Stampcoin

A single-file, self-validating blockchain stamped on Bitcoin

Version 1.0 · June 2026

Abstract

Stampcoin is a complete, account-based blockchain implemented as a single self-contained HTML file with no external assets, small enough to be stamped directly onto Bitcoin. It uses the browser’s built-in Web Crypto API for real ECDSA (P-256) key pairs, digital signatures, and SHA-256 proof-of-work—no third-party libraries are loaded. The system implements signed transactions, a replay-derived ledger, native-coin transfers, user-issued tokens, an automated market maker with withdrawable liquidity, non-fungible assets with on-chain generative art, and a multi-node network that gossips blocks and resolves forks by longest-valid-chain reorganisation. Protocol rules are enforced at four independent layers—structural block verification, state-transition validation at block acceptance, mempool admission, and ledger derivation—making invalid blocks unable to be stored or to become canonical. This paper documents the architecture, the consensus and validation model, the embedded economic simulation, an explicit account of which components are real versus simulated, the deployment model as a Bitcoin Stamp, and the test results that back the design.

Property	Value
Native coin	STC (integer units), block reward 50 STC, fixed fee 1 STC
Cryptography	ECDSA P-256 signatures, SHA-256 hashing — via Web Crypto API
Ledger model	Account-based; state derived by replaying the chain
Consensus	Proof-of-work; longest valid chain (simulated multi-node)
Artifact	One HTML file, no external assets, ~61.8 KB minified
Verification	Engine + UI + stress + adversarial + precision suites, all passing on the minified artifact

1. Introduction and Motivation

Bitcoin Stamps store arbitrary data inside transaction outputs, where it becomes part of the UTXO set and is therefore effectively unprunable—once stamped, the data persists for as long as Bitcoin does. This durability makes Stamps an attractive substrate for self-contained programs: a single HTML file, with all of its logic, styling, and assets inlined, can be stamped and then run by anyone who retrieves it. The constraint is severe but clarifying: there can be no external scripts, stylesheets, fonts, or images, and the whole artifact must fit within a strict byte budget.

Stampcoin asks how faithfully a blockchain can be reproduced inside that envelope. The goal was not a toy with mocked cryptography, but a system whose security-relevant mechanics—key generation, signing, hashing, proof-of-work, ledger derivation, and consensus rules—are genuinely implemented, while being honest about the one property a sandboxed web page cannot provide: a live, decentralised peer-to-peer network. The result is a single-player sandbox that models the full mechanics of a chain and an on-chain economy, suitable as an educational artifact and a demonstration of what fits in a Stamp.

1.1 Design goals and constraints

- **Single file, zero external assets.** No CDNs, libraries, web fonts, or images; everything is inline so the stamped bytes are complete.
- **Real cryptography.** Use the platform’s audited primitives rather than reimplementing or faking them.
- **Verifiable rules.** The ledger must be reconstructable and checkable by replaying the chain, with rules enforced by the engine rather than the interface.
- **Within the Stamp budget.** Stay under the practical single-transaction limit (~65 KB) after minification.
- **Intellectual honesty.** Clearly separate what is cryptographically real from what is simulated.

2. Cryptographic Foundations

All cryptography uses the Web Crypto API (crypto.subtle), a standard component of the browser runtime. Because it is part of the platform, using it adds nothing to the file and requires no import—it is as built-in as JSON or Math. It is available in secure contexts (HTTPS, localhost, and local files), which is how stamped pages are served by Stamp explorers.

2.1 Wallets and addresses

Each account is a genuine ECDSA key pair on the NIST P-256 curve, generated locally. An address is derived deterministically from the public key as the string S followed by the first 38 hexadecimal characters of the SHA-256 hash of the raw public key. Private keys never leave the page and are revealed in the interface only on explicit request.

2.2 Transaction signing

A transaction is signed over a canonical JSON serialisation of its core fields (type, sender, recipient, amount, nonce, fee, timestamp, type-specific data, and the sender’s public key)—everything except the signature itself. The transaction identifier is the SHA-256 of that same canonical form, which binds the identifier to the content. Verification recomputes the hash, checks the signature against the embedded public key, and confirms that the address derived from that key matches the declared sender. A

tampered transaction fails verification; a transaction signed by the wrong key is rejected.

3. Ledger Model and Transactions

Stampcoin uses an account-based model in the style of Ethereum rather than a UTXO model. The global state—native balances, per-account nonces, token definitions and balances, assets, and liquidity pools—is not stored directly; it is *derived* by replaying every block's transactions from genesis. Anyone with the chain can recompute the exact state and verify it independently, which is the property that makes the ledger trustless in principle.

Each account carries a monotonically increasing nonce that must match the next expected value for a transaction to apply, preventing replay and fixing ordering. All monetary values are integers handled with a safe coercion that floors toward zero and preserves full precision up to 2^{53}, avoiding the silent 32-bit overflow that a naive bitwise conversion would introduce for large token supplies.

3.1 Transaction types

Type	Effect on state
coinbase	Mints the block reward plus the sum of block fees to the miner. One per block, first position.
transfer	Moves STC from sender to recipient; amount must be positive.
token_create	Defines a new token (ticker, name, supply) and mints the full supply to the creator.
token_transfer	Moves a positive token amount from sender to recipient.
asset_create	Mints a unique non-fungible asset owned by the creator.
asset_transfer	Transfers ownership of an asset; only the current owner may sign it.
pool_create	Locks STC and token liquidity into a constant-product pool.
swap	Trades STC for a token or vice versa against a pool.
pool_withdraw	Returns a pool's full reserves (deposit plus accrued fees) to its provider and closes it.

Table 1. The nine transaction types and their effects.

4. Blocks and Proof-of-Work

A block references its predecessor by hash, forming a tamper-evident chain: altering any historical block changes its hash and invalidates every block after it. Each block commits to its transaction set through a Merkle root computed over the transaction identifiers, and the block hash is the SHA-256 of its header fields (index, timestamp, previous hash, Merkle root, miner, difficulty, and nonce).

Mining is genuine proof-of-work. The miner increments the nonce and re-hashes until the block hash begins with a required number of leading zero hexadecimal digits—the difficulty, adjustable from 1 to 5. There is no shortcut; higher difficulty demands proportionally more hashing. The first transaction in every block is the coinbase, whose amount must equal exactly the block reward (50 STC) plus the sum of the fees of the other transactions; any other value is rejected (Section 6), which prevents a miner from minting coins out of nothing.

5. Tokens, Assets, and the Automated Market Maker

Any account may issue a token by specifying a ticker, name, and supply; the entire supply is minted to the creator and can then be transferred freely. Non-fungible assets are minted with a seed that deterministically generates a full-colour identicon rendered as inline SVG—the artwork is computed from on-chain data, so no image files are stored. Assets have a single owner and can be transferred.

Trading uses a constant-product automated market maker. A liquidity provider creates a pool by depositing STC and a token; the product of the two reserves defines the price curve. A swap of input amount x against reserves (R_{in}, R_{out}) yields, after a 0.30% fee, an output of floor($R_{out} \cdot x \cdot 0.997 \,/\, (R_{in} + x \cdot 0.997)$)—the same mechanism as Uniswap v2. The fee accrues into the reserves, so the product $k = R_{in} \cdot R_{out}$ never decreases through trading. Liquidity is genuinely earning: the provider may withdraw at any time and receives the full current reserves—their original deposit plus every fee that accumulated—after which the pool closes.

6. Validation and Security

Protocol rules are enforced at four independent layers, so that no single check is solely responsible for correctness and an invalid block can neither be stored by a node nor become canonical.

Layer	What it enforces
Structural verification	Block links to parent, hash is correct and meets difficulty, Merkle root matches, exactly one leading coinbase, no negative fees, each transaction identifier commits to its content, and each signature verifies (cached for performance).
Coinbase check (two gates)	The coinbase amount must equal reward plus total fees—checked both at block verification and again during state derivation. No inflation is possible.
Acceptance state-validation	Before a node stores or relays a received block, it replays the parent chain plus the block and rejects it if any transaction would be invalid (overspend, bad nonce, etc.).
Mempool admission	A transaction is replayed against the current state plus pending transactions and rejected up front if it could not be mined, keeping the mempool clean.
Ledger derivation	Replaying the chain enforces nonces, balances, positive amounts, ownership, and pool rules; any violation makes the chain invalid.

Table 2. The independent validation layers.

Concretely, the engine rejects negative or zero transfer and token amounts, negative fees, double spends, replays via the nonce, asset transfers by non-owners, second pools for an existing token, and any coinbase that does not equal reward plus fees. Native supply is conserved by construction: the only source of new STC is the block reward, and fees move from sender to miner, so the sum of all balances plus all pooled STC always equals 50 times the chain height.

7. The Node Network and Consensus

A node is a miner and validator that holds its own view of the chain as a set of known blocks and selects, as its head, the longest valid chain it can assemble—with a deterministic tie-break on block hash. When a node mines, it assembles valid mempool

transactions, constructs the coinbase, solves the proof-of-work, and broadcasts the block. Peers independently verify and, if the block extends a longer valid chain, adopt it—reorganising away from a shorter branch when necessary. When two nodes mine on the same parent, the chain forks, and the network converges once one branch grows longer. The interface visualises this with a live node graph and an event log that surfaces forks and reorganisations as they happen.

For performance at scale (up to 100 nodes), signature verification is cached per transaction across the network, and each node memoises its derived state by chain head. Measured cost is roughly 14 ms per block at 50 nodes and 18 ms at 100 nodes, growing gently with chain length; mining is throttled by the block interval, so this is imperceptible in interactive use.

8. The Embedded Economic Simulation

Because a single page cannot host a real network of strangers, Stampcoin includes an in-page economy. Two kinds of entity are distinct: **nodes** are miners/validators (up to 100), and **wallets** are economic actors—the user plus up to 30 autonomous bots. The only link between them is each node's payout address: the user's node pays the user's active account, while peer nodes pay bots in round-robin (or throwaway addresses when no bots exist). This is how bots are funded—they earn STC by mining—without any artificial faucet.

Each active bot is a full key pair that periodically performs a weighted-random action: transfer STC, buy or sell against a pool, transfer a token, issue a token, seed a pool, mint an asset, or gift an asset. Their transactions propagate to every node's mempool and are mined like any other, so the economy—prices drifting along pool curves, tokens and assets changing hands—evolves on its own.

8.1 User deletion semantics

Deleting bots illustrates a deliberate boundary between the immutable chain and a presentation overlay. Whatever a deleted user merely *held* that was created by a surviving account is returned to that creator through real signed transactions, settled in a block—so those holdings remain genuinely spendable. Whatever a deleted user *created* is wiped from the canonical view through a reconciliation overlay, because an append-only chain cannot retroactively un-issue what others may hold. The underlying chain is never rewritten; only the displayed state reconciles. This is explicitly a simulation convenience, not a claim about real blockchains.

9. What Is Real and What Is Simulated

A central design commitment is honesty about the system's boundaries.

Real (cryptographically genuine)	Simulated (modelled in one page)
ECDSA P-256 keys, signing and verification	Decentralised peer-to-peer networking
SHA-256 hashing and proof-of-work	Independent machines and real latency
Merkle commitments and block linking	Adversarial hashpower / Sybil resistance
Replay-derived ledger and supply conservation	Autonomous user (bot) intent — it is scripted
Constant-product AMM pricing	Economic scarcity and real value
Longest-valid-chain consensus logic	User-deletion cleanup (a display overlay)

Table 3. The boundary between genuine cryptography and simulation.

In one sentence: the cryptography, hashing, mining, ledger derivation, and the *logic* of consensus are real and would withstand scrutiny; what is simulated is the decentralisation—the trustless network of independent participants—together with bot behaviour and the deletion feature.

10. Persistence and Deployment

Because the page holds all state in memory, refreshing resets it. The entire world—wallets and their keys, bots, the deleted-user set, every node's chain, the mempool, and settings—can be exported to a JSON file and re-imported later to resume exactly where it left off. Keys are stored in JWK form so that signing continues to work after a reload.

The deliverable is a single HTML file of approximately 61.7 KB after minification, which fits within the practical ~65 KB budget of a single Stamp transaction (the larger OLGA encoding). Once stamped, the artifact is fixed and self-contained—there are no updates and nothing further to fetch. The only requirement to run it is a secure context, which Stamp explorers satisfy by serving over HTTPS.

11. Verification and Results

Correctness is established less by counting assertions than by the scale and variety of behaviour exercised while invariants are continuously re-checked. The engine is tested by a suite independent of the interface; the interface is driven in a headless DOM; a stress harness runs large randomised multi-node simulations; and an adversarial harness fuzzes malformed inputs. A single representative run mines on the order of 150 blocks and applies several hundred transactions spanning all nine types, and the stress suite repeats several such scenarios across multiple random seeds per invocation—so the cumulative exercised volume runs to thousands of transactions. Throughout, the same invariants are asserted: native and per-token supply are conserved, chains re-validate block-by-block, and the constant-product never decreases. Every suite passes, repeatably, on the exact minified artifact intended for deployment.

Suite	Coverage	Result
Engine	Signatures, tamper and key-mismatch rejection, proof-of-work, Merkle determinism, supply conservation, double-spend and nonce rejection, tokens, AMM invariant and withdrawal, asset ownership, fork and reorg consensus, coinbase and amount guards.	pass
Interface	End-to-end flows in a headless DOM: mining, multi-account sends, token and pool and swap, asset minting, chain validation, export/import round-trip, bot activity, deletion give-backs and wipes.	pass
Stress	50- and 100-node runs over many blocks; deletion mid-simulation; export/import after heavy activity; a 400-swap AMM fuzz; determinism. Verifies supply and token conservation, full chain re-validation, and the constant-product invariant.	pass
Adversarial	Hundreds of mutations on signed transactions (corrupted signatures, altered amounts/nonces/fees, forged senders, wrong keys, tampered data and type) and a dozen kinds of malformed block (bad Merkle root, broken links, forged proof-of-work, inflated coinbase, negative fees, duplicate coinbase) are all rejected. A structurally valid but overspending block is refused at acceptance.	pass

Suite	Coverage	Result
Precision	The AMM is probed at reserve scales from thousands up to billions. Balances are conserved exactly at every scale and reserves stay positive with no overflow.	pass

Table 4. Test suites and outcomes.

11.1 Threats to validity

Several boundaries are worth stating plainly. First, all automated testing runs in a headless DOM with Node’s Web Crypto rather than a production browser; this is a faithful proxy for the cryptography and logic, but a final manual pass in a real browser is still recommended before deployment. Second, the system trusts the platform’s Web Crypto implementation for the correctness of ECDSA and SHA-256 rather than re-deriving them. Third, the constant-product arithmetic is exact only while the product of a pool’s reserves stays below 2^{53} (about 95 million times 95 million); beyond that the pricing uses double-precision and loses exactness, though token and STC balances remain conserved regardless, and realistic pools sit far below this bound. Finally, the implementation is validated by testing, not by formal proof.

12. Limitations

Because the artifact is immutable once stamped, the following are permanent characteristics of this system rather than a roadmap:

- **No real peer-to-peer.** The nodes are in-page actors cooperating within one browser tab, not independent machines exchanging blocks over a network. There is no real latency, hashpower competition, or Sybil resistance.
- **Fixed difficulty and fee.** Difficulty and the per-transaction fee are constants; there is no difficulty retargeting and no congestion-based fee market, so fees are a flat charge rather than an economic signal.
- **Validation cost grows with history.** Block acceptance re-derives the parent chain, so per-block validation cost increases gently with chain length rather than being constant-time.
- **Single-provider pools.** Each liquidity pool has one provider and no fractional liquidity-provider shares, so liquidity cannot be pooled across multiple participants.
- **In-memory state.** All state lives in the page and is lost on refresh unless exported; persistence is manual via the export/import file.

13. Conclusion

Stampcoin demonstrates that a faithful blockchain—real keys, real signatures, real proof-of-work, a verifiable replay-derived ledger, an on-chain token and NFT economy, an automated market maker, and a fork-resolving multi-node consensus model—fits within a single self-contained HTML file small enough to stamp on Bitcoin. By enforcing rules at four independent layers and being explicit about the boundary between genuine cryptography and in-page simulation, it serves as both a working sandbox and an honest reference for how a chain operates mechanically. What it deliberately does not claim to be is a live decentralised network—and naming that boundary clearly is part of the contribution.

Stampcoin is an educational simulation. Generated keys are not secure for real value. “STC” is a simulated unit with no monetary worth.