BitFS: A Peer-to-Peer Encrypted File System on Blockchain

v0.0.1

Alex Tong
alex@bitfs.org

Abstract

BitFS is a decentralized file system that maps Unix filesystem semantics onto a blockchain-based directed acyclic graph (the Metanet protocol on BSV). Every file is encrypted by default using elliptic-curve Diffie–Hellman key derivation (Method 42), and the distinction between private, free, and paid data reduces to a single question: who can derive the decryption key. A hierarchical deterministic wallet mirrors the filesystem tree, providing stable node identities and deterministic per-file encryption keys. The content hash key_hash = SHA256(SHA256(plaintext)) serves simultaneously as the content-addressed storage key and the key-derivation salt; metadata is committed to Metanet transactions on-chain. Trustless data commerce is achieved through hash time-locked contracts (HTLC), in which the seller reveals a cryptographic key capsule as the hash preimage to claim payment. The system operates entirely in SPV mode without querying the blockchain. An agent-first HTTP interface combined with the HTTP 402 payment protocol enables AI agents to discover, navigate, and purchase content autonomously.

Contents

1 Introduction

Existing decentralized storage systems face an unresolved tension between data privacy and data commerce.

Problem statement. The fundamental contradictions of current approaches are well-documented. IPFS provides content-addressed storage but offers no native encryption or payment mechanism. Filecoin adds economic incentives but requires computationally expensive zero-knowledge proofs for storage verification. Centralized cloud storage offers convenience but demands trust in a single custodian.

Three key observations motivate the design of BitFS:

1. *Metadata and content require different guarantees.* Metadata benefits from blockchain immutability and global ordering; file content requires efficient storage and retrieval. The two should be separated—metadata on-chain, content off-chain.
2. *Encryption should be universal, not optional.* When all data is encrypted by default, the distinction between public and private reduces to key distribution. Public data simply uses a trivially derivable key.
3. *AI agents are first-class citizens.* A file system designed for the coming decade must expose machine-readable interfaces that enable agents to discover content, negotiate payment, and transact without human intermediation.

BitFS in one sentence: a Unix filesystem with blockchain public-key inodes and deterministic ECDH encryption, where a single cryptographic framework (Method 42 [1]) simultaneously provides encryption, access control, authentication, commerce, and storage proofs.

2 Metanet as Filesystem

The Metanet protocol [2] defines a directed acyclic graph on the BSV blockchain. Each node is a transaction containing an `OP_RETURN` output with a node public key P_{node} and a reference to the parent transaction. Edges are cryptographically authenticated: a child transaction must include an input signed by the parent's private key D_{parent}, which the network verifies as part of standard transaction validation. No additional trust assumptions are needed.

BitFS maps Unix filesystem concepts onto this structure:

A filename exists only in the child entry list of the parent directory, not in the node itself—exactly as in Unix, where an inode contains no name.

Node types. Three types are defined:

- FILE: holds `key_hash`, MIME type, and file size.

Unix concept	BitFS equivalent
inode	P_{node} (compressed public key, 33 bytes)
directory entry	`ChildEntry(index, name, type, pubkey)`
inode number	index (monotonic auto-increment per directory)
parent directory (..)	parent field in payload

- DIR: holds a children list and `next_child_index`.
- LINK: holds a target reference. Two subtypes exist: SOFT points to a P_{node} and always resolves to the latest version; SOFT_REMOTE points to a `domain/path` for cross-trust-boundary references resolved via DNS.

Hard links are implemented implicitly: multiple directory entries (possibly in different directories) reference the same P_{node}. This is not a LINK node type but a property of directory entries—two entries sharing a public key share the underlying file data.

Versioning. Version control is intrinsic. The same P_{node} may appear in multiple transactions, each representing a version. The transaction with the greatest block height (or latest topological order within a block) is the current version. All previous versions remain immutable on-chain, providing a complete audit trail without any additional versioning mechanism.

3 HD Key Derivation

The identity key P_{node} of each node is derived from a BIP32 [3] hierarchical deterministic wallet, structured to mirror the filesystem hierarchy:

```
m/44'/236'/0'           Fee keychain (shared across all vaults)
m/44'/236'/1'/0/0       Vault 0 root directory
m/44'/236'/1'/0/0/1     First child of root
m/44'/236'/1'/0/0/1/3   Third child of first child
m/44'/236'/2'/0/0       Vault 1 root directory (independent tree)
```

The BIP32 derivation path mirrors the filesystem path. When a new file is created in a directory, it receives the next available child index, which becomes both its directory entry number and the final segment of its BIP32 derivation path.

Vault concept. A *Vault* is an independent Metanet tree rooted at a BIP32 account. Multiple vaults share a single seed but maintain entirely separate directory hierarchies. Each vault can bind to its own domain name. A single user can thus maintain personal files, a company website, and a public dataset as isolated trees, all recoverable from one BIP39 mnemonic.

Two core properties:

1. *Stable node identity.* The P_{node} of a directory does not change when its contents are updated, enabling persistent references and reliable soft links.
2. *Deterministic recovery.* The entire key hierarchy can be regenerated from the mnemonic seed. (Transaction data must be restored from backup.)

4 Universal Encryption

All files are encrypted before storage. Rather than treating encryption as optional, BitFS encrypts everything and uses key derivation rules to determine who can decrypt.

4.1 Method 42 ECDH Key Derivation

The encryption key for each file is derived in four steps:

1. key_hash = SHA256(SHA256(plaintext)) (double hash: key derivation + content commitment)
2. point = ECDH(D_{node}, P_{node}) = $D_{\text{node}} \times P_{\text{node}}$ (elliptic-curve Diffie–Hellman)
3. aes_key = HKDF-SHA256(ikm = point.x, salt = key_hash, info = `"bitfs-file-encryption"`)
4. ciphertext = AES-256-GCM(plaintext, aes_key)

Here D_{node} is the BIP32-derived private key of the node (the Metanet inode), $P_{\text{node}} = D_{\text{node}} \times G$ is the corresponding public key, and key_hash serves dual duty as the key-derivation salt and the content-addressed storage identifier. Using the double hash SHA256(SHA256(·)) prevents exposing the raw content hash on-chain while still allowing post-decryption verification.

A key design choice is that D_{node} directly uses the BIP32-derived key, preserving the algebraic relationship needed for directory-tree-level capsule derivation.

4.2 Three Access Levels

Three access levels emerge from this single mechanism without any changes to the encryption scheme:

Access type	Key derivation	Who can decrypt
Private	aes_key = HKDF(ECDH($D_{\text{node}}, P_{\text{node}}$).$x$, key_hash, info)	Owner only (self-encryption)
Free	aes_key = HKDF(P_{node}.x, key_hash, info) (trivial key)	Anyone who knows P_{node}
Paid	Standard Method 42 ECDH capsule exchange	Buyer (after HTLC swap)

Trivial key trick. For free data, the KDF input is $P_{\text{node}}.x$ rather than an ECDH shared secret. Since P_{node} is published via DNS, anyone can derive the decryption key. The data remains encrypted on disk, maintaining a uniform storage model, but the key is publicly derivable.

Private data. The entire Metanet payload is encrypted with the owner's symmetric key. Filenames, sizes, timestamps, and directory structure are invisible on-chain.

4.3 Permission Decision Chain (v0.0.1)

Read operations follow a short-circuit evaluation chain with strict priority:

1. **acl_ref**: if an ACL reference is present, resolve it and use the result. If resolution fails, *fail closed* (deny access).
2. **access field**: if the node's metadata contains an explicit access field, apply it.
3. **owner fallback**: if neither of the above applies, fall back to owner-only access.

Write operations are restricted to the owner exclusively, enforced at the application layer. This priority chain ensures that the system defaults to the most restrictive policy when ambiguity exists.

5 Content-Addressed Storage

BitFS separates metadata from content. Metanet transactions on the blockchain store only metadata (directory structure, content hashes, access policies, pricing). Actual encrypted file content is stored independently in a flat key-value store:

```
~/.bitfs/data/{key_hash}  ->  ciphertext bytes
```

The daemon serves content over standard HTTP via `GET /data/{hash}`. Since all content is encrypted, the storage layer requires no access control of its own. Deduplication occurs naturally: identical ciphertext shares a single `key_hash` and a single stored copy. No external dependency such as IPFS is required.

6 Transaction Structure

Every filesystem operation produces a Metanet transaction with a self-sustaining UTXO structure:

```
Inputs:
  [0] Spend UTXO locked to P_parent -> Sig(D_parent) creates Metanet edge
  [1] Spend fee keychain UTXO -> pays miner fee

Outputs:
  [0] OP_RETURN: <MetaFlag> <P_node> <TxID_parent> <TLV payload>
  [1] P2PKH -> P_node     (1 sat, spendable node output)
  [2] P2PKH -> P_parent   (1 sat, refreshes parent UTXO)
  [3] P2PKH -> change address
```

Self-sustaining UTXOs. Output [2] refreshes the parent node's spendable UTXO, enabling the next operation without pre-funding. Initial funding comes solely from the shared fee keychain. The filesystem can grow indefinitely without per-node funding setup.

TLV payload. The `OP_RETURN` payload uses a Type-Length-Value encoding covering: node type, operation (CREATE/UPDATE/DELETE), content metadata, access control fields, directory children entries, and optional fields such as keywords, description, and domain binding.

Pricing. A `price_per_kb` field (satoshis/KB) supports directory inheritance: a file without an explicit price inherits from its nearest ancestor that defines one.

6.1 Paymail Binding

A vault can be associated with a Paymail address (`user@domain`) through the `paymail bind` command. This enables email-style addressing for BitFS resources and supports `.well-known/bsvalias` discovery. The binding is recorded in the Metanet transaction payload, establishing a verifiable link between the on-chain Metanet tree and the Paymail identity. Paymail operates purely at the discovery layer; raw public keys are the only identifiers embedded in blockchain transactions.

7 Trustless Data Commerce

Paid content is purchased through a hash time-locked contract (HTLC) atomic swap. The protocol requires no trusted intermediary, no buyer database, and no session state on the seller's side. The seller is *completely stateless.*

7.1 HTLC Atomic Swap Flow

The purchase proceeds in nine steps:

1. Buyer connects to the seller's daemon.
2. Method 42 ECDH handshake for mutual authentication.
3. Buyer requests the target file's `capsule_hash`.
4. Seller computes the capsule (ECDH shared secret between D_{node} of owner and P_{buyer}) and its hash.
5. Seller returns `capsule_hash` + payment address.
6. Buyer creates an HTLC transaction (SHA256 preimage verification + timeout refund).
7. Buyer broadcasts the HTLC to the blockchain.
8. Seller reveals the capsule (preimage) to claim payment.
9. Buyer reads the capsule from the chain, derives the decryption key, and decrypts the file.

Atomicity guarantee: the seller receives payment if and only if the buyer receives the key capsule. No trusted intermediary is needed.

7.2 HTLC Script Design

The HTLC uses a 106-byte pure Bitcoin Script with no sCrypt dependency:

```
<invoiceId> DROP
IF
  SHA256 <capsuleHash> EQUALVERIFY
  DUP HASH160 <sellerPkh> EQUALVERIFY CHECKSIG
ELSE
  DUP HASH160 <buyerPkh> EQUALVERIFY CHECKSIG
ENDIF
```

Field offsets within the 106-byte script:

Field	Byte offset
`invoiceId`	1
`capsuleHash`	21
`sellerPkh`	57
`buyerPkh`	83

Claim path. The seller provides `<sig> <pubkey> <fileTxID||capsule> TRUE`. The 64-byte preimage consists of the file transaction ID concatenated with the capsule, and its SHA256 hash must equal the committed `capsuleHash`.

Refund path. The buyer provides `<sig> <pubkey> FALSE`. Timeout is enforced via `nLockTime` on the refund transaction rather than `OP_CLTV`. This avoids compatibility issues: on post-Genesis BSV, opcode `0xb1` is treated as `OP_NOP2` and would be rejected by the `DISCOURAGE_UPGRADABLE_NOPS` policy.

Cross-language consistency. The Go and TypeScript implementations produce byte-identical 106-byte scripts, ensuring interoperability across platforms.

7.3 Method 42 Handshake

Before the HTLC flow, buyer and seller perform a Method 42 handshake:

1. Exchange P_{buyer}, P_{seller}, nonces, and timestamps.
2. Both parties compute: $\text{session\_key} = \text{SHA256}\big(\text{ECDH}(D_{\text{self}}, P_{\text{other}}).x \parallel \text{nonce}_{\text{buyer}} \parallel \text{nonce}_{\text{seller}}\big)$
3. HMAC verification. The seller's P_{seller} must match the P_{node} published via DNS, preventing man-in-the-middle attacks.

8 SPV Operation

BitFS operates entirely in Simplified Payment Verification mode [4]. The owner stores all self-created transactions and their Merkle proofs locally. Visitors obtain metadata from the owner's daemon and never query the blockchain directly.

Local storage structure:

```
~/.bitfs/
  wallet.enc            Argon2id-encrypted HD seed
  txstore/{txid}.tx     Complete transaction data
  txstore/{txid}.proof  Merkle proofs
  headers/              Block header chain
  cache/                Visitor metadata + decrypted content + capsule cache
```

A visitor verifies received metadata through a complete SPV verification chain: confirm transaction well-formedness, verify the Merkle proof against the block header, check that the block header belongs to the longest chain, and after decryption verify that SHA256(SHA256(plaintext)) matches the key_hash committed in the Metanet payload.

Third-party indexing services serve only as a degraded fallback when the owner's daemon is unreachable. Seed recovery restores the HD key hierarchy; transaction data must be restored from backup, as the system deliberately never scans the blockchain.

9 DNS Resolution and Publishing

BitFS uses two DNS record types to bind a Metanet tree to a domain:

- `_bitfs.example.com TXT "bitfs=02a1b2c3..."` — P_{node} identity.
- `_bitfs._tcp.example.com SRV 10 60 443 cdn1.example.com` — service endpoint.

Bidirectional verification. The DNS TXT record points to P_{node}, and the Metanet payload's domain field records the bound domain. Both must agree.

SRV load balancing. Priority and weight fields enable CDN-like distribution across multiple endpoints.

URI resolution flow:

```
bitfs://example.com/docs/readme.txt
  -> DNS TXT -> P_node
  -> DNS SRV -> endpoint list
  -> Connect to optimal endpoint, Method 42 handshake
  -> Request metadata (SPV) -> verify domain binding
  -> Traverse directory tree -> return content
```

Direct public-key addressing (`bitfs://<pubkey>/path`) bypasses DNS entirely for cases where the node identity is already known.

10 Agent-First Interface

BitFS treats AI agents as first-class users. The daemon implements HTTP content negotiation to serve both humans and agents from the same endpoints:

Accept header	Response
`text/html`	HTML + WebMCP tool declarations (planned)
`text/markdown`	CLI usage guide
`application/json`	Structured metadata

HTTP 402 for paid content. The 402 response includes payment metadata headers: `X-Price`, `X-Price-Per-KB`, `X-File-Size`, `X-Invoice-Id`, and `X-Expiry`. A human sees a paywall. A browser-based agent discovers WebMCP tools and pays autonomously. A CLI agent receives a command template (`bget --buy bitfs://...`) and executes it.

Core insight: an agent with a wallet can transparently complete micropayments. The HTTP 402 status code, traditionally a dead end for automated clients, becomes a programmable payment API.

11 Dual-Mode CLI

BitFS provides two classes of command-line tools following the Unix philosophy:

Read-only tools (b*): stateless, no wallet required, serving the visitor role.

- `bls` / `bcat` / `bget` / `bstat` / `btree` — analogous to `ls` / `cat` / `wget` / `stat` / `tree`.

Read-write commands (bitfs): require wallet and private keys, serving the owner role.

- `bitfs put/mkdir/rm/mv/cp/link` — filesystem operations.
- `bitfs sell/encrypt/decrypt` — commerce and access control.
- `bitfs vault/wallet/publish/daemon` — infrastructure management.
- `bitfs shell` — FTP-style REPL for interactive navigation.

All tools support `--json` for structured output and `--offline` for operation with locally cached metadata.

12 Incentive Layer

12.1 Method 42 Storage Proofs

The same ECDH construction that provides file encryption also enables efficient storage proofs. When content is distributed to Metanet Nodes, each copy is re-encrypted with a provider-specific ECDH key, making every replica cryptographically unique. Verification reduces to a simple Merkle challenge-response, replacing the computationally expensive zero-knowledge proofs used by systems such as Filecoin. The cost drops from hours of GPU computation to millisecond-scale ECDH and AES operations.

12.2 Metanet Chain

The Metanet network is a BSV-homomorphic sidechain with its own proof-of-work consensus and token economy (21 million fixed supply with halving schedule). OP_RETURN Merkle roots are periodically anchored to the BSV main chain. The Metanet chain provides the economic foundation for incentivizing Metanet Nodes to serve and store content. Its design is detailed in the Metanet whitepaper [5].

13 Conclusion

BitFS demonstrates that a Unix-style file system can be built on blockchain primitives without sacrificing the properties that make filesystems useful: hierarchical organization, stable references, version history, and familiar command-line semantics.

The design centers on a single cryptographic construction—ECDH key derivation on secp256k1—which simultaneously provides:

- File encryption (AES-256-GCM with HKDF-derived keys).
- Access control tiering (private, free, paid) via key distribution.
- Mutual authentication (Method 42 handshake).
- Trustless atomic commerce (HTLC with capsule as preimage).
- Storage proofs (provider-specific re-encryption + Merkle challenges).

By encrypting all data by default, BitFS eliminates the configuration surface area where privacy failures typically occur. By operating in SPV mode, it avoids dependence on blockchain indexing infrastructure. By designing for AI agents as first-class users, it transforms the HTTP 402 paywall from a barrier into a programmable payment interface.

The separation of BitFS (the filesystem protocol) from the Metanet network (the distribution layer) allows each to evolve independently while maintaining a clean interface: BitFS produces encrypted, content-addressed data with on-chain metadata; the Metanet network distributes that data with economic incentives aligned toward retrieval performance rather than storage capacity. The future belongs to autonomous agents transacting with decentralized services at machine speed; BitFS is designed for that future.

References

[1] C. S. Wright, "An Immutable File and Data Store," nChain, 2019. Method 42: Deterministic key derivation for per-file encryption using ECDSA/secp256k1.

[2] nChain, “The Metanet Technical Summary v1.0,” 2020. Blockchain-based directed acyclic graph for organizing data relationships.

[3] P. Wuille, “BIP32: Hierarchical Deterministic Wallets,” Bitcoin Improvement Proposals, 2012.

[4] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” 2008. Section 8: Simplified Payment Verification.

[5] GB2608179A, “Multi-level Blockchain,” UK Intellectual Property Office, 2021. Embedding secondary data chains on a core blockchain.