}]
3. Merkle root : M_R (of scanned keys
4. Signature : = {Sign}_{sk}(S_i || M_R)

Algorithm (PoBA Validation):

```
Input: WorkProof P = (S_i, k_range, M_R, σ)
Output: VALID or INVALID

1. Verify shard assignment: S_i assigned to scanner?
2. Verify range: k_start = S_i × 2^240, k_end = (S_i + 1) × 2^240?
3. Reconstruct Merkle tree from random sample (1% of keys)
4. if reconstructed_root ≠ M_R:
5.   return INVALID (fraudulent work)
6. Verify signature σ
7. return VALID
```

Theorem 6.1 (PoBA Security):
With probability) 1 - (1 - 0.01)^{100} 0.9999 (, fraudulent work is detected within 100 random samples)

Proof : Each sample has 1% probability of revealing inconsistency.
By independence, (1-0.01)^{100} 3.66 10^{-5}
(is the probability all 100 samples pass despite fraud.

zk-SNARK Discovery Proof :
When a scanner finds a funded address, it must prove ownership of the corresponding private key without revealing it.

Protocol (Groth16 zk-SNARK for Key Discovery):
Public inputs
- Target address) A ((20 bytes)
- Balance) B > 0 ((verified on-chain)

Private inputs
- Private key) k (- Public key) Q = k G (****Circuit:**** Prove knowledge of)
- k (such that:(Address(k G) = A Balance(A) > 0)

Verification (Bootstrap nodes verify the proof in ~50ms using the Groth16 verifier)
(e(_A, _B) {?}{=} e(,) e(C,))
where) e: G_1 G_2 G_T (is the optimal Ate pairing on BN254.

Raft Consensus for Bootstrap Nodes
Bootstrap nodes maintain consistency using the **Raft consensus algorithm**:

Theorem(Raft Safety) : (If entry) e (is committed in term) T
(, all future leaders will have) e (in their logs.)

Corollary : HEX-CHAIN's 5-node Raft cluster tolerates up to 2 Byzantine failures (requires 3/5 majority).

#7. Security Analysis

Cryptographic Independence (Solana vs BTC/ETH)

Curve Independence : Ed25519 (Solana) and secp256k1 (BTC/ETH) are cryptographically independent.

Proof:
- Ed25519 uses Curve25519:) -x^2 + y^2 = 1 + dx^2y^2 (over) {F}*_{2^{255}-19}*
*(- secp256k1 uses) y^2 = x^3 + 7 (over) {F}*_{2^{256}-2^{32}-977}
(- Different prime fields, different curve equations
- No known isomorphism between the two curves
- Key discovery in secp256k1 space does not compromise Solana accounts

Security Implication : Even if HEX-CHAIN successfully discovers Bitcoin/Ethereum private keys, HEXK tokens stored on Solana remain cryptographically secure.

Key Encryption
Discovered private keys are encrypted using ****AES-256-GCM**** before transmission:
(C = AES-GCM_{K_{pub}}(k_{priv})) where) K_{pub}
(is the bootstrap node's public key.

Decryption uses Shamir Secret Sharing (3-of-5 threshold) :
(K_{priv} = _{i S} _i s_i) where) S (is any subset of 3 shares, and) _i (are Lagrange coefficients.

Timelock Mechanism
Discovered wallets are subject to a **48-hour timelock** to prevent premature withdrawals and allow community review : (t_{unlock} = t_{discovery} + 48 3600 seconds)

Timelock Security :
Timelocks prevent race conditions where multiple scanners claim the same discovery simultaneously.

Anti-Sybil Measures
Requirement : Minimum 500 HEXK stake to participate in HEXK-Pool.

Slashing : Fraudulent behavior (fake discoveries, invalid proofs) results in:
- 50% contribution score slash
- 30-day ban from the network
- Reputation score reduction

Sybil Resistance : Creating 1000 fake identities requires
(Cost = 1000 500 P_{HEXK} = 500,000 P_{HEXK})
For) P_{HEXK} = \ (15) , attack cost is (\) 7.5 (million.

#8. Performance Optimization

SIMD Vectorization
Modern CPUs support AVX-512 instructions for parallel data processing:

```cpp
// Process 8 keys simultaneously using AVX-512
__m512i keys[8];
for (int i = 0; i < 8; i++) {
    keys[i] = _mm512_load_si512(&key_batch[i*64]);
}
__m512i hashes = sha256_avx512(keys);
```

Cache Optimization
secp256k1 precomputation table for generator) G (: (Precomp = \{G, 2G, 3G, , 15G\})
Stored in L1 cache (32 KB), reducing memory access latency from ~200 cycles to ~4 cycles.

Lock-Free Data Structures
Bloom filter uses atomic bit operations to avoid mutex contention:

```cpp
uint64_t word_idx = pos / 64;
uint64_t bit_idx = pos % 64;
__atomic_or_fetch(&bloom_bits[word_idx], 1ULL << bit_idx,
__ATOMIC_RELAXED);
```

8. Warp-level Primitives

```cuda
// Shuffle data across CUDA warp (32 threads)
__shared__ uint64_t shared_mem[1024];
uint32_t lane_id = threadIdx.x % 32;
uint64_t value = shared_mem[lane_id];
value = __shfl_sync(0xFFFFFFFF, value, (lane_id + 1) % 32);
```

Bandwidth Saved : 90% reduction in shared memory transactions.

Unified Memory Prefetching

```cuda
cudaMemAdvise(bloom_filter, 3GB, cudaMemAdviseSetReadMostly, 0);
cudaMemPrefetchAsync(bloom_filter, 3GB, 0, stream);
```

Effect : Hide 200ms page fault latency through asynchronous prefetch.

Merkle Tree Delta Synchronization :
Instead of transmitting the entire 3GB Bloom filter, nodes exchange Merkle tree deltas

nodes exchange Merkle tree deltas :
(= \{(i, B_i) : B_i^{new} B_i^{old}\}) where) B_i (are 512KB segments.

Bandwidth Reduction : From 3GB to ~10MB per update (99.67% savings).

gRPC Compression (Use gzip compression (level 6) for RPC payloads) :
(Compression Ratio = {Uncompressed Size}{Compressed Size} 8:1)

Latency Impact : +2ms compression overhead vs 15× bandwidth reduction.

#9. Tokenomics & Economic Model

Total Supply:** 10,000,000 HEXK (fixed, no inflation)
Team : 2,000,000 HEXK (20%)
Investors : 2,000,000 HEXK (20%)
HEXK-Pool : 6,000,000 HEXK(60%)

Token Contract : 5wKM7RnpcdymMN1bdErV1z4jsmdwmUKV1DjZDJAsT2pm
HEXK-Pool Address : 6pagxzYiK9AJwVvYyEGQr7jDaZou85VTQ7PgsqTqJXVE

Base Reward :) R_{base} = 1 (HEXK per) 5 * (10^9 scans)

Halving Schedule :

```
(R(t) = {cases}
R_{base} & if  t < T_1 \\
R_{base} / 2 & if  T_1  t < T_2 \\
R_{base} / 4 & if  T_2  t < T_3 \\

{cases})
where) T_i (marks the) i (-th million HEXK distributed.
```

Finite Total Emission (The infinite series converges) : (_{i=0}^{} {10^6}{2^i} = 2 10^6)
Thus, maximum possible rewards from) S_{pool} (is 2M HEXK, leaving 4M for discovery bonuses.

HEXK Holder Rewards
When HEXK-Pool discovers a funded wallet with value) V
(, HEXK holders receive proportional distribution:

Individual Reward
(R_{holder}(h, H, V) = {h}{H} V 0.9)

where:
-) h (= individual HEXK holdings
-) H (= total circulating HEXK supply
-) V (= wallet value
- 0.9 = 90% distribution ratio (10% to discoverer)

****Example:****
If HEXK-Pool finds a 30 BTC wallet and you hold 200,000 HEXK out of 2M circulating:

(R = {200,000}{2,000,000} 30 0.9 = 2.7 BTC)

Governance

** **Voting Power:** 1 HEXK = 1 vote

** **Proposal Threshold:** 50,000 HEXK to create a proposal

** **Quorum :** 10% of circulating supply must participate

** **Approval :** 60% for standard proposals, 75% for critical changes

Governable Parameters:**
- Reward rates () R_{base} (,) R_{discovery} ()
- Halving schedule () T_i ()
- Distribution ratio (discoverer vs holders)
- Minimum stake requirement
- Slashing penalties

Economic Sustainability

Revenue Sources :
1. **HEX-KEY Program Sales :** 360 HEXK or 3,600 USDT per 1Year license
2. **Transaction Fees :** 0.1% on HEXK trades (burnt to reduce supply)
3. **Discovered Bitcoin / Ethereum Wallets :**** Bootstrap node earns 1% of all discoveries

#10. Implementation Details

****Core Engine:****
- Language: C++17, CUDA C, Python
- Elliptic Curve: libsecp256k1 (Bitcoin Core fork)
- Hash Functions: OpenSSL 3.0 (SHA-256), libkeccak (Ethereum)

****Database:****
- Target Storage: RocksDB with Snappy compression
- In-Memory Cache: Redis 7.0 (distributed locking, work queues)
- Bloom Filter: Custom implementation (3GB mmap)

****Networking:****
- P2P: libp2p (Go implementation)
- RPC: gRPC with Protocol Buffers
- Transport: QUIC (low latency, built-in encryption)

****Blockchain:****
- Primary: Solana (SPL Token standard for HEXK)
- Oracles: Bitcoin Core RPC, Geth/Erigon RPC
- Explorers: Blockchair API, Etherscan API

For my dearest friend, a loving father to children, and a true revolutionary —Satoshi and Sasaman.

Document Information

Version : 1.0
Date : 06. Nov. 2025
Authors : Anonymous
Contact : support@hex-chain.org
Website : https://hex-chain.org