BRC-116: Proof-of-Indexing Hash-to-Mint Tokens

Richard Boase (richard@b0ase.com)

Abstract

This BRC defines a mechanism for minting BSV-21 tokens via Proof-of-Work where the hash preimage includes an **indexing work commitment** — a merkle root of verifiable overlay network activity. The token supply is locked in an sCrypt smart contract at deployment. The contract verifies the PoW solution on-chain and permanently records the work commitment, while an L2 gossip network verifies the honesty of the claimed work off-chain. This creates an economically incentivized, decentralized overlay network where nodes are rewarded for indexing BSV-21 token activity, serving content, and maintaining network connectivity.

Motivation

The Overlay Network Bootstrap Problem

BSV overlay networks (BRC-22) require nodes to index and serve specialized transaction data. Currently, overlay operation depends on centralized service providers (GorillaPool, WhatsOnChain) or volunteer node operators with no economic incentive. This creates single points of failure and limits decentralization.

The fundamental question for any overlay network is: **who runs the nodes, and why?**

The Indexer Incentive Gap

BSV-21 tokens[1] enable rich on-chain economies — content access tokens, domain tokens, creator tokens — but these economies require indexing infrastructure to function. Someone must watch the chain, track transfers, maintain state, and serve queries. Without economic incentives, this work falls to a small number of centralized operators.

The PoW20 Limitation

The existing POW-20 protocol[2] uses inscription-based minting where PoW solutions are embedded as JSON in transaction data. Validation depends entirely on off-chain indexers:

1. The BSV network does not verify that the PoW solution is valid
2. Different indexers can disagree on token state
3. A single corrupted indexer can produce an inconsistent ledger
4. No on-chain record of what work was claimed

This Proposal

BRC-116 solves these problems by:

1. **Moving PoW verification on-chain** via an sCrypt Hash-to-Mint smart contract (BSV-21)
2. **Binding mining to indexing work** by including a work commitment in the hash preimage
3. **Creating a permanent on-chain audit trail** of all claimed network activity
4. **Using L2 gossip consensus** to verify work honesty, with economic penalties for dishonesty

The result is a self-sustaining overlay network where running a node is profitable, honesty is economically rational, and no centralized indexer is required.

Specification

1. Token Standard

BRC-116 tokens are BSV-21 fungible tokens[1] deployed via the `deploy+mint` operation. The entire token supply is locked in a single sCrypt smart contract UTXO at genesis. Tokens enter circulation only when a miner successfully calls the contract's `mint` method with a valid Proof-of-Work solution.

The token is identified by its genesis outpoint (`<txid>_<vout>`), not by a ticker string. This eliminates ticker squatting and ensures globally unique identification.

2. Hash-to-Mint Contract

2.1 Contract State

The contract maintains two stateful properties that change with each mint:

Property	Type	Description
`supply`	`bigint`	Remaining unminted tokens. Decremented each mint.
`mintCount`	`bigint`	Total successful mints. Used for halving calculation.

The contract maintains four immutable properties set at deployment:

Property	Type	Description
`lim`	`bigint`	Base tokens per mint (before halving).
`difficulty`	`bigint`	PoW difficulty target. See Section 2.3.
`halvingInterval`	`bigint`	Number of mints per halving era.
`max`	`bigint`	Maximum total supply (inherited from BSV20V2).

2.2 The `mint` Method

The contract exposes a single public method:

```
mint(dest: Addr, nonce: ByteString, workCommitment: ByteString)
```

Parameters:

Parameter	Size	Description
`dest`	20 bytes	Miner's P2PKH address. Receives minted tokens.
`nonce`	Variable	Miner's solution. Iterated until hash meets difficulty.
`workCommitment`	32 bytes	SHA-256 merkle root of claimed indexing work. See Section 3.

Execution flow:

1. **Validate inputs.** Assert `len(workCommitment) == 32` and `len(nonce) > 0`.
2. **Build PoW challenge.** Concatenate:

   `challenge = prevTxId || workCommitment || dest || nonce`

   Where `prevTxId` is `this.ctx.utxo.outpoint.txid` (32 bytes) — the transaction ID of the current contract UTXO. This value changes with every mint, preventing pre-computation of solutions.

   Including `dest` prevents mempool front-running: a solution is only valid for the miner who computed it.

   Including `workCommitment` binds the PoW to specific claimed work.
3. **Verify PoW.** Compute `h = SHA256(SHA256(challenge))`. Assert that `h` meets the difficulty target (see Section 2.3).
4. **Calculate mint amount.** Compute the halving era: `era = mintCount / halvingInterval`. The mint amount is `lim / 2^era`. Assert `amount > 0`.
5. **Decrement supply.** `supply -= amount`. Assert `supply >= 0`.
6. **Increment mint count.** `mintCount += 1`.
7. **Build outputs.**
   - If `supply > 0`: Output 0 is the state continuation UTXO (contract with updated state).
   - Next output: P2PKH transfer of `amount` tokens to `dest`.
   - Final output: Change (satoshis for transaction fees).
8. **Enforce output integrity.** Assert `SHA256(SHA256(outputs)) == this.ctx.hashOutputs`. This is the standard sCrypt output enforcement that prevents any manipulation of the transaction outputs.

2.3 Difficulty Representation

OPEN QUESTION 1: Difficulty representation format.

Two approaches are viable:

Option A — Leading zero bytes. The `difficulty` property is an integer representing the number of leading zero bytes required in the hash output. Simple to implement but coarse-grained (each increment is 256x harder).

```
difficulty = 3  →  hash must start with 0x000000...
difficulty = 4  →  hash must start with 0x00000000...
```

Verification: `slice(h, 0, difficulty) == bytes(0, difficulty)`

Option B — Integer target comparison. The `difficulty` property is a 256-bit integer target. The hash (interpreted as a big-endian unsigned integer) must be less than the target. This allows fine-grained difficulty adjustment (same as Bitcoin's nBits).

```
hashInt = bigEndianToInt(h)
assert(hashInt < target)
```

Option B is recommended for production deployments. Option A is acceptable for initial testnet deployments.

We request community feedback on which representation to standardize.

2.4 Halving

The mint amount decreases by half every `halvingInterval` mints:

Era	Mints	Amount per Mint
0	1 – `halvingInterval`	`lim`
1	`halvingInterval+1` – `2*halvingInterval`	`lim / 2`
2	`2*halvingInterval+1` – `3*halvingInterval`	`lim / 4`
N	…	`lim / 2^N`

When `lim / 2^era` rounds to zero, no more tokens can be minted and the contract terminates.

Implementation note: sCrypt does not support variable bit-shift operations. The halving must be implemented as a chain of conditional divisions:

```
amount = lim
if (era >= 1) { amount = amount / 2 }
if (era >= 2) { amount = amount / 2 }
...
```

The number of conditional branches determines the maximum number of halving eras the contract supports. Eight branches covers >99.6% of supply for typical configurations.

2.5 Contract Deployment

A BRC-116 token is deployed as a single BSV-21 `deploy+mint` transaction. The contract constructor receives all immutable parameters. The `id` field is empty at deployment and auto-initialized to `<txid>_<vout>` on first mint via the BSV20V2 base class.

The deployment transaction creates one output: the contract UTXO carrying the full token supply. This is the genesis of the token.

OPEN QUESTION 2: Parallel contract UTXOs.

Since the contract is a single UTXO chain, only one mint can succeed per BSV block for a given contract instance. Under high competition, most miners' transactions will fail (double-spend of the same contract UTXO).

Possible mitigations:

- **Option A — Single UTXO.** Accept contention as a natural rate limiter. Failed miners retry on the next UTXO. Simple but limits throughput to one mint per contract UTXO step (i.e., one per block at most).
- **Option B — Parallel UTXOs at genesis.** Deploy N contract instances in the genesis transaction, each with `max/N` supply. Miners target different instances. Increases throughput N-fold but complicates supply tracking.
- **Option C — Splittable UTXOs.** Add a `split` method to the contract that divides a single UTXO into two child UTXOs with proportional supply. Allows dynamic scaling but adds contract complexity.

We request community feedback on which approach to standardize.

3. Proof of Indexing

The `workCommitment` parameter in the `mint` method is a 32-byte SHA-256 merkle root representing indexing work the miner claims to have performed. The contract records this value permanently on-chain but does not interpret it. Interpretation and verification happen at L2 (Section 4).

Enforcement Boundaries

It is important to state the precise security guarantees of this system. The title "Proof-of-Indexing" refers to a two-layer verification model, not a single on-chain truth:

- **L1 (on-chain) proves:** (a) that sufficient Proof-of-Work was performed, and (b) that a specific work commitment (merkle root) was permanently and immutably recorded at the time of mining. The contract does not and cannot verify that the committed data represents honest indexing.
- **L2 (off-chain) proves:** that the work commitment represents genuine overlay network activity, verified probabilistically via peer gossip, challenge-response protocols, and reputation scoring.

"Proof-of-Indexing" is therefore an **economic and reputational guarantee**, not a deterministic on-chain truth. The on-chain record provides an immutable audit trail; the off-chain network provides the honesty verification. This is analogous to how Bitcoin's Proof-of-Work proves energy expenditure on-chain while the economic incentive structure (block rewards, fees) makes honest mining rational off-chain.

3.1 Work Items

A work item represents a discrete unit of overlay network activity:

```
interface WorkItem {
    id: string        // SHA-256(type || data || timestamp), hex-encoded
    type: WorkType    // Category of work performed
    data: string      // JSON-encoded type-specific proof data
    timestamp: number // Unix milliseconds when work was performed
}
```

3.2 Work Types

Type	Trigger	Proof Data
`tx_indexed`	Client observes and records a BSV-21 token transfer	`{ txid, tokenId, from, to, amount }`
`content_served`	Client serves content to a peer who presented a valid ticket	`{ contentHash, requesterPeerId, bytesServed, ticketTokenId, stampSignature }`
`stamp_validated`	Client validates a ticket's stamp chain (all signatures)	`{ ticketUtxo, chainLength, isValid, tokenId }`
`market_indexed`	Client indexes a listing or sale on a BSV-21 marketplace	`{ listingTxid, tokenId, action, priceSats }`
`peer_relayed`	Client forwards a valid gossip message to peers	`{ messageHash, messageType, relayedTo }`

OPEN QUESTION 3: Work type extensibility.

Should the set of valid work types be fixed in this specification, or should it be extensible? Extensibility allows future overlay networks to define new work types, but fixed types are simpler to verify across implementations.

We request community feedback.

3.3 Work Commitment Construction

1. Collect work items from an in-memory mempool (minimum 5, maximum configurable).
2. Sort items by `id` field (lexicographic, ascending). This ensures deterministic ordering across implementations.
3. Compute a binary Merkle tree:
   - Leaf nodes: `SHA-256(workItem.id || workItem.type || workItem.data || workItem.timestamp)`
   - Internal nodes: `SHA-256(leftChild || rightChild)`
   - Odd leaf count: duplicate the last leaf.
4. The 32-byte root hash is the `workCommitment`.

OPEN QUESTION 4: Minimum work quality.

Should the specification define a minimum ratio of "substantive" work items (e.g., `tx_indexed`, `content_served`) versus "passive" items (e.g., `peer_relayed`)? Without this, a node could fill its mempool with trivial relay events and mine with minimal useful work.

Options:

- **Option A — No minimum.** Any valid work items are acceptable. The market (peer reputation) handles quality.
- **Option B — Type quotas.** At least 50% of items must be `tx_indexed` or `content_served`.
- **Option C — Weighted items.** Different types have different "weights" and the commitment must reach a minimum total weight.

We request community feedback.

4. L2 Work Verification

The on-chain contract guarantees that PoW was performed and records the work commitment. The L2 gossip network verifies that the work commitment represents honest indexing.

4.1 Gossip Topic

Work verification uses a dedicated GossipSub topic:

`$402/mining/v1`

This topic carries three message types: `MINING_ANNOUNCEMENT`, `WORK_CHALLENGE`, and `WORK_RESPONSE`.

Authentication: Gossip messages on this topic are explicitly **unauthenticated** — any node may publish and subscribe without identity verification. Implementations that wish to run the mining gossip over authenticated channels SHOULD use BRC-103[12] mutual authentication semantics over HTTP transport (BRC-104[3]). Because the default mode is unauthenticated, reputation rules (Section 4.3) MUST assume Sybil-heavy conditions: a single entity may operate multiple peer identities.

4.2 Mining Announcement

When a node successfully mints tokens, it broadcasts a `MINING_ANNOUNCEMENT`:

```
interface MiningAnnouncement {
    mint_txid: string          // On-chain HTM contract spend txid
    contract_utxo: string      // The contract UTXO that was spent
(<txid>:<vout>)
    nonce: string              // The winning nonce (hex)
    work_commitment: string    // The 32-byte merkle root (hex)
    miner_address: string      // Reward recipient (base58check)
    work_items: WorkItem[]     // The items behind the merkle root
    merkle_proof: string[]     // Intermediate hashes for merkle verification
    era: number                // Current halving era
    amount: number             // Tokens minted
    timestamp: number          // Unix ms
```

```
    signature: string          // ECDSA signature by miner's key
}
```

Nodes MUST broadcast `MINING_ANNOUNCEMENT` within 60 seconds of a successful mint. Nodes that consistently fail to announce lose reputation but are not penalized on-chain.

4.3 Peer Verification Protocol

Receiving nodes MUST perform the following verification steps:

1. **Merkle verification.** Recompute the merkle root from the provided `work_items`. If the computed root does not match `work_commitment`, the miner is flagged as **dishonest**. This check is deterministic and cannot produce false positives.

2. **On-chain verification.** Verify that `mint_txid` exists on-chain and spends the claimed `contract_utxo`. This confirms the PoW was accepted by the network.

3. **Work item plausibility.** For each work item, verify against local state:

   - `tx_indexed`: Does the claimed `txid` exist in the verifier's local index? Does the data match?
   - `content_served`: Is the `stampSignature` valid for the claimed content and requester?
   - `market_indexed`: Does the claimed listing exist?
   - `peer_relayed`: Was the claimed `messageHash` observed by the verifier?

   Compute a **plausibility score**: `verifiedItems / totalItems`. A score below 0.5 flags the miner as **suspicious**.

   Note: Plausibility verification is probabilistic. Nodes see different subsets of network activity. An honest miner should achieve ~60-90% plausibility across diverse verifiers, not necessarily 100%.

   Sybil resistance: Because the gossip layer is unauthenticated (Section 4.1), reputation scores are **local to each node**, not globally consistent. A node MUST NOT assume that its ban list or reputation scores are shared by or authoritative for other nodes. Strong penalties (reputation set to 0, ban) SHOULD only be applied after corroboration from at least **3 independent verifiers** to reduce the impact of Sybil peers submitting false challenges or false verification results. Implementations MAY additionally require identity costs (e.g., a minimum on-chain history of successful mints, or a staking deposit per BRC-116 Open Question 5) to raise the cost of Sybil identities.

4. **Reputation update.** Adjust the miner's reputation score:

   - Merkle match + plausibility >= 0.7: increase reputation
   - Merkle match + plausibility 0.5-0.7: no change
   - Merkle match + plausibility < 0.5: decrease reputation
   - Merkle mismatch: set reputation to 0, add to ban list

4.4 Work Challenge / Response

Nodes MAY issue targeted challenges to verify specific work claims after the fact:

WORK_CHALLENGE:

```
interface WorkChallenge {
    challenge_id: string         // Unique ID for this challenge
    target_mint_txid: string     // Which mint to challenge
    requested_item_ids: string[] // Specific work item IDs to prove
}
```

WORK_RESPONSE:

```
interface WorkResponse {
    challenge_id: string
    proofs: {
        item_id: string
        merkle_path: string[]   // Merkle path from item to root
        item_data: string       // Full work item data
    }[]
}
```

A node that fails to respond to a `WORK_CHALLENGE` within 30 seconds loses reputation. A node that responds with data that does not verify against the on-chain `workCommitment` is flagged as dishonest.

4.5 Economic Security Analysis

Miner Behavior	On-Chain Outcome	L2 Outcome	Net Result
Valid PoW + honest work	Tokens minted	Full peer trust	Profitable: mining income + marketplace access
Valid PoW + fake work	Tokens minted	Peers reject, banned	Unprofitable: tokens minted but cannot sell, no marketplace
Valid PoW + empty/trivial work	Tokens minted	Reduced reputation	Marginal: tokens minted but deprioritized by peers
Invalid PoW	Transaction rejected by Transaction Processors	N/A	No tokens, wasted computation

Terminology note: In BSV, Transaction Processors (TPs) validate scripts and enforce consensus rules, rejecting invalid or double-spend attempts from the mempool. Miners package TP-validated transactions into blocks via Proof-of-Work. BRC-116's L1 security depends on both: TPs enforce the contract's spending conditions (PoW validity, supply checks, output integrity), and miners provide finality by including the valid transaction in a block.

The key economic insight: **$402 tokens are only valuable if you can trade them.** Trading happens on the L2 overlay network, where your reputation determines your access to trading partners. A miner who cheats on work commitments can mint tokens but has no market for them.

OPEN QUESTION 5: Can the economic penalty for fake work be strengthened?

In the current design, a dishonest miner still receives tokens on-chain — they just can't sell them easily on the overlay. If $402 has exchange listings outside the overlay (e.g., on a centralized exchange), the penalty is weaker. Possible mitigations:

- **Option A — Accept the current model.** Overlay reputation is sufficient for early-stage networks.
- **Option B — Staking requirement.** Require miners to lock BSV satoshis in a separate contract before mining. Dishonest miners lose their stake via a fraud proof.
- **Option C — Coinbase maturity.** Minted tokens are unspendable for N blocks (analogous to Bitcoin's 100-block coinbase maturity rule). During the maturity window, L2 peers verify the work commitment. Dishonest miners receive tokens but suffer reputation damage before tokens become liquid. No burning or fraud proofs required — the delay provides a verification window while the penalty remains economic (market exclusion).

We request community feedback on whether strengthened penalties are necessary.

5. Difficulty Adjustment

5.1 Target Rate

The recommended target is **1 successful mint per minute** (1,440 per day). This balances token distribution speed with network activity.

5.2 Adjustment Period

Difficulty is evaluated every **144 successful mints** (~2.4 hours at target rate).

5.3 Adjustment Algorithm

```
Let elapsed = time between mint (mintCount - 144) and mint (mintCount)
Let target  = 144 * 60 seconds = 8,640 seconds
Let ratio   = elapsed / target

If ratio < 0.5:  difficulty increases (solutions coming too fast)
If ratio > 2.0:  difficulty decreases (solutions coming too slow)
Otherwise:        difficulty unchanged
```

Time source: If on-chain difficulty adjustment is used (Option B below), the contract requires a reliable measure of elapsed time. Viable approaches include: (a) using `nLockTime` constraints to embed block height or timestamp in the spending transaction, allowing the contract to compute elapsed time between adjustment epochs; (b) reading block header timestamps via sCrypt's transaction context if available; or (c) tracking only `mintCount` (already in state) and relying on the off-chain network to supply wall-clock time as a signed oracle input. Approach (a) is recommended as it requires no external oracle and is enforceable by Transaction Processors.

OPEN QUESTION 6: On-chain vs off-chain difficulty adjustment.

- **Option A — Fixed on-chain difficulty.** The contract's difficulty never changes. Simple and trustless but cannot adapt to changing hashrate.
- **Option B — Stateful on-chain adjustment.** The contract tracks `mintCount` and block timestamps, adjusting `difficulty` (a `@prop(true)`) according to the algorithm above. More trustless but adds significant contract complexity.
- **Option C — Off-chain adjustment with contract migration.** The overlay network reaches gossip consensus on new difficulty. A new contract is deployed with the adjusted difficulty, and remaining supply is migrated via a `migrate` method on the old contract. Flexible but requires a migration protocol.

We request community feedback on which approach to standardize.

6. Content Token Integration

BRC-116 is designed to incentivize overlay networks that index BSV-21 content tokens. Content tokens are separate BSV-21 tokens deployed by content issuers and are NOT part of this specification. However, this section describes the intended interaction pattern.

6.1 Token Separation

Concern	Token	Standard	Deployed By	Regulation
Overlay incentive	$402 (BRC-116)	BSV-21 HTM	Smart contract (no issuer)	Commodity-like (pure PoW)
Content access	$EXAMPLE	BSV-21 (standard)	Content issuer	Issuer's responsibility

BRC-116 tokens are not required to access content. Users purchase content tokens directly with BSV. BRC-116 tokens incentivize the overlay infrastructure that makes content token economies possible.

6.2 Ticket Stamp Chains

Content tokens function as tickets with cryptographic provenance (see BRC-104[3] for the authentication substrate). When a node serves content in exchange for a ticket, it creates a stamp — a signed record of the serve event. These stamps form a chain of provenance that is both a quality signal and a source of `content_served` work items for BRC-116 mining.

Where HTTP endpoints are monetized (e.g., paid content serving that generates `content_served` work items), implementations SHOULD use BRC-105[4] to define the payment challenge/response flow. This ensures interoperability with other BSV services that implement the HTTP 402 monetization pattern.

6.3 Relationship to BRC-24 Lookup Services

"Proof-of-Indexing" fundamentally incentivizes the provision of lookup capability as described in BRC-24[13]. The `tx_indexed` and `market_indexed` work types correspond directly to BRC-24 query handling — a node that indexes token transfers and marketplace activity is providing the

data layer that BRC-24 lookup services query. While BRC-116 overlay nodes are not required to expose a BRC-24 interface, the work they perform (indexing, serving, validating) is the same work that underpins BRC-24 lookup availability.

7. Reference Parameters

The following parameters are recommended for the initial deployment of a BRC-116 token:

Parameter	Recommended Value	Rationale
`max`	21,000,000	Familiar supply cap, sufficient for large networks
`lim`	1,000	Balanced: not so large that early miners dominate, not so small that mining is unrewarding
`difficulty`	5 (leading zero bytes)	~1 in 1,048,576 per hash. CPU-mineable on commodity hardware.
`halvingInterval`	10,500	With 1 mint/minute target: first halving after ~7.3 days. Fast enough to create scarcity.
`decimals`	0	Whole tokens. Simplifies contract logic.

These values are recommendations. Deployers MAY choose different parameters.

8. BRC-100 Wallet Separation

A compliant BRC-116 mining implementation MUST allow the miner to use any BRC-100[11] compliant wallet for key management, signing, encryption, and transaction submission. The mining client is an **application** that requests operations from a wallet, not a wallet itself.

Specifically, a conforming mining client MUST be able to:

1. **Request a destination address** (`dest`) from the wallet for receiving minted tokens.
2. **Construct the contract-spend transaction template** (the unsigned mint transaction with PoW solution, work commitment, and outputs).
3. **Request the wallet to sign** any required inputs (funding inputs for transaction fees, change outputs).
4. **Submit the signed transaction** for broadcast via the wallet's broadcast interface, or via any standard broadcast endpoint.

A reference implementation MAY include an integrated wallet for testing and development purposes, but a production deployment MUST NOT require any specific wallet implementation. This ensures that BRC-116 mining remains vendor-neutral and interoperable with the broader BSV wallet ecosystem.

9. sCrypt Implementation Notes

9.1 Dependencies

- `scrypt-ts` >= 1.0.0 — sCrypt TypeScript SDK
- `scrypt-ord` >= 1.0.0 — BSV-21/Ordinals contract base classes

9.2 Base Class

BRC-116 contracts MUST extend `BSV20V2` from `scrypt-ord`. This base class provides:

- `buildStateOutputFT(amt)` — Builds the state continuation output with updated token amount
- `BSV20V2.buildTransferOutput(addr, id, amt)` — Builds a P2PKH transfer output with BSV-20 inscription
- `buildChangeOutput()` — Builds the change output for transaction fees
- `isGenesis()` / `initId()` — Genesis detection and token ID initialization

9.3 Hash Functions

- `hash256(x)` — Double SHA-256: `SHA256(SHA256(x))`. Used for PoW verification.
- `sha256(x)` — Single SHA-256. Used for individual work item hashing.

Both are available as Bitcoin Script opcodes and execute natively in the BSV VM.

9.4 Output Enforcement

Every `mint` call MUST include:

```
assert(hash256(outputs) == this.ctx.hashOutputs)
```

This ensures the transaction outputs exactly match what the contract specifies, preventing any output manipulation.

9.5 Script Size Considerations

RESOLVED — Contract script size.

The reference contract has been compiled using sCrypt compiler v1.20.0. Results:

Component	Script Size
BSV20V2 base class (Ordinal + Shift10 libraries)	22.81 KB
BRC-116 additions (PoW, halving, work commitment)	1.10 KB
Total compiled script	**23.91 KB**

For comparison, a minimal `anyonecanmint` BSV20V2 contract (no PoW, no custom logic) compiles to 22.81 KB. The 23 KB base cost is inherent to the BSV-21 Ordinal inscription library and is shared by all deployed BSV-21 HTM tokens on mainnet.

At BSV's standard fee rate of 0.5 sat/byte, each mint transaction costs approximately 15,000 sats (~$0.01 USD) including inputs, outputs, and fees. This is economically viable for CPU mining at any non-zero $402 token price.

The 24 KB script is well within BSV's miner relay limits (~10 MB typical) and poses no propagation or validation concerns.

Open Questions Summary

This specification contains six open questions requiring community input before finalization, plus one resolved item:

#	Question	Section	Status	Options
1	Difficulty representation format	2.3	**Open**	A: Leading zero bytes, B: Integer target comparison
2	Parallel contract UTXOs for throughput	2.5	**Open**	A: Single UTXO, B: Parallel at genesis, C: Splittable
3	Work type extensibility	3.2	**Open**	Fixed set vs extensible registry
4	Minimum work quality requirements	3.3	**Open**	A: No minimum, B: Type quotas, C: Weighted items
5	Strengthened penalties for fake work	4.5	**Open**	A: Reputation only, B: Staking, C: Coinbase maturity
6	Difficulty adjustment mechanism	5.3	**Open**	A: Fixed, B: Stateful on-chain, C: Off-chain + migration
7	Contract script size feasibility	9.5	**Resolved**	23.91 KB compiled — within BSV limits, ~$0.01/mint

Feedback may be submitted as issues on the BRC repository or discussed in the BSV developer community channels.

Implementations

1. **path402 client** — Reference implementation (in development). Repository: github.com/b0ase/path402. Includes mining engine, gossip network, content store, and web GUI.
2. **$402 HTM contract** — sCrypt smart contract (pending deployment). Will be deployed to BSV testnet for community testing before mainnet launch.

References

1. BSV-21 Token Standard. docs.1satordinals.com/fungible-tokens/bsv-21
2. POW-20 Protocol. protocol.pow20.io
3. BRC-104: HTTP Transport for Mutual Authentication. github.com/bitcoin-sv/BRCs
4. BRC-105: HTTP Service Monetization Framework. github.com/bitcoin-sv/BRCs
5. BRC-22: Overlay Network Topics. github.com/bitcoin-sv/BRCs
6. sCrypt BSV20V2 Base Class. github.com/sCrypt-Inc/scrypt-ord
7. Lock-to-Mint Pattern (msinkec). gist.github.com/msinkec/6389a7943ed054fa5c74ba8f79bf730e
8. POW20 Miner (Rust). github.com/yours-org/pow20-miner
9. BSV-21 Overlay. github.com/b-open-io/bsv21-overlay
10. BRC-62: Background Evaluation Extended Format (BEEF). github.com/bitcoin-sv/BRCs
11. BRC-100: Wallet-to-Application Interaction Substrate. github.com/bitcoin-sv/BRCs
12. BRC-103: Peer-to-Peer Mutual Authentication. github.com/bitcoin-sv/BRCs
13. BRC-24: Overlay Lookup Services. github.com/bitcoin-sv/BRCs