DESIGN · SPV · DATA CARRIERS

The data carrier: proven without being mined, and unmineable because everyone receives it

Data published with Bitcoin can be checked by anyone, whether or not it was ever mined into a block. This paper walks through that check one picture at a time: what simplified payment verification (SPV) actually proves, the difference between data that is verifiable on chain and data that is verifiable off chain, where payments attach to each, and why a network that delivers every publication to many recipients at once may carry data only in transactions that cannot be mined. It closes with the whole of a third-party verification laid out step by step, and with what SPV cannot tell you.

Author: Jeff Harris · Lightweb Inc. · jeff@lightweb.net

Paper 011 · **Version** 01

Audience: application architects, developers, and anyone evaluating data applications built on Bitcoin. No Bitcoin background is assumed.

We = Lightweb Inc. and the 1BSV network; **you** = anyone handed data by a stranger and asked to believe it. Every figure uses the same colors: gray for the chain, sand for data that is never mined, purple for the parties, green for delivery, and a teal line for every proof.

The short version

- **Verifiable does not mean mined.** SPV needs three things: the bytes, a short Merkle path, and the chain's block headers (80 bytes each, about 4.2 MB a year). A transaction that is never mined is still verifiable when it spends an output of one that was.
- **Data can be verifiable on chain or off chain.** On chain, the data's own transaction is in a block: permanent, ordered, paid for by the byte, never retractable. Off chain, the data rides a carrier that is never mined but spends a mined parent: provable, free to carry, and retractable by one mined spend.
- **Mineable and unmineable are two fields apart.** A lock time of 1 January 2100 and an input marked "not final" make a transaction invalid in any block before then. That is a consensus rule every node enforces, not a fee policy a miner may waive.
- **On a wide-recipient network, whatever is mineable is mineable by every recipient.** A signed transaction is its own authority to settle: whoever holds the bytes can hand it to a miner. Multicast hands the bytes to everyone at once, so the publisher's choice of whether and when to settle ends at the first recipient.
- **So a shared plane carries nothing whose settlement is still a choice.** Data rides unmineable carriers; state rides transactions their publisher has already broadcast; a payment travels sealed inside an encrypted carrier that only its payee can open.
- **Payments attach at four points.** Postage (a mined funding tree pays once for many carriers), commitment (a mined state token names a carrier), the sealed payment (value delivered with a message), and the priced read (a payment for an answer). Data never settles; payments always do.
- **Third-party verification is a fixed checklist of eleven steps.** Nine are arithmetic on bytes already in hand; one is the host's answer and one asks the header sources you chose. A carrier on our published test vector is 823 bytes as delivered, proofs included.
- **SPV proves inclusion, not currency.** It cannot say whether an output is still unspent, so a retraction is seen by a host that watches the chain or by asking a chain view or index, never by SPV alone.

1. Three questions a verifier asks

Someone hands you some bytes and says: this is what Alice published. Before you act on it, you need three answers, and Bitcoin gives you a way to get each one without trusting the person who handed you the bytes.

Figure 1 - the three questions. The first two are settled by arithmetic on bytes in hand; only the third needs something from outside, the block headers, and those you fetch once from a source you choose.

The rest of this paper is those three questions, asked of two different kinds of data. A few terms recur, so here they are once:

Term	Plain meaning
transaction	a signed message that spends earlier outputs and creates new ones
output	an amount locked by a small script; it can be spent exactly once
txid	the double SHA-256 of a transaction's bytes: a 32-byte name for exactly those bytes
block header	an 80-byte summary of a block; it holds the Merkle root of the block's transactions
Merkle path	the sibling hashes that link one txid to its block's Merkle root (BUMP, BRC-74)
SPV	checking a transaction against block headers alone, without the full chain (BRC-9, BRC-67)
BEEF	an envelope holding a transaction, its ancestors and their Merkle paths (BRC-62, BRC-95)
mined	included in a block
unmineable	invalid in any block until a set date, by a rule every node enforces
carrier	an unmineable transaction whose output holds the data

2. SPV in four pictures

2.1 Headers: the one thing you must hold

A block header is 80 bytes. Each one names the header before it and carries the proof of work spent to make it, so a chain of headers can be checked for that work by anyone. At about 144 blocks a day the headers grow about 4.2 MB a year, and the whole history since 2009 is roughly 75 MB: small enough for a phone.

Choosing where your headers come from, and checking their work, is the only act of trust in SPV. Everything else is arithmetic. That trust is easy to spread: take headers from several sources, served by a plurality of independent nodes, and accept only the chain they agree has the most work.

2.2 The Merkle path: from a transaction to a header

Every header holds a Merkle root: the top of a tree of hashes built over every transaction in its block. To show that one transaction is in that block you need only the hashes beside its path to the top, one per level. At a million transactions in a block that is about 20 hashes, some 640 bytes, however large the block is.

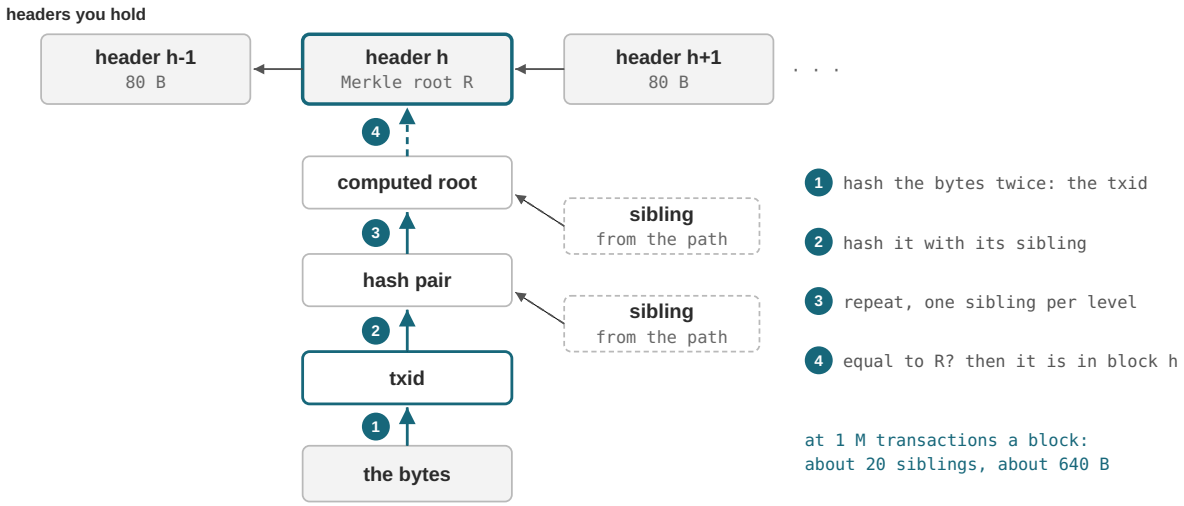


Figure 2 - SPV for a mined transaction. Hash the bytes into a txid, hash up the path one sibling at a time, and compare the result with the root in a header you already hold. Equal means the transaction is in that block.

2.3 Scripts: why a spend proves itself

An output is locked by a small script, usually "a signature by this key". A transaction that spends it supplies the unlocking half, usually that signature. The verifier runs the two halves together; if they come out true, the spend was authorized by whoever held the key. It also checks that the new outputs do not add up to more than the inputs.

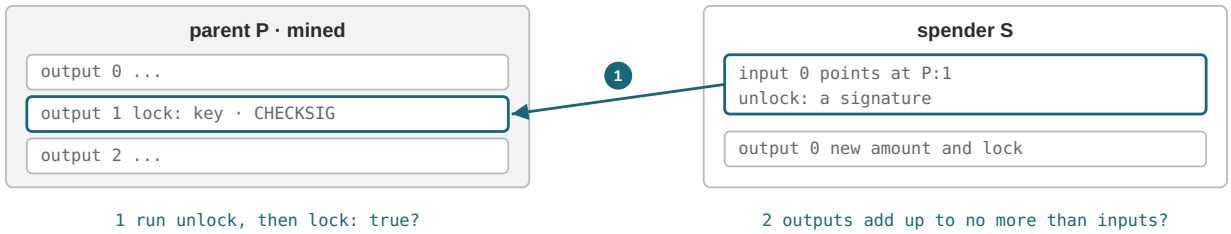


Figure 3 - a spend. The spender's input points at one output of its parent and supplies the signature that output asks for. This is how a transaction that is never mined still proves itself: through a parent that was.

2.4 BEEF: everything but the headers, in one envelope

A verifier should not have to go looking for parents and paths. BEEF packs the subject transaction, the ancestors it needs, and their Merkle paths into one envelope; Atomic BEEF names exactly one subject. With the headers you already hold, a BEEF is complete: nothing else is fetched.

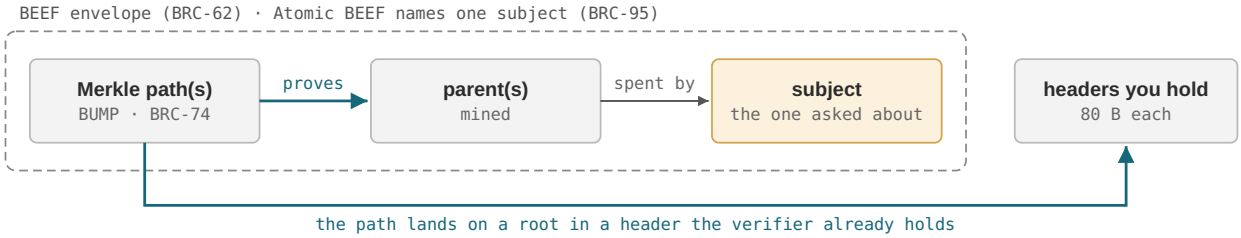


Figure 4 - a BEEF envelope. The path proves the parent; the parent's output is what the subject spends. The headers stay with the verifier.

3. On-chain and off-chain verifiable data

Data can sit in one of two places and still be checked by a stranger. The difference is not whether it can be proven; both can. The difference is what proving it costs, how long it lasts, and whether anyone can take it back.

3.1 On chain: the data's own transaction is mined

Put the data in an output of a transaction and have it mined. Its proof is its own Merkle path. It is ordered by its place in the chain and it is permanent: every full copy of the chain keeps it, it is paid for by the byte, and nobody, its publisher included, can retract it.

3.2 Off chain: the data rides a carrier that spends a mined parent

Put the data in an output of a transaction that is never mined, the **carrier**, and have the carrier spend one output of a small transaction that *is* mined, the **funding tree**. The carrier's proof is its parent's Merkle path plus one script check (Figure 3). Its txid names the data exactly. Nothing is paid to miners per byte of data, and the publisher can take it back: spend the funding output yourself, in a mined transaction, and the carrier becomes a double spend of an output already spent.

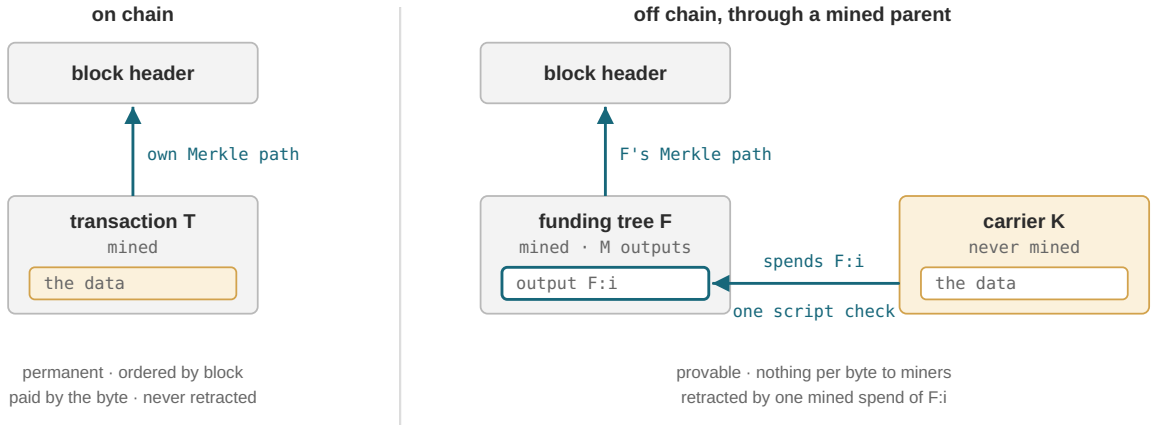


Figure 5 - the same three questions, two placements. On chain, the data's transaction carries its own proof. Off chain, the carrier borrows its proof from a mined parent through one script check, and stays retractable.

3.3 Both: commit on chain, carry off chain

The two combine. A **state token** is a tiny mined output that holds one thing, the carrier's txid, and each update spends the previous token. The chain then supplies what only a chain can supply, order and a permanent 32-byte name, while the data itself stays off chain where it remains retractable. We call this "mine the state, not the data".

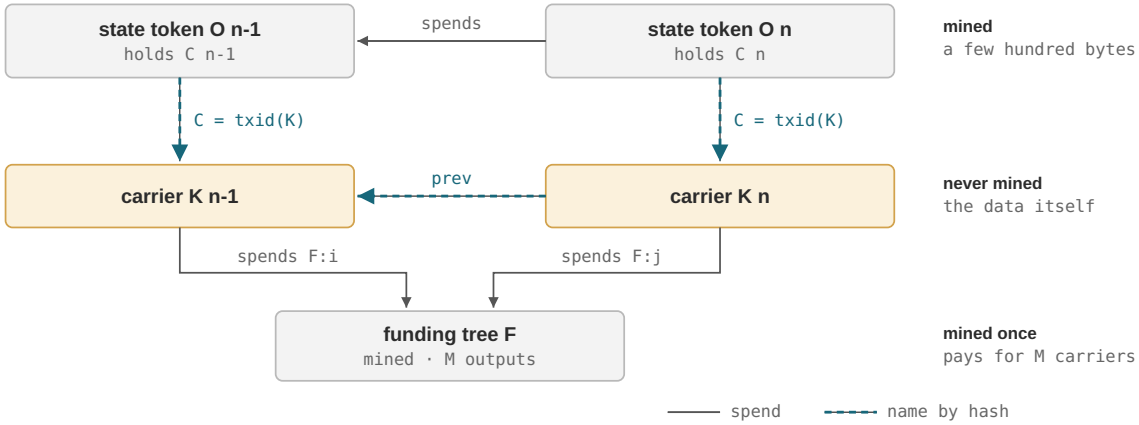


Figure 6 - commit on chain, carry off chain. Solid lines are spends; dashed lines are names by hash. Each token names one carrier, each carrier names the one before it, and every carrier proves itself through the same mined funding tree.

	On chain	Off chain (carrier)	Both
Where the data lives	in a block	in hosts' storage, inside the carrier	in hosts' storage; its 32-byte name in a block
Proven by	its own Merkle path	the parent's path and one script check	both
Miner fees	per byte of data	none for the carrier; the parent pays once for M carriers	one token per update, plus the parent
Ordered by	block position	the signed payload (sequence, previous name)	the chain of tokens
Retractable	never	yes, by one mined spend	the data yes; the fact of the commitment no
On our test vector	not applicable	823 B carrier as delivered	plus 327 B (create) to 440 B (update) mined

Proof comes from the chain either way; permanence is a choice the publisher makes.

4. Mineable and unmineable: two fields apart

Every transaction carries a **lock time**. Zero means no lock. A value of 500,000,000 or more is read as a calendar time, and a transaction is allowed into a block only once that time has passed (measured against the median time of the last eleven blocks), unless every one of its inputs is marked final. An input is final when its **sequence number** is the maximum, `0xFFFFFFFF`.

The carrier sets its lock time to 4,102,444,800, which is 1 January 2100 at midnight UTC, and its one input's sequence number to zero. That pair makes it invalid in any block before 2100. Every node refuses it, and a block that included it would be refused too. The device is BRC-60's, turned the other way round: BRC-60 keeps a transaction open so that it can be mined later; the carrier keeps it open so that it never is.

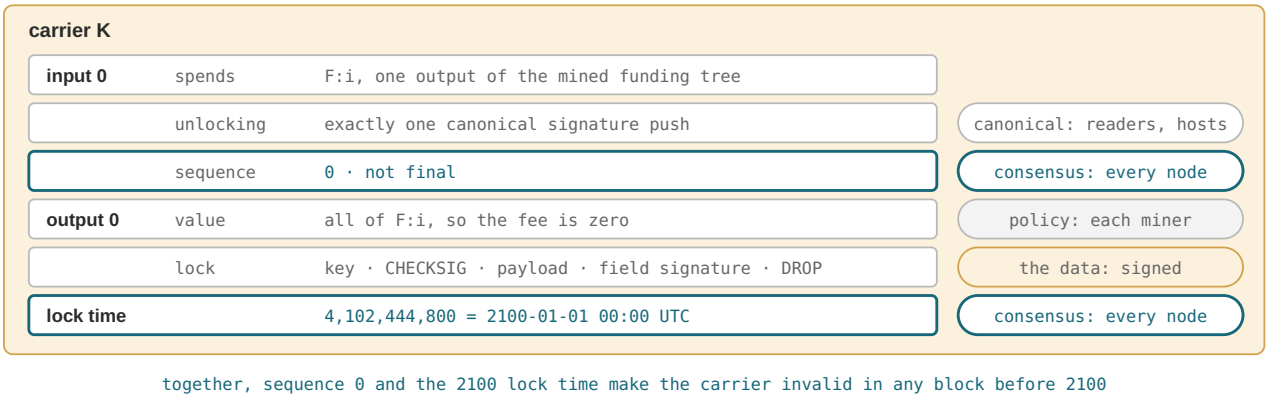


Figure 7 - a carrier, field by field. The two highlighted fields together are what make it unmineable; the rest make it provable and make it the publisher's alone.

Field	Value	Enforced by	What it does
lock time and sequence	2100, and 0	**consensus**: every node	no block may include the carrier before 2100
fee	zero	miner policy only	no miner wants it, but a miner may choose to take a zero-fee transaction
unlocking script	one canonical signature push	the verifier and every host	a twin re-encoded by anyone else has a different txid and is refused, so it retracts nothing
field signature	over SHA-256 of the payload	the verifier	the payload is the key holder's, independent of the spend

Fee zero is a preference; the lock time is a rule. That is why the carrier needs both.

SPV still accepts a carrier: it checks scripts, amounts and proofs, not finality, and the SDKs' SPV paths do not read the lock time. BRC-67 does list among SPV's criteria that "nLocktime, and nSequence of each input are set to the expected values". For a payment, the expected value is final. For a carrier it is the opposite, so the verifier checks it itself: a carrier that *could* be mined is refused, by every reader and by every conforming host. (BRC-67's fee check, inputs greater than outputs, is a check on payments; a carrier is not one, pays nothing by design, and its outputs never exceed its inputs.)

5. Why a wide-recipient network carries nothing mineable

5.1 The bytes are the authority to settle

A signed transaction needs nothing more from its signer to be settled. Anyone who holds the bytes can hand them to a miner. Between two parties that is the point: you pay someone by giving them a transaction, and they broadcast it. Multicast changes the arithmetic. The network copies one publication to every subscriber of its topic, whether that is one host or thousands, and whatever they serve downstream, and every one of them now holds the same authority. The publisher cannot know how many are listening, so the rule cannot depend on the count.

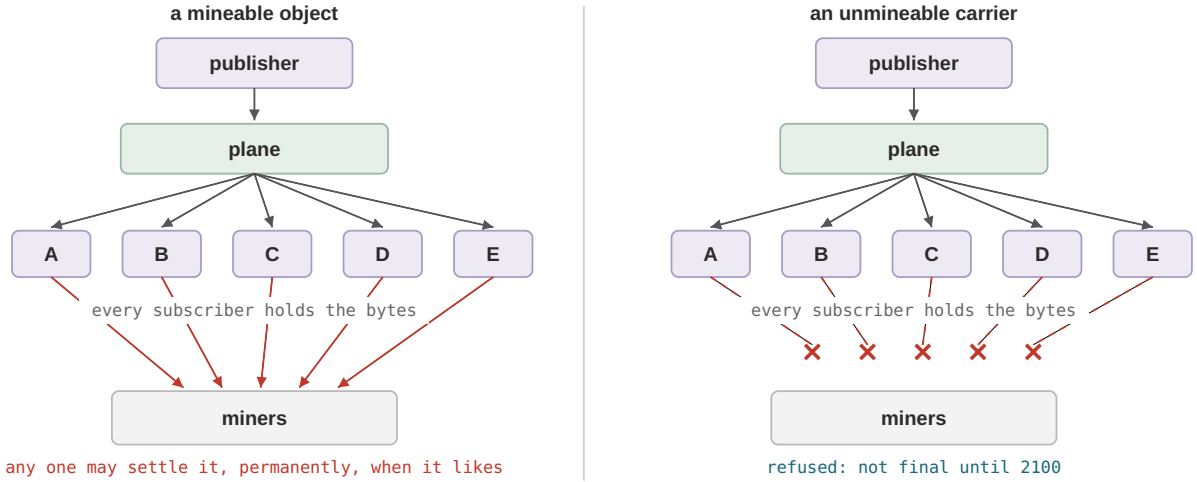


Figure 8 - one publication, many holders. On the left every recipient can settle the object, at a moment of its choosing. On the right every attempt is refused by consensus, and the recipients simply hold data they can verify.

5.2 What a recipient could do with a mineable data object

1. **Make the data permanent.** Mine it, and it lives in every full copy of the chain forever. The publisher's ability to retract it is gone.
2. **Win the retraction race.** A retraction spends the funding output. A recipient who mines the carrier first spends that output instead, and it is the publisher's retraction that is refused as a double spend (Figure 9).
3. **Choose the moment.** When the transaction settles becomes the recipient's decision, not the publisher's.
4. **Load the chain with the bulk.** Data meant for hosts' storage lands in blocks, at whatever rate the plane carries it.

None of this needs malice. A misconfigured relay that forwards objects to a miner does the same thing. And fee zero does not close the door. The carrier's one input signs the whole transaction, so nobody can add an input that pays a fee; its one output is locked to the publisher's key, so nobody can attach a child transaction that pays for it. That leaves only a miner's own choice to take a zero-fee transaction, which a miner is free to make. The lock time removes that choice from every miner at once.

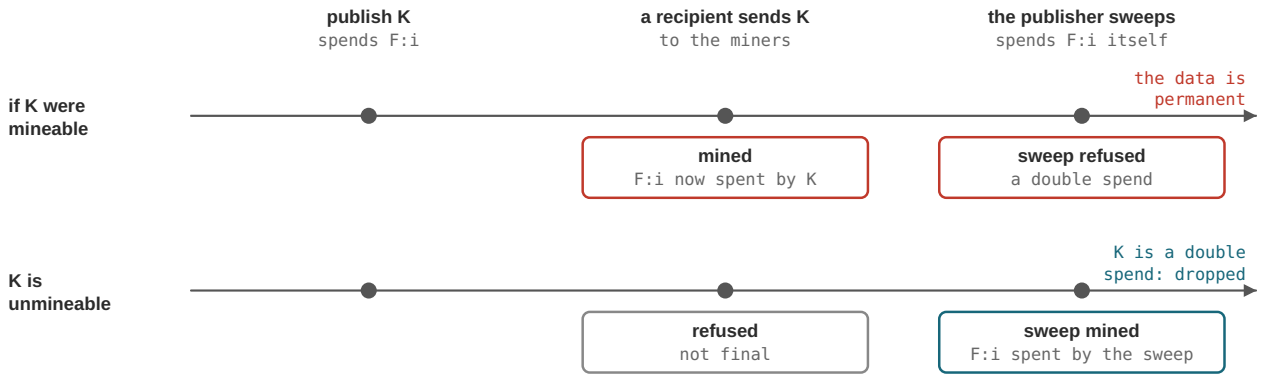


Figure 9 - the retraction race. With a mineable carrier any recipient can spend the funding output first and make the data permanent. With an unmineable one only the publisher's own spend can ever reach a block.

5.3 Two planes, one fabric

The 1BSV fabric already keeps the two kinds of traffic apart. BRC-148 divides the multicast address space into planes: the **settlement plane** carries raw transactions to miners and settlement infrastructure, and the **BEEF object plane** carries self-proving objects to overlay hosts, wallets and indexers. Their publishers and subscribers are different by design. Unmineability makes a leak between them harmless: an object that strays onto a miner's path is refused by the first node that sees it.

5.4 The rule, stated precisely

What travels	Mineable?	Safe to send to everyone?	Why
a carrier (data)	no	**yes**	no recipient can settle it
a state token or funding tree, after its publisher broadcast it	yes	**yes**	its fate is already decided; a rebroadcast is the same transaction
a payment not yet broadcast, in the clear	yes	**no**	any recipient could broadcast it first, and the payer loses the right to reclaim it
a payment sealed inside an encrypted carrier	yes, inside	**yes**	only the payee can read it, so only the payee can settle it

On a shared plane nothing travels whose settlement is still a choice, unless only one party can read it.

6. Where payments attach

Data never settles. Payments always do. The design question is only where the two meet, and there are four places.

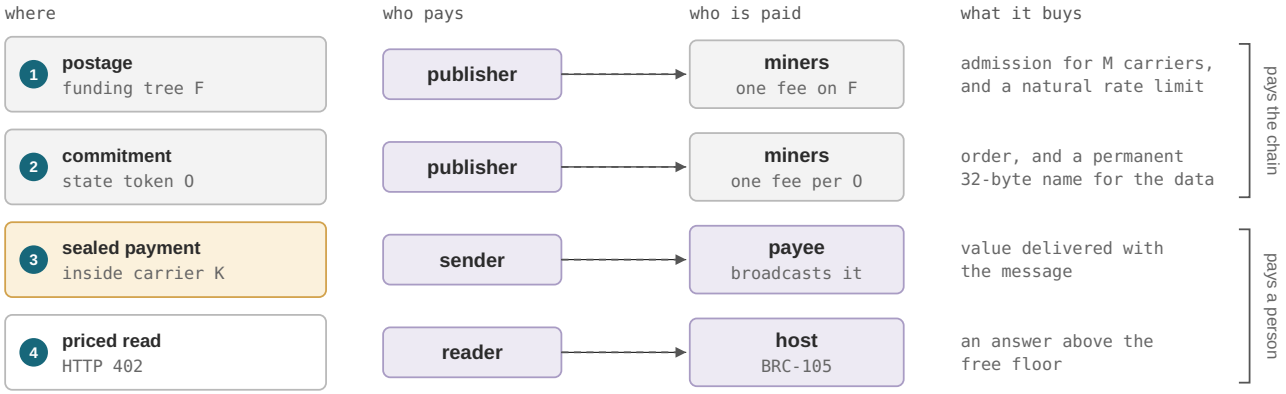


Figure 10 - the four places a payment meets data. Dashed arrows are value, from the payer to the paid. Two pay miners for the chain's work; two pay a person or a host for theirs.

Postage. One mined funding tree with M small outputs pays one miner fee, and each output admits exactly one carrier. No unspent output, no publication: flooding costs real money at the door, with no rate-limiter service to run.

Commitment. A state token is an ordinary mined transaction and pays an ordinary fee. What it buys is order and a permanent name. On our test vector a create is 327 bytes and an update 440.

The sealed payment. A payment is mineable on purpose: the payee is meant to broadcast it. BRC-169 says the sender must not broadcast it; the payment travels inside a message and the recipient broadcasts it on internalization. On a shared plane that payment rides inside an encrypted carrier, so hosts and other subscribers see only ciphertext; our message box application, bbox, works exactly this way. Two time windows keep both sides honest: the message lives at least two hours, the payee takes the payment only until an hour before it expires, and the sender may reclaim the coins only from an hour after it expires.

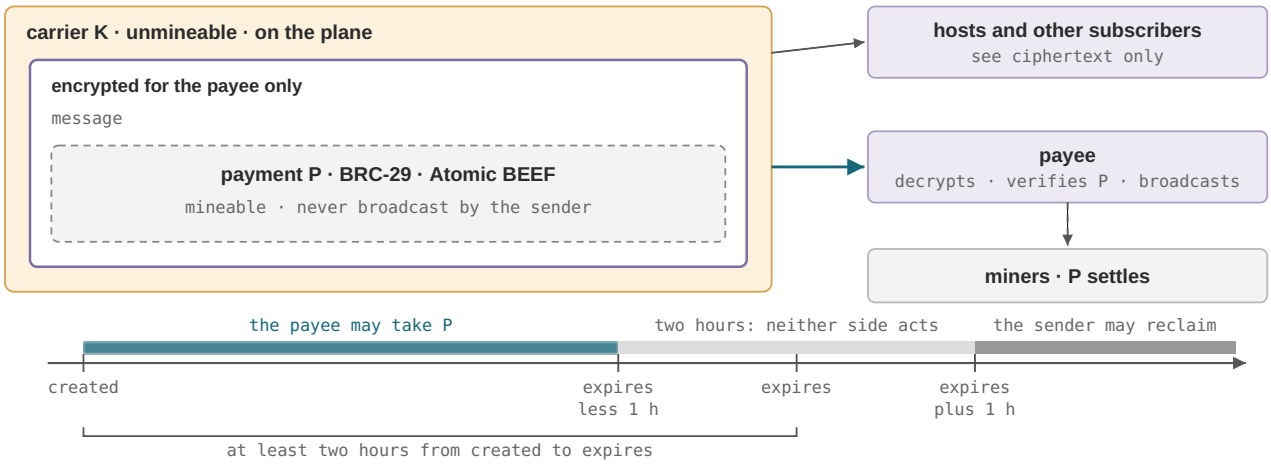


Figure 11 - a sealed payment. The payment is mineable, but only the payee can read it, so only the payee can settle it. The carrier around it is unmineable, so nobody can settle the envelope either.

The priced read. The base answer from a host is free; richer questions (history, proofs, keyed content) can be priced. The reader pays the host over HTTP with a BRC-105 payment, one to one, never through the plane.

Attachment	Mineable?	How it travels	Who broadcasts	What it buys
postage	yes	to miners first, then on the plane	publisher	admission for M carriers
commitment	yes	to miners first, then on the plane	publisher	order and a permanent name
sealed payment	yes, sealed	inside an encrypted carrier	payee	value delivered with a message
priced read	yes	one to one, to a host	host	an answer above the free floor

Data never settles, payments always do, and on a shared plane a payment travels sealed.

7. Third-party verification, step by step

7.1 Who does what

Besides the verifier, five parties take part, and the verifier trusts only one of them: its header sources, ideally several, run by independent nodes. Figure 12 is the life of one record, from the funding tree to the verifier's verdict.

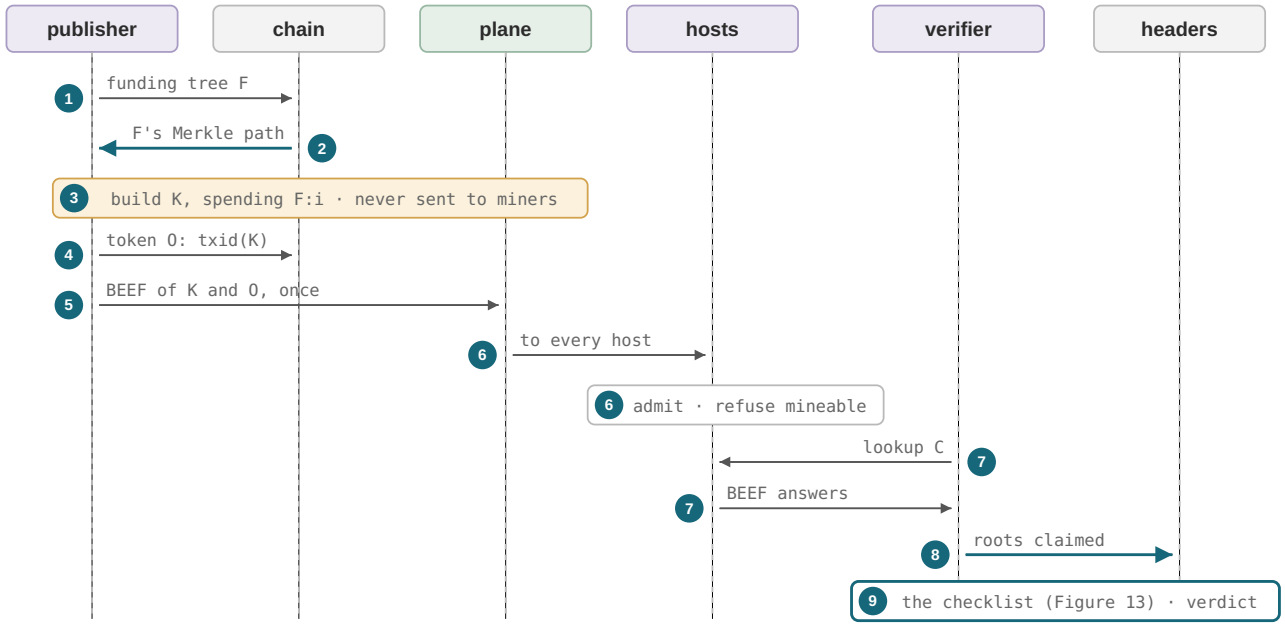
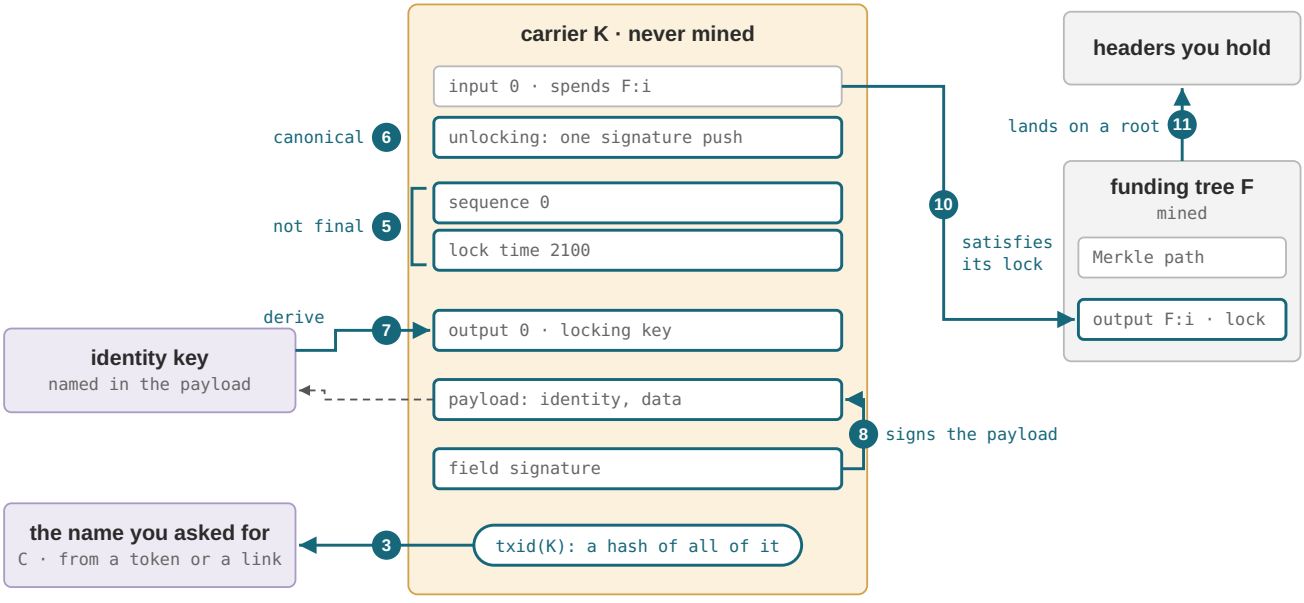


Figure 12 - one record's life. The chain sees the funding tree and the token; it never sees the carrier. The plane delivers once to every host. The verifier asks any host, then checks everything against its own headers.

7.2 What proves what

Figure 13 is the heart of the paper: every dependency the verifier follows, numbered to match the checklist that comes after it. Read it from the name you asked for, at the bottom left, to the headers you hold, at the top right.



not drawn, checks on the whole: 1 one answer · 2 the BEEF parses · 4 the payload's rules · 9 the application's expectations

Figure 13 - the dependency map. Each numbered line is one check, and each check uses only the boxes it joins; steps 2, 4 and 9 read the payload as a whole. Steps 2 to 10 are arithmetic on the bytes the host sent; step 11 is the one that reaches your headers.

7.3 The checklist

#	Check	Uses	If it fails[1]
1	the host answered exactly one output for the name asked	the host's answer	nothing found, or a fork
2	the BEEF parses within size bounds and holds one carrier	the BEEF	refused: decode
3	the carrier's txid is the name you asked for	the carrier's bytes	refused: commitment
4	the payload follows its application's rules	the payload	refused: decode
5	**the carrier is unmineable**: lock time at least 2100, every input not final	two fields	**refused: mineable**
6	the input's unlocking script is exactly one canonical signature push	the input	refused: unlocking
7	the locking key is derived from the identity the payload names	identity key, lock	refused: key derivation
8	the field signature verifies over SHA-256 of the payload	key, payload	refused: signature
9	the application's own expectations (kind, identity)	the payload	the application's refusal
10	the input satisfies the funding output it spends	carrier, F:i	refused: signature
11	the funding tree's Merkle path lands on a root in your headers	path, headers	refused: proof; no answer from the header sources is an error, never a verdict

1. Refusal names are the reader's codes in plain words. The same checks run in Go and TypeScript, against the same published test vectors.

Three choices in the order are deliberate:

- **The name comes first.** A host answering a different, perfectly valid carrier than the one you asked for is the case step 3 exists for; every later check would pass on it.
- **Content before proof.** Steps 2 to 9 cost nothing but arithmetic, so a carrier refused on its content costs the verifier no header lookup.
- **Refused for the first rule it breaks.** A carrier that breaks two rules always gets the same refusal, so two implementations agree on why, not just on whether.

With a state token, the token is checked too. It proves itself by its own Merkle path before the carrier is looked at, the name C that step 3 compares against is read from it, and its lock and field signature are checked the same way as steps 7 and 8. **For data on chain,** the list is short: the txid, the transaction's own Merkle path, the signature, and the payload's rules.

8. What SPV cannot tell you

1. **Whether an output is still unspent.** SPV proves that a parent was mined, not that its output is unspent today. A retracted carrier served from a stale copy still passes steps 2 to 11. You see a retraction by asking a host that watches the funding tree, or a chain view or index that answers who spent F:i.
2. **Whether a host is withholding.** A host can hold back an answer; it cannot forge one. Ask more than one host and compare, and refuse answers that disagree.
3. **Whether your headers are honest.** Check their proof of work, take them from sources you chose, preferably several served by a plurality of independent nodes, compare them, and never take them from whoever answered your question.
4. **Whether the data is true.** A signature proves who said it. It does not prove that what they said is so.

9. Assumptions and limits

Quantity	Value	Basis
Block header	80 B	**structural**: the Bitcoin header format
Header growth	about 4.2 MB a year; about 75 MB since 2009	**modeled**: 144 blocks a day at 80 B, over 17.8 years
Merkle path at 1 M transactions a block	about 20 hashes, about 640 B	**structural**: log2(10^6) is 19.9 levels of 32 B
Carrier lock time	4,102,444,800 (2100-01-01 00:00 UTC)	**verified**: bcommon `carrier.LockTime`
Carrier input sequence	0	**verified**: bcommon `carrier.Sequence`
Carrier fee	0	**verified**: `carrier.Mint` carries the whole input value to output 0
Carrier as delivered	823 B Atomic BEEF	**measured**: bfinger's published test vector
Token per transition	327 B create, 440 B update	**measured**: the same vector
Retraction (sweep)	398 B	**measured**: the same vector
Funding tree	16 outputs by default; at most 1,023	**verified**: bfinger default; bcommon `mint`
Sealed-payment windows	at least 2 h of life; payee until expiry less 1 h; payer from expiry plus 1 h	**verified**: bbox specification §10
SPV does not read finality	not applicable	**verified**: go-sdk v1.5.2 and v1.7.1, and the TypeScript SDK's unmined path

Limits, stated plainly:

1. **Unmineable until 2100, not forever.** Once the chain's median time passes 1 January 2100, a carrier whose funding output is still unspent becomes mineable. The date is frozen at the first mint and a reader refuses any lower one; a publisher can sweep before then, and a format meant to outlive the date needs a later one or a different device.
2. **Unmineable is not private.** The lock time stops settlement, not reading. Anything that must stay private is encrypted, as the sealed payment is.
3. **A carrier proves its parent, not its own time.** The funding tree's proof dates the tree, not the carrier. Order and time come from the signed payload, or from a mined commitment such as the state token.
4. **Retraction is enforced by honest hosts, not by SPV.** A retraction makes every conforming host drop the record. A copy kept elsewhere still passes SPV; only a spentness check (§8, item 1) reveals that it was withdrawn.
5. **The verifier trusts its header sources.** Wrong headers give wrong verdicts. Proof of work bounds the damage, several independent sources outvote one dishonest or lagging one, and choosing them is the verifier's own act.
6. **The sizes are one application's.** They are measured on one published test vector; another payload changes the carrier's size, not its shape or its checks.

References

Satoshi Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*, §8 Simplified Payment Verification: https://bitcoin.org/bitcoin.pdf

BRC-9, *Simplified Payment Verification*: https://github.com/bsv-blockchain/BRCs/blob/master/transactions/0009.md

BRC-67, *Simplified Payment Verification*: https://github.com/bsv-blockchain/BRCs/blob/master/transactions/0067.md

BRC-74, *BSV Unified Merkle Path (BUMP) Format*: https://github.com/bsv-blockchain/BRCs/blob/master/transactions/0074.md

BRC-62, *Background Evaluation Extended Format (BEEF) Transactions*: https://github.com/bsv-blockchain/BRCs/blob/master/transactions/0062.md

BRC-95, *Atomic BEEF Transactions*: https://github.com/bsv-blockchain/BRCs/blob/master/transactions/0095.md

BRC-60, *Simplifying State Machine Event Chains in Bitcoin*: https://github.com/bsv-blockchain/BRCs/blob/master/state-machines/0060.md

BRC-48, *Pay to Push Drop*: https://github.com/bsv-blockchain/BRCs/blob/master/scripts/0048.md

BRC-42, *BSV Key Derivation Scheme (BKDS)*: https://github.com/bsv-blockchain/BRCs/blob/master/key-derivation/0042.md

BRC-29, *Simple Authenticated BSV P2PKH Payment Protocol*: https://github.com/bsv-blockchain/BRCs/blob/master/payments/0029.md

BRC-105, *HTTP Service Monetization Framework*: https://github.com/bsv-blockchain/BRCs/blob/master/payments/0105.md

BRC-169, *Universal Handle Addressing and Resolution for the Metanet*, §6 Payments: https://github.com/bsv-blockchain/BRCs/blob/master/peer-to-peer/0169.md

BRC-148, *Multicast Shard Domain Partitioning and the BEEF Object Plane*: https://github.com/bsv-blockchain/BRCs/blob/master/transactions/0148.md

The overlay object plane (Lightweb Inc., Paper 006): https://1bsv.net/papers/overlay-object-plane.pdf

The bstack: publish once, prove everything, and let users ride free (Lightweb Inc., Paper 009): https://1bsv.net/papers/bstack-patterns.pdf

bcommon, the carrier and its verification (Lightweb Inc.): https://github.com/lightwebinc/bcommon

---

1BSV - BSV Layered Multicast - cash moves fast on 1BSV. · 1bsv.net

---

Lightweb Inc. · Jeff Harris · jeff@lightweb.net · github.com/lightwebinc

Unbounded Solutions for a Small World™ · © Lightweb Inc.

© 2026 Lightweb Inc. All rights reserved. 1BSV and Unbounded Solutions for a Small World are trademarks of Lightweb Inc. This paper describes an architecture in rapid development: the design, its interfaces and this document are subject to change without notice, and later versions supersede this one. Nothing in this paper is a commitment to ship any feature, an offer of service, or professional advice.