OpenBazaar: Signed Catalogs, Sealed Orders, and Scriptless Two-Party Escrow for a Non-Custodial Marketplace

Brian Hoffman
OpenBazaar project

November 2026

Abstract

OpenBazaar is a peer-to-peer marketplace for ordinary physical and digital goods paid in bitcoin. The original system (developed from 2014, with its last substantial releases in 2020) published stores over IPFS, carried orders over libp2p, and protected buyers with a 2-of-3 multisignature escrow whose script and moderator key appeared on chain when it was spent. This paper presents a redesign of the network and argues which of its security and privacy properties hold, and under which assumptions.

Stores and listings are public, content-addressed records signed with BIP 340 Schnorr signatures over a canonical encoding, and carry the vendor's own content labels; third parties can publish signed label sets that buyers choose to follow, and all filtering happens on the buyer's device. Orders are private letters sealed with HPKE to a single vendor and delivered to that vendor's onion-service mailbox; a client device talks only to its owner's node. Payments go directly to the vendor, one transaction per seller. Moderated purchases use a scriptless escrow: a MuSig2 Taproot key-path output whose timeout and dispute spends are all presigned before funding. A dispute is settled by an adaptor signature whose completing secret is split between the favored party and a moderator half that the losing party verifiably encrypts to the moderator. The moderator never holds a signing key over the funds and cannot change any payout.

We state the threat model, specify the escrow protocol with the checks each party must run and its block-height timeline, argue the security properties, tabulate what every observer learns, and state the limitations and open problems, including a moderator-fee bias, the size of the verifiable encryption, and collusion between the moderator and one party. Measurements of a reference implementation show that catalog signing, verification, and order sealing cost well under a millisecond, and that the escrow's verifiable encryption costs about 50 KB and 50 ms per share.

1 Introduction

A marketplace has to do four things: let sellers publish what they offer, let buyers find it, carry an order from buyer to seller, and settle payment in a way both sides can accept. Hosted marketplaces do all four in one place. That place can see every order, hold every payment, and remove any seller. OpenBazaar's premise is that these functions can be split so that no single operator is needed for any of them, and so that the parties who must learn something (the seller who ships, the moderator who rules on a dispute) learn it only when they need to.

1.1 The original OpenBazaar

OpenBazaar began in 2014 as a prototype peer-to-peer market and was developed as `openbazaar-go` and a desktop client until late 2020 [39]. A node published its store as an IPFS directory addressed

by IPNS [5]. Listings were signed by the node's identity key. Messages traveled over libp2p when both peers were online; when the recipient was offline, the sender stored the message on its own node or an external storage host and left a pointer in the distributed hash table. Search was a separate hosted index that nodes notified on publish. The wallet supported several cryptocurrencies. Moderated purchases used a 2-of-3 multisignature script over buyer, vendor, and moderator keys, with a default timeout of 1080 hours.

That design had strengths the redesign keeps: listings were signed and an order committed to a hash of the listing it bought, so later edits could not rewrite a deal; the moderator could not take funds alone. It also had properties the redesign changes:

- A store was available only while its own node, or a pinning service, was online.
- The libp2p peer identity was long-lived and observable, and peers dialed each other directly, so the network layer linked a store identity to IP addresses.
- The default search index was a single hosted service. Delisting from it removed a store from practical discovery.
- Spending a 2-of-3 P2SH or P2WSH output reveals the full script, including the moderator's public key, so every moderated purchase was recognizable on chain and attributable to its moderator.
- A phone could not run the daemon in the background, so a vendor without an always-on desktop missed orders.
- The multi-coin wallet made wallet maintenance the dominant engineering cost.

1.2 This paper

The redesign keeps the contract between buyer, vendor, and moderator and replaces the machinery underneath. Its main ideas are:

1. **A public catalog, private orders.** Store and listing records are public, signed, and content-addressed, and replicated by any node (§6). Orders are not catalog records; each is a letter sealed to one vendor and carried to that vendor's own mailbox (§8). Vendors label their own records, and buyers filter what they see on their own devices (§6.4).
2. **Clients that only talk to their own node.** The user-facing app never names a peer, seed, onion address, or vendor URL. It sends requests to its owner's node, which verifies records and delivers letters (§5).
3. **Direct, per-seller payments.** A buyer pays each vendor in a separate transaction, never batching two sellers together (§9).
4. **Scriptless two-party escrow with a bounded moderator.** The escrow output is an ordinary-looking Taproot key-path output. Every non-cooperative outcome is presigned by both parties before the buyer funds it. The moderator only reveals one secret scalar that completes one of two presigned dispute outcomes (§10).

The contributions are: (i) a threat model for a marketplace with this split (§4); (ii) a precise description of the two-party escrow protocol, its message rounds, the checks each party and the moderator must run, and its block-height timeline (§10); (iii) argued security properties, each with its assumptions, and the attacks the checks exist to stop (§12); (iv) a table of what every party and observer learns (§13); and (v) measurements of a reference implementation (§14) and a statement of what the design does not achieve (§15).

A reference implementation in Rust, with a web and desktop client, is available at `https://github.com/OpenBazaar/openbazaar-rs`. It is a testnet prototype; this paper describes the design it implements, not its deployment state.

2 Design Goals and Non-Goals

2.1 Goals

G1. No custody. No party other than the buyer and the vendor ever holds a key, or a set of keys, sufficient to move a buyer's payment. The moderator in particular holds no signing key over escrowed funds.

G2. Verifiable public catalog. Anyone can check that a listing was published by the store it names and has not been altered. A listing's address is derived from its content.

G3. Private orders. An order's contents, including any ship-to address, are readable only by the vendor it is addressed to. No public relay or index stores a record that a given buyer wrote to a given vendor.

G4. Bounded moderation. A moderator can choose between two payouts that buyer and vendor already signed. It cannot sign alone, redirect funds, change amounts, or learn about an order unless a party opens a dispute.

G5. Liveness without a platform. A buyer can recover escrowed funds from a vendor who disappears, and a vendor can collect from a buyer who stops responding, without anyone's cooperation beyond broadcasting a presigned transaction.

G6. Thin clients. A phone or browser client does not need to run Tor, hold spend keys, or dial the peer-to-peer network. Those jobs belong to a node the user controls.

G7. Honest degradation. When a privacy mechanism is unavailable (no Tor, no Payjoin partner, no Lightning route), the system says so rather than silently falling back.

2.2 Non-goals

- **Anonymity against a global passive adversary.** We do not claim unlinkability against an adversary that observes most of the network. Section 13 states what such an adversary learns.
- **Hiding the catalog.** Listings are public by design. A market that cannot be discovered does not serve its purpose.
- **Hiding the ship-to address from the seller or the carrier.** A seller of a physical good must learn where to send it, and the carrier must deliver it. This is a ceiling on buyer privacy for physical goods, not a gap to be engineered away.
- **Deciding factual disputes cryptographically.** Whether a parcel contained the right item is a judgment by a person. The protocol bounds what that person can do with the funds; it does not replace the judgment.
- **Reputation.** A review or rating system is outside this paper.
- **Content policy.** The protocol is designed for ordinary goods. Policies for what may be listed, and the obligations of people who operate replicas or seeds, are outside the scope of this paper.

3 System Model and Actors

Devices. A *client* is the user-facing application: an installed web app on a phone, or the same screens inside a desktop shell. A *node* is a long-running process the user controls, on a desktop computer or an Android device, that holds keys, stores records, talks to the network, and watches the Bitcoin chain. A client is paired with exactly one node. The phone has no Tor client and never dials the peer-to-peer network; it talks only to its paired node and stops working when that node is unreachable.

Roles.

Buyer. Browses the catalog, sends a sealed order, and pays. For moderated purchases, funds the escrow output and holds a presigned refund.

Vendor. Publishes a signed store and listings, runs a node with an onion-service mailbox, opens letters, ships, and collects payment. For moderated purchases, holds a presigned claim.

Moderator. Publishes a record of its keys, fee, payout script, and acceptance policy, and is named in a listing and in the signed escrow terms. Receives nothing unless a party opens a dispute. On a dispute, verifies a self-contained letter and reveals one secret scalar to the party it rules for. Paid a fixed fee by the dispute outcome.

Labeler. Publishes signed sets of content labels about stores and listings. Buyers choose which labelers to follow; a labeler cannot change or remove anyone's records.

Replica. A node that stores and serves catalog records it did not create, so that a store remains readable while its publisher is offline.

Seed. A node whose address ships with the software or is pasted by the user, used only to find other nodes. Seeds are untrusted for content: every record is verified against its signature.

External parties. The Bitcoin network and chain; a source of chain data for the node (its own Bitcoin node, compact block filters, or a block-explorer API); Tor relays; Lightning routing nodes; an exchange-rate source for fiat estimates; and, for physical goods, a parcel carrier.

Records and messages. A *store record* names a store key, a display name, a region, direct-payment strings, and the store's content rating. A *listing* names its store key, describes one offer, and carries the vendor's content label. A *label set* is a labeler's signed statement about stores and listings. An *order letter* is an HPKE ciphertext addressed to one vendor. *Escrow messages* are the three funding rounds and the release exchange between buyer and vendor. A *dispute letter* goes from one party to the moderator, and a *ruling* comes back.

Network assumptions. We assume the Bitcoin chain makes progress and that a transaction paying a sufficient fee rate is confirmed within a bounded number of blocks; the escrow windows in §10 are sized against this assumption. We do not assume synchronous delivery between buyer and vendor nodes.

4 Threat Model

We consider the adversaries in Table 1. For each, we state what it controls and whether defending against it is in scope. "In scope" means the design claims a property against that adversary

under stated assumptions; "characterized" means we describe what the adversary achieves without claiming to prevent it.

Adversary	Capabilities	Scope
A1 Network observer	Sees, delays, and drops traffic on some links (a local network, an ISP, a single Tor relay).	In scope
A2 Malicious replica or seed	Serves, withholds, or reorders catalog records; lies about other nodes; logs who fetches what.	In scope for integrity; availability characterized
A3 Malicious counterparty	A buyer or vendor who deviates arbitrarily from the protocol, including going silent.	In scope
A4 Malicious moderator	Rules dishonestly, rules for both sides, refuses to rule, or logs disputes.	In scope for funds; ruling quality characterized
A5 Moderator colluding with one party	The moderator and either the buyer or the vendor act together.	Characterized
A6 Compromised device	Reads files or memory on a user's node or client.	Characterized
A7 Global passive adversary	Observes most network links and the chain.	Out of scope; characterized

Table 1: Adversaries considered.

A5 is inherent to any escrow with a third-party arbiter: if the arbiter and one party agree, the arbiter can rule for that party. The protocol limits A5 to choosing the presigned outcome that favors the colluding party; it cannot pay a third party or change amounts (§12). A6 is out of scope except for the at-rest encryption described in §7, whose limits we state. A7 is out of scope for anonymity, but §13 lists what it learns.

Cryptographic assumptions. Discrete logarithm hardness in the secp256k1 group, with hash functions modeled as random oracles for the Schnorr, MuSig2, and sigma-protocol arguments; collision resistance of SHA-256; IND-CCA2 security of HPKE base mode with the X25519/HKDF-SHA256/ChaCha20-Poly1305 suite [4]; and AEAD security of ChaCha20-Poly1305 [37]. The reference implementation uses established libraries (`libsecp256k1` through the `bitcoin` and `musig2` crates, the `hpke` crate, Arti for Tor). We assume these are correct; we did not audit them.

5 Architecture

Figure 1 shows the components. A user runs a client and a node. The node holds the keys, the wallet, the encrypted vault, the verified catalog, and the escrow state; it watches the chain and talks to other nodes. The client renders the catalog, the cart, and the order screens, and holds no keys.

5.1 The client-node boundary

A node listens only on a loopback address and refuses requests whose TCP peer is not a loopback address. On first start it creates a 256-bit random pairing token, stored in a file readable only by its owner. Requests that place orders, read or write the address book, or read the buyer's order

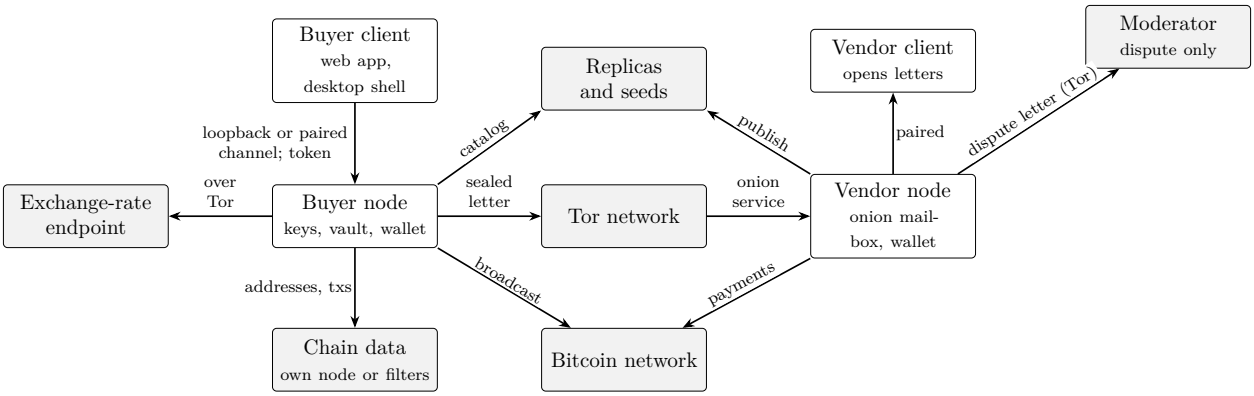


Figure 1: Components and data paths. A client talks only to its own node. Everything between nodes, including catalog fetches, sealed letters, escrow messages, and dispute letters, travels over Tor. The moderator is contacted only when a party opens a dispute.

list must carry the token. Cross-origin responses name only the client's origins, so a web page from another origin in the user's browser cannot read the token or call the node with it. A phone that pairs with a node on another machine needs an authenticated, encrypted channel to it in addition to the token; we leave that pairing transport to future work (§15.2).

This boundary protects against remote callers and against other web origins in the user's browser. It does not protect against another process running as the same user on the same machine, which can read the token file; we treat that as a compromised device (A6).

Nodes leave escrow letters for each other in a drop that takes no pairing token, since the sender is another node, not the owner's client. The drop only stores the bytes. A letter has effect only if it opens under one of the node's keys and parses as an escrow message for an escrow the node is in, and every message is then checked against the signed terms (§10).

The node refuses any client request that names a destination: a field such as `peer`, `onion`, `seed`, `replica`, or `url`, or any string value containing `://` or `.onion`. A client therefore cannot direct the node to dial an address of the client's choosing. The vendor is always the store that signed the listing the node already holds, reached at the mailbox that store published.

5.2 Deployment

A vendor runs a node that stays awake (a desktop computer or an Android device), publishes the catalog, runs an onion service that accepts sealed letters, and watches the chain. A buyer uses the same software; a buyer on an iPhone pairs the installed web app with their own desktop or Android node. All peer-to-peer traffic, including catalog fetches, publishing, label sets, letter delivery, and escrow messages, runs over Tor between nodes. A moderator is a node like any other, with a published record and an onion mailbox; it needs to be online only to answer disputes.

6 Catalog and Replication

6.1 Records

A *store record* contains a display name, a region, two direct-payment strings (a BOLT 12 offer and a BIP 352 code), a content rating for the whole store, and the store's 32-byte x-only public key S, which is the store identifier. A *listing* contains S, a title, a category, a price in satoshis, a ship-from

region, a description, a seller display name, a stock count, the date it was signed, whether it accepts a moderator, a kind (physical or digital), an optional shared product key, flat shipping in satoshis, the subtotal above which shipping is waived, and the vendor's content label. Neither record has a field for a legal name, an email address, a phone number, or a street address, and a node should refuse to sign or relay free-text fields that contain them.

6.2 Canonical encoding

Each record body is a fixed sequence of fields: a four-byte magic (`OBST` for stores, `OBLS` for listings, `OBLB` for label sets), a one-byte version, then fields in a fixed order. Integers are little-endian of fixed width. Strings are a two-byte little-endian length followed by UTF-8 bytes; blobs are a four-byte length, capped at 8 MiB. The decoder rejects a wrong magic, an unknown version, invalid UTF-8, truncation, and trailing bytes. Because every variable-length field is length-prefixed and the field order is fixed, distinct records have distinct encodings, and a decoded record re-encodes to the same bytes. Appendix A gives the layouts. A new version adds fields at the end of the body; older versions remain decodable, so records signed before a field existed keep verifying.

6.3 Signatures and addresses

The signature is a BIP 340 Schnorr signature [54] over a tagged hash of the body:

$$\sigma = \mathsf{Sign}_s\big(\mathsf{H}_{\mathrm{tag}}(\mathit{body})\big), \quad \mathsf{H}_{\mathrm{tag}}(m) = \mathrm{SHA256}\big(\mathrm{SHA256}(\mathrm{tag}) \,\|\, \mathrm{SHA256}(\mathrm{tag}) \,\|\, m\big),$$

with tags `ob/store/v1`, `ob/listing/v1`, and `ob/labels/v1`, so a signature over one record type is not valid for another. Signing uses BIP 340's deterministic nonce derivation without auxiliary randomness. The signature is appended to the body. A listing can only be signed by the secret key of the store it names, and verification checks the signature against the S inside the listing body.

A listing's address is SHA256(*body*), computed over the body without the signature. An order commits to this address, so a vendor who later edits a listing (for example, raising the price) produces a new address and cannot claim that the buyer agreed to the new terms. A node refuses an order for an address that it has not verified.

6.4 Content labels and labelers

The network carries ordinary goods, but "ordinary" differs by buyer: some do not want to see weapons, alcohol, or adult material, and some jurisdictions restrict them. The design answers this with labels and filtering on the buyer's own device, not with a central moderator of the catalog.

Self-labels. Each listing carries a content rating (*general*, *sensitive*, or *adult*) and a 16-bit set of tags (nudity, violence, drugs, weapons, gambling; the other bits are reserved). The store record carries a rating that applies to everything the store sells. Both are inside the signed body, so a replica cannot strip or soften a label without breaking the signature, and a vendor who wants a different label must sign a new record (for a listing, at a new address).

Labelers. A *labeler* is a BIP 340 key that publishes numbered *label sets*. Each entry in a set names a target (a listing address or a store key) and either gives it a rating, a tag set, and a short reason code, or retracts the labeler's earlier label on that target. For each labeler and target, the entry in the highest-numbered set that names the target is the current one, so a later set may carry only changes, and a retraction leaves no label. Entries are sorted by target with no repeats, so

one set has one encoding; a set is addressed by the hash of its body like a listing. A reason is a short code from a fixed alphabet, not free text, so a label set cannot be used to publish arbitrary statements about a vendor.

Filtering. A buyer chooses which labelers to follow; the follow list is private to the buyer's node. The client combines the store rating, the listing's self-label, the labels from followed labelers, and the buyer's own blocked words and hidden stores and listings, and decides to show, cover (show a placeholder the buyer can open), or hide each listing. When several rules apply, the strictest wins, and the client shows which source caused it. Safe mode is on by default and hides sensitive and adult listings; the buyer can relax each rule.

What labels do not do. A label changes only what one device shows. It does not remove a record from any replica, block an order, or bind other buyers. A vendor can mislabel its own goods; a labeler can label wrongly or selectively; and a buyer who follows no labeler relies only on self-labels and its own lists. The design accepts this: it gives buyers tools to filter without giving any party the power to delete. Fetching a labeler's sets reveals to whoever serves them that the node follows that labeler, so sets should be fetched through the same transport as catalog records.

6.5 What the signature does not give

A valid signature proves that the store key holder signed those bytes at some time. It does not prove that the listing is current or that a set of listings is complete. A replica can serve an older, validly signed version of a listing (an old price or stock count), and it can omit listings entirely, without either being detectable by signature checks alone. Addressing a listing by content hash makes an order unambiguous about which version it bought, but does not tell a buyer whether a newer version exists. A per-store sequence number or a signed index of current listings would address freshness and completeness (§15.2).

6.6 Replication

Every node stores the signed headers of the catalog it has seen, so that a store remains browsable while its publisher is offline, and images and files must be held by at least one replica before a listing counts as published. Because records are self-authenticating, a replica needs no trust for integrity: a client or node that receives a record from anyone checks the signature and the content hash. Replicas are trusted only for availability and see who fetches what (§13).

An open catalog invites spam. Each published header should carry a cost that receiving nodes check, such as a small bitcoin fee; the mechanism is open (§15.2). Without one, replication is limited to stores a node has chosen to follow or has received through a pasted link.

7 Identity and Keys

7.1 Key roles

Each user has one backup phrase, and the node derives a separate key for each role, so that a key used to sign the public catalog is never the key that spends coins. The phrase is a 12-word BIP 39 mnemonic [42] with BIP 32 derivation [53]:

Role	Derivation path
Spend wallet (BIP 86 Taproot [11])	`m/86'/c'/0'/{0,1}/*`
Store signing key	`m/1984'/c'/0'/0'`
Escrow key i	`m/1984'/c'/1'/i'`
Escrow mail key	`m/1984'/c'/2'/0'`
BIP 352 scan key	`m/352'/c'/0'/1'/0`
BIP 352 spend key	`m/352'/c'/0'/0'/0`

The coin type c is 0 on mainnet and 1 on test networks, and each network has its own data directory, so a phrase used on both never yields the same key on both. All role keys are pairwise distinct. Escrow keys are indexed, and the index is advanced and persisted before a key is used, so each escrow gets a new key and no key is reused across escrows. Labeler keys and moderator keys are role keys of the same kind, on their own paths, for users who act in those roles.

7.2 The mailbox key

Orders are sealed to an X25519 public key belonging to the vendor (§8). That key must be bound to the store, so that a buyer who verified a listing also knows which key to seal to. The binding is a field in the signed store record, next to the vendor's onion address. The mailbox key is derived from the phrase on its own hardened path under `m/1984'` and is never the store signing scalar reinterpreted as an X25519 key: reusing one secret across two groups and two protocols has no known attack in this combination, but there is no reason to rely on that.

7.3 Encryption at rest

The node stores the buyer's address book, the buyer's own order records, the backup phrase, the wallet state, and the labeler follow list encrypted with ChaCha20-Poly1305 under a random 256-bit key and a random 96-bit nonce per write. The buyer's order record keeps the label of the address used (for example, "Home"), not the street.

This protects against the encrypted files being copied or synced on their own, for example by a backup tool that skips key files, and keeps addresses out of logs and the catalog. If the key is stored in the same data directory, it does not protect against anyone who can read that directory; wrapping it with an operating-system keystore or a passphrase does. It is not a defense against a compromised device (A6).

8 Order Sealing and Transport

An order is a private letter to one vendor, not a catalog record and not an event on a public relay. Figure 2 shows the flow.

8.1 Order contents

The order plaintext (magic `OBOD`, version 1) contains: the address of the first listing; its quantity; an options map of up to 16 string pairs, which carries the settlement type, the shipping charge, a delivery note for digital goods, and the address and quantity of any further listings from the same seller; the ship-to text (at most 1024 bytes) for physical goods; the payment rail; and a fresh X25519 public key R that the vendor uses to reply. The order carries no long-term buyer identity key and no buyer signature. The vendor learns who the buyer is only from what the buyer writes into the letter (for a physical good, the ship-to) and from the payment.

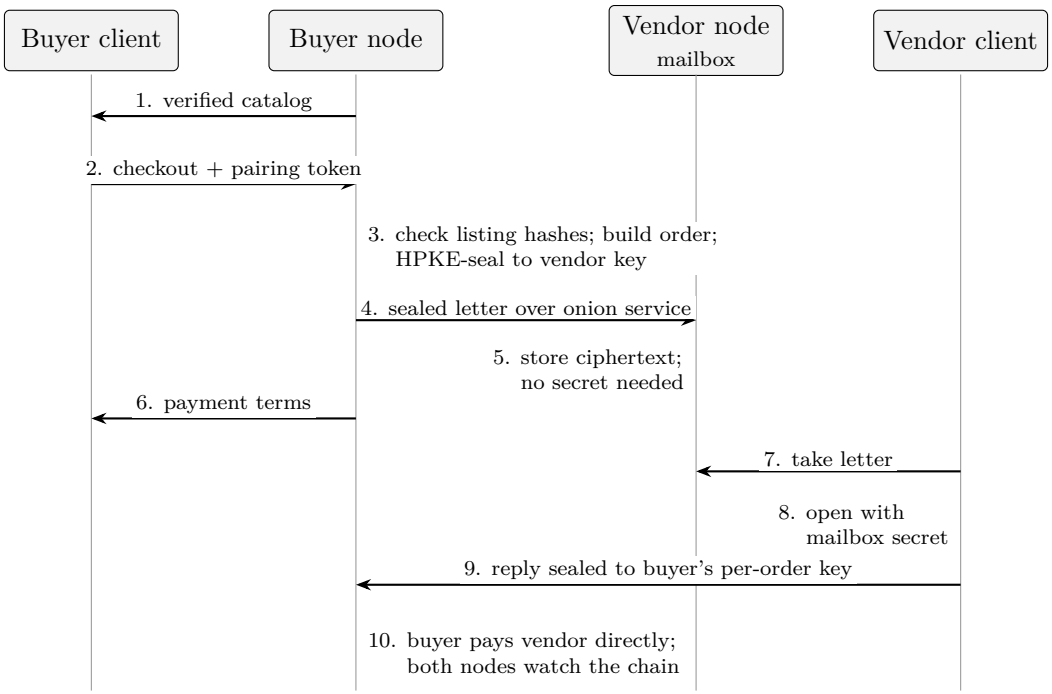


Figure 2: Order flow for one seller. Steps 4 and 9 travel over Tor to the recipient's onion mailbox. A multi-seller checkout repeats steps 3–10 independently for each seller.

8.2 Sealing

The node seals the plaintext with HPKE [4] in base mode with the suite DHKEM(X25519, HKDF-SHA256) [31, 32], HKDF-SHA256, and ChaCha20-Poly1305 [37]. The recipient key is the vendor's mailbox key P_V; the HPKE `info` string is `openbazaar-tormail-order-v1`; the associated data is empty. The sealed letter is

$$\texttt{OBSL} \,\|\, \texttt{0x01} \,\|\, enc \,\|\, \mathrm{len}_{32}(ct) \,\|\, ct,$$

where *enc* is HPKE's 32-byte encapsulated ephemeral key. Letters larger than 64 KiB are rejected. A vendor reply is sealed the same way to the buyer's per-order key R with `info` `openbazaar-tormail-reply-v1` and a size limit of 8 KiB. Because R is generated per order and travels inside the ciphertext, no relay or mailbox needs to remember which buyer wrote to which vendor in order to route a reply.

Base mode means letters are not authenticated to a sender: anyone who knows P_V can produce a letter that the vendor will decrypt. This is deliberate, since a buyer should not need a long-term identity to order, but it means the vendor must treat an order as an offer that becomes real when paid. It also means that a vendor's mailbox can be filled with junk letters; the mailbox caps letter size but does not rate-limit.

Letters are not padded. The ciphertext is exactly 16 bytes longer than the plaintext, so an observer who sees a letter at rest or in transit learns its length, which correlates with the length of the ship-to and the number of lines. Over Tor, a letter of a few hundred bytes fits in a small number of fixed-size cells, which hides the exact length from relays but not the approximate size. Padding to fixed size buckets would hide it (§15.2).

8.3 Multi-seller checkout

A checkout may contain packages from several sellers. The node validates every package (each listing exists in the verified catalog, each package is one seller and each seller one package, moderated packages only contain listings that accept the moderator) before sealing anything, then seals one letter per seller to that seller's key. A seller learns only the lines in its own package. Each package gets its own payment instructions and its own payment (§9).

8.4 The mailbox

The vendor's node accepts letters into a mailbox directory without loading any secret: a letter is written to a temporary file and renamed to a random 128-bit identifier, so the identifier is not derived from the sender. Letters are opened later with the mailbox secret. A node that is compromised while the vendor is away therefore holds ciphertext, not orders, unless the mailbox secret is also stored on it.

8.5 Transport

Letters travel over Tor [16]. The vendor's node runs a v3 onion service [51] whose address is published with its store record. The buyer's node dials it on a fixed virtual port and writes one framed letter (`OBML`, version 1); the listener accepts one stream per rendezvous, rejects streams for any other port, and writes the letter into the mailbox. If the vendor's node is unreachable, the buyer's node keeps the letter in an outbox and retries until a deadline, after which the buyer is told the order was not delivered. No third-party relay stores letters: a vendor whose node is never awake is a closed shop. Escrow letters (§10) and dispute letters use the same transport to the counterparty's or moderator's onion mailbox.

Using an onion service for the vendor's mailbox means the vendor's IP address is not revealed to buyers, and the buyer's node connects through a Tor circuit, so the vendor does not learn the buyer's IP address. Tor does not protect against an adversary who observes both ends of a circuit [16, 28]; §13 discusses this. Carrying letters over a mix network such as Nym [15, 44] to resist timing correlation is future work (§15.2).

9 Payments

All payments are in bitcoin. There is no swap service, mixer, ecash mint, or second asset in the design. Direct payment goes from the buyer to the vendor with no intermediary; moderated payment goes through the escrow of §10.

9.1 One payment per seller

A multi-seller checkout produces one order per seller, and each order is paid by its own transaction with exactly one recipient output and, if needed, one change output back to the buyer's own wallet. Two sellers are never paid in one transaction, because a transaction with outputs to two vendors would tell anyone watching the chain that the same person bought from both.

Separate transactions are necessary for unlinkability, not sufficient. If the wallet funds the second payment with the change from the first, the common-input-ownership and change heuristics [34] link the two payments. The wallet must therefore choose inputs so that the payments of one checkout do not share ancestry; when the wallet has no such coins, it should say so rather than chain one payment onto another's change.

9.2 On-chain direct payment

Each direct order receives a fresh receive address from the vendor's descriptor wallet (BIP 86 Taproot) and a BIP 21 [47] URI for the exact amount. The vendor's node watches that address and marks the order as payment seen, then paid once a configured number of confirmations is reached. The buyer can pay from its node's wallet or from any external wallet.

Chain data. Every query to a block explorer such as an Esplora server [7] names a specific address. Whoever answers learns the full set of addresses the node watches, their timing, and, without Tor, the node's IP address, which together link a user's orders to each other and to a network location. Routing such queries over Tor hides the IP address but not the address set. Running one's own Bitcoin node, or using compact block filters [40, 41], hides both, and is the recommended configuration; a node using a public explorer should warn its user.

9.3 Lightning: BOLT 12

A store record carries a BOLT 12 offer [46], and direct checkout can return an offer for the order's exact amount, answered by a Lightning node the vendor runs. An offer without a blinded path names the issuer's node key, so the offer links the store to a Lightning node; offers should use a blinded path and an issuer key distinct from the store signing key, so that the catalog identity and the Lightning identity are not trivially linked. On a payment, every routing hop sees the amount it forwards and its neighbors; the recipient's position is hidden from the payer only to the extent that a blinded path of more than one hop is used [29].

9.4 Silent payments: BIP 352

The store record also carries a BIP 352 silent-payment code [3]. A silent-payment code lets a payer derive a fresh output for each payment from a static public code, so a single published code does not lead to address reuse, and a buyer can pay without first asking the vendor for an address. The vendor's node scans the chain for outputs to its code with the watch-only scan key, which stays on the node so that scanning does not hand the key to a public indexer. Scanning is expensive without a full node, which is part of why the recommended configuration runs one.

9.5 Payjoin: BIP 77

Payjoin v2 [23] lets a receiver add an input to the buyer's transaction through an asynchronous directory reached over Oblivious HTTP [52], which breaks the assumption that all inputs belong to the payer. Payment terms can carry a `pj=` endpoint for this. When Payjoin is used, the vendor learns which coins the buyer spent, and the buyer learns one of the vendor's coins.

9.6 What the chain shows

For an on-chain direct payment, a chain observer sees an ordinary transaction with one or two outputs. The amount is the listing price times quantity plus shipping. Prices are public, so an amount that matches a distinctive price can identify the listing that was bought, even though nothing else in the transaction refers to OpenBazaar. Round prices reduce this and do not remove it. For escrow, see §10.10.

10 Scriptless Escrow

A moderated purchase protects the buyer against non-delivery and the vendor against a buyer who will not release payment, with a moderator to decide disputes. The original OpenBazaar used a 2-of-3 multisignature script for this. The redesign keeps the economic model (a moderator together with one party can decide the outcome) and changes three things: the moderator holds no key over the funds; every outcome looks like an ordinary single-key Taproot spend; and the moderator can choose only between two payouts that buyer and vendor both signed before funding.

This section specifies the two-party protocol: the parties' messages, the checks each party and the moderator run, the transactions, and the timeline.

10.1 Notation

G generates the secp256k1 group of prime order n. Lowercase letters are scalars and uppercase letters are points. H_{tag} is a BIP 340 tagged hash.

- Long-term keys: buyer $(\bar{b}, \bar{B})$, vendor $(\bar{v}, \bar{V})$. Moderator: a signing key (k_s, K_s) for rulings and a separate encryption key (k, K) for shares.
- Per-order keys (b, B) and (v, V), fresh for every order: each party derives its key from its own seed at a fresh hardened BIP 32 index (§7). Alternatively, $b = \bar{b} + \mathsf{H}_{\text{ord}}(\bar{B}, s)$ for an order secret s that travels inside the sealed order letter.
- Aggregate key: $P = \mathsf{KeyAgg}(B, V)$ under BIP 327 [36] with the two keys sorted; output key $Q = P + \mathsf{H}_{\text{TapTweak}}(P)\,G$, the BIP 341 [55] tweak with no script tree. The funding output is P2TR with output key Q and value F.
- Moderator record $\mathcal{M} = (K_s, K, S_M, \varphi, \Pi_M)$: the moderator's two keys, payout script, fee, and acceptance policy, signed and published by the moderator.
- Terms $\tau = (B, V, \mathcal{M}, F, f, h_D, h_C, h_R, S_B, S_V)$: funding value F, network fee f, dispute deadline h_D, vendor claim height h_C, refund height h_R, and payout scripts for buyer and vendor. Both parties sign τ; the signed terms are τ^*.
- Outpoint o of the funding output.
- For a dispute branch favoring party $X \in \{B, V\}$, with $\bar{X}$ the other party: the *party half* t_X, $T_X = t_X G$, derived by X from its per-order secret key and the branch; the *moderator half* u_X, $U_X = u_X G$, sampled by $\bar{X}$; the adaptor point $Y_X = T_X + U_X$.
- $\mathsf{VEnc}_K(u; ctx)$: verifiable encryption of u to K, bound to context ctx (§10.9). $\pi^{\text{pok}}(t)$: Schnorr proof of knowledge of t.

10.2 Terms and transactions

Before any funding round, the parties agree on τ. The vendor proposes it, typically with a delivery window of about two weeks (h_D around 2,016 blocks above the tip) and ruling and claim windows of 144 blocks each.

10.3 Checks on the terms

Each party checks τ twice before round 1, under its own policy and under the moderator's policy Π_M carried in τ, and again at funding against its own view of the chain tip. A party that finds any

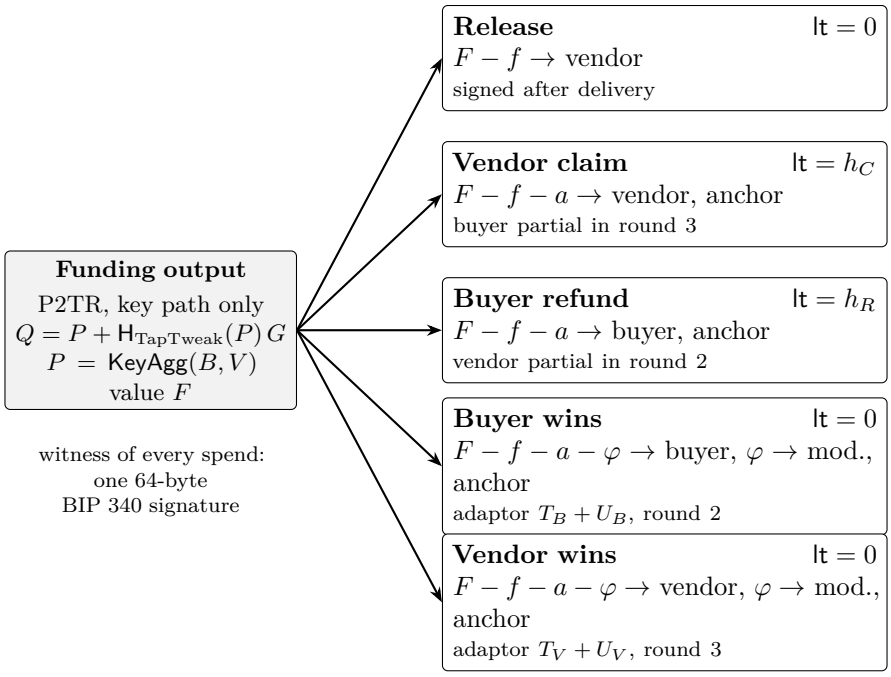


Figure 3: Escrow transaction graph. All five spends are key-path spends of the same output under the MuSig2 aggregate of the buyer's and vendor's per-order keys; the moderator's key is not part of Q. Four spends are presigned before funding; the release is signed cooperatively later. f is the network fee, φ the moderator fee, and a the 240-satoshi pay-to-anchor output, all fixed by the signed terms; lt is `nLockTime`.

check false aborts; nothing has been signed that could move funds. The attack each check stops is in §12.1.

(C1) *Moderator.* $\mathcal{M}$ equals, byte for byte, the moderator record the party already trusts, so its keys, payout script S_M, fee φ, and policy cannot be substituted by the counterparty.

(C2) *Keys.* The four keys B, V, K_s, K are distinct, and the moderator's ruling key differs from its encryption key.

(C3) *Heights in the future.* $0 < h_D < h_C < h_R$ are block heights; $h_C - h_D$ and $h_R - h_C$ are each at least 144 blocks; and, relative to the party's current tip, h_D is at least 1,008 blocks away and h_R at most 12,960 blocks away. Both bounds are checked again when funding is signed.

(C4) *Payouts.* The three payout scripts S_B, S_V, S_M are distinct, each is a standard type (P2WPKH, P2WSH, P2TR, P2PKH, or P2SH), and each output in every spend is above that type's dust limit. The moderator payout is exactly (S_M, φ) from $\mathcal{M}$.

(C5) *Fees.* f is non-zero and below a cap (50,000 satoshis), and every presigned spend meets a fee floor (4 sat/vB) on its signed size. Every spend except the release carries an anchor output (below).

(C6) *Policy.* τ satisfies Π_M: the moderator's minimum funding value and minimum windows.

(C7) *Funding.* The funding transaction spends only native segwit inputs, none of which signals replace-by-fee, so its transaction id is final before anything is presigned; it creates exactly one output with key Q and value F at o.

From τ and o, each party derives the same five spends (Figure 3). Each has one input spending

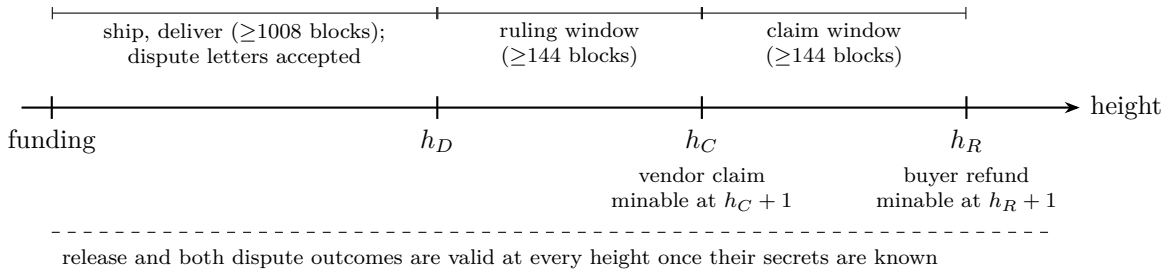


Figure 4: Escrow timeline. The default after h_C is the vendor; the refund at h_R matters only if the vendor never claims. Window sizes are policy minimums checked by each party at funding against the current tip; the minimums are 1,008 blocks from the tip to h_D and 144 blocks for each later window.

o with sequence `0xfffffffe` (so `nLockTime` is enforced) and is signed as a BIP 341 key-path spend with `SIGHASH_DEFAULT`:

Spend	`nLockTime`	**Outputs**	**When signed**
$\mathsf{tx}_{\mathrm{rel}}$ (release)	0	$F - f \to S_V$	cooperatively, after delivery
$\mathsf{tx}_{\mathrm{clm}}$ (vendor claim)	h_C	$F - f - a \to S_V$, $a \to \mathsf{A}$	presigned (buyer, round 3)
$\mathsf{tx}_{\mathrm{ref}}$ (buyer refund)	h_R	$F - f - a \to S_B$, $a \to \mathsf{A}$	presigned (vendor, round 2)
tx_{B} (buyer wins)	0	$F - f - a - \varphi \to S_B$, $\varphi \to S_M$, $a \to \mathsf{A}$	adaptor Y_B (vendor, round 2)
tx_{V} (vendor wins)	0	$F - f - a - \varphi \to S_V$, $\varphi \to S_M$, $a \to \mathsf{A}$	adaptor Y_V (buyer, round 3)

Here A is a keyless pay-to-anchor output and $a = 240$ satoshis; anyone can spend it to bump the parent's fee by child-pays-for-parent. The release is re-signed cooperatively whenever needed, so it has no anchor. All five are version-2 transactions. A transaction with `nLockTime` h can first be mined in block $h + 1$.

10.4 Funding rounds

Let N denote a party's MuSig2 public nonces for the four presigned spends and $\mathsf{ps}(\cdot)$ a MuSig2 partial signature [35] (or partial adaptor signature, for tx_{B} and tx_{V}).

Round 0. Both parties sign τ.

Round 1, buyer → vendor. The buyer prepares, but does not sign or broadcast, a funding transaction from native segwit inputs that do not signal replace-by-fee, so its transaction id is fixed before signing and cannot change afterwards. It sends $\big(o,\ \mathsf{N}_B,\ T_B,\ \pi^{\mathrm{pok}}(t_B),\ \mathsf{VEnc}_K(u_V; ctx_V)\big)$. The buyer samples u_V because the buyer loses branch V.

Round 2, vendor → buyer. The vendor checks $\pi^{\mathrm{pok}}(t_B)$ and the encryption proof for U_V (it is the favored party in branch V, so it must be sure the moderator can reveal u_V). It replies $\big(\mathsf{N}_V,\ T_V,\ \pi^{\mathrm{pok}}(t_V),\ \mathsf{VEnc}_K(u_B; ctx_B),\ \mathsf{ps}_V(\mathsf{tx}_{\mathrm{ref}}),\ \mathsf{ps}_V(\mathsf{tx}_{\mathrm{B}}; Y_B)\big)$.

Round 3, buyer → vendor. The buyer checks $\pi^{\mathrm{pok}}(t_V)$ and the encryption proof for U_B, adds its own partial signatures, and verifies that the completed refund is a valid signature under Q for the exact outpoint and value, and that the buyer-wins pre-signature is valid for Y_B. Only after these checks may the buyer sign the funding transaction, and only after it has durably stored the refund. The buyer sends $\big(\mathsf{ps}_B(\mathsf{tx}_{\mathrm{clm}}),\ \mathsf{ps}_B(\mathsf{tx}_{\mathrm{V}}; Y_V)\big)$. The vendor completes the claim, verifies the vendor-wins pre-signature, and durably stores both.

Funding. The buyer signs and broadcasts the funding transaction. The vendor checks that a transaction creates exactly the agreed output at o, and ships only after it confirms.

Each party's state is consumed when it advances and cannot be copied, and secret nonces live only inside those states, so a nonce cannot be reused for two messages. The release uses fresh nonces generated later.

Why each piece is there.

- *Refund before funding.* The buyer never signs the funding transaction until it holds a refund that verifies against the aggregate key. A vendor who aborts after round 1 costs the buyer nothing.
- *Claim before refund.* The buyer presigns a claim at $h_C < h_R$, so a buyer who received the goods and stops responding loses at $h_C + 1$.
- *Proof of knowledge on T_X.* Without it, X could choose $T_X = T' - U_X$ for a T' whose discrete log it knows, making $Y_X = T'$ and letting X complete its own winning branch without the moderator.
- *Verifiable encryption of u_X.* Without it, the losing party $\bar{X}$ could encrypt a value that does not match U_X, and the moderator would be unable to rule for X.
- *Per-order keys.* A fresh key per order gives a different Q for every order between the same two parties, so funding outputs between repeat trading partners are not linkable by address.

10.5 Release

When the buyer is satisfied, the vendor sends a fresh public nonce for $\mathsf{tx}_{\mathrm{rel}}$, and the buyer returns its nonce and partial signature. The vendor aggregates them into a 64-byte signature and broadcasts. If the vendor loses its nonce, the buyer can sign again until the release confirms; once the buyer has signed a release, its own refund and dispute paths are closed. The moderator is not involved.

10.6 Disputes

Before h_D, either party X can ask the moderator to rule for it by sending a *dispute letter*, sealed to K and carried to the moderator's mailbox:

$$L_X = \big(\tau^*,\ o,\ F,\ \mathit{script}(Q),\ Q,\ X,\ \mathsf{tx}_{\mathrm{X}}\ \text{(unsigned)},\ T_X,\ \mathsf{VEnc}_K(u_X),\ \hat{\sigma}_X,\ \mathit{memo}\big)\ \text{signed by } X.$$

Here $\hat{\sigma}_X$ is the aggregated adaptor pre-signature on tx_{X} for Y_X. The letter is self-verifying: the moderator checks that the signed terms carry exactly its own published record and satisfy its published policy, checks the funding output at o against its own chain view, recomputes P, Q, the funding script, and tx_{X} from the terms and outpoint, checks that $\hat{\sigma}_X$ is a valid pre-signature on tx_{X} for $T_X + U_X$, checks the encryption proof, and checks X's signature over the letter. It refuses a letter that arrives after h_D by its own chain view. It then durably records the branch X for $(\mathsf{H}(\tau^*), o)$ and refuses any later letter for the other branch of the same escrow. Only then does it decrypt u_X. A letter whose exact bytes it has already verified skips the proof checks but not the deadline and chain checks.

If it rules for X, the moderator returns a *ruling* $(\mathsf{H}(\tau^*), Q, o, X, Y_X, u_X)$ signed with k_s. X computes $t_X + u_X$, completes $\hat{\sigma}_X$ into a valid BIP 340 signature, and broadcasts tx_{X}, which pays X and the moderator's fee.

A ruling for X is useless to $\bar{X}$, because $\bar{X}$ does not know t_X. A moderator that signs rulings for both branches of the same terms and outpoint produces two signed rulings that together are transferable proof of equivocation. An honest moderator's ruling store prevents this by accident.

Deliberate equivocation is detectable, not preventable: both dispute outcomes are valid from height 0, so if both parties receive a ruling, the first to confirm wins.

10.7 Timeline

Figure 4 shows the heights.

- From funding until h_D, the vendor ships, the buyer receives, and either party may send a dispute letter.
- Between h_D and h_C, the moderator rules and the winner must confirm its outcome. A buyer who wins must confirm tx_{B} before $h_C + 1$, when $\mathsf{tx}_{\mathrm{clm}}$ also becomes valid.
- At $h_C + 1$, the vendor can claim. A vendor who never claims leaves the funds in the output until $h_R + 1$, when the buyer's refund becomes valid. If the vendor's claim is not confirmed by then, both are valid and the first to confirm wins.

The default outcome is therefore the vendor's. We considered and rejected a design in which the default flips to the buyer until the vendor posts a shipping attestation: without a trusted third party that observes shipping, such an attestation is not a cryptographic event, and a buyer who wants to cheat would decline to countersign it. The consequence is that a buyer who did not receive the goods must open a dispute before h_D.

10.8 Moderator fee

The moderator fee is an output of each dispute outcome and of nothing else, and it is fixed in the signed terms. It is collected only when the winner broadcasts the dispute outcome. A vendor who wins a ruling close to h_C can skip the fee by waiting for the fee-free claim; a buyer who wins cannot. This biases the moderator's income toward rulings for the buyer. §15 discusses two remedies.

10.9 Verifiable encryption of the moderator half

The moderator half u with $U = uG$ is encrypted bit by bit with exponential ElGamal under K. For each bit β_i of u ($i = 0, \ldots, 255$), the encryptor picks r_i and publishes $(A_i, C_i) = (r_i G,\ r_i K + \beta_i G)$ with a Cramer–Damgård–Schoenmakers OR proof [14] of two Chaum–Pedersen statements [9] showing that $\beta_i \in \{0, 1\}$. A Chaum–Pedersen proof then shows

$$\log_G\Big(\textstyle\sum_i 2^i A_i\Big) = \log_K\Big(\textstyle\sum_i 2^i C_i - U\Big),$$

which holds exactly when $\sum_i 2^i \beta_i \equiv u \pmod{n}$. Anyone with K and U can check the ciphertext; only the holder of k can decrypt, since $C_i - kA_i$ is either the identity or G. All challenges are tagged hashes over length-prefixed inputs and a 32-byte context that binds the proof to one escrow branch. This is the bitwise approach to verifiable encryption of a discrete logarithm [8, 50]. Its cost is size: one encrypted share is 49,761 bytes (§14).

10.10 What the chain shows

Every spend is a single-input Taproot key-path spend with one 64-byte signature. No script, no moderator key, and no multisignature structure appears on chain in any outcome, including a dispute. This is the main privacy improvement over the original 2-of-3 design. What remains visible:

- The funding amount F (price plus shipping plus fees), which can point to a listing with a distinctive price.
- The 240-satoshi pay-to-anchor output on the refund, claim, and dispute outcomes is distinctive: few ordinary wallet payments carry one, so these spends are recognizable as presigned contract spends. A cooperative release has no anchor and looks like an ordinary one-output Taproot spend.
- Dispute outcomes have three outputs, one of which is the fixed moderator fee φ. If a moderator reuses its payout script S_M, every dispute it rules on is linked on chain to it and to the others. The moderator record fixes S_M, so a moderator reuses its payout script across all escrows under one record; a moderator that wants its fees unlinked must publish new records.
- The claim and refund carry non-zero `nLockTime` values at specific heights, and all spends signal no replace-by-fee. These are weak fingerprints, since many wallets set `nLockTime` near the current height.
- Timing: a spend shortly after funding versus one after h_C.

10.11 Fees and message size

Presigned transactions have fees fixed at funding, at or above the floor of (C5). If fee rates rise, the recipient of a refund, claim, or dispute outcome bumps it through the anchor output by child-pays-for-parent. The transactions are version 2 rather than TRUC (version 3) [56], so they do not get TRUC's protection against pinning; a third party who attaches a large low-fee child to the anchor can delay a bump. Each verifiable share is about 50 KB, so a dispute letter is close to the 64 KiB letter limit. A dispute letter carries the escrow terms and the claimant's memo, not the order plaintext: the moderator learns the ship-to only if the claimant chooses to include it.

11 Discovery and Search

11.1 Search on the device

Search runs over listing headers already held by the client: a case-insensitive substring match over title, category, ship-from region, kind, description, and seller name. The query is not sent anywhere, so neither the node, a replica, nor an index learns the search terms. The cost is that the client must hold the headers it searches. We measured this filter at 0.2 ms per query over 1,000 headers and 5 ms over 10,000 headers on a laptop (§14); 100,000 headers took 49–67 ms per query. Matching is literal; there is no ranking, stemming, or index. Results then pass through the buyer's content filter (§6.4), which also runs on the device.

11.2 Finding stores and nodes

The design treats every source of nodes as untrusted for content and only partly trusted for availability:

- A user can paste a store link (a store key plus an onion address). A store reached this way opens with an empty seed list.
- Seeds shipped with the software, and peers remembered from earlier sessions, are used to find replicas.

- Every record received from any of them is verified against its signature and content hash before use.

Because records are self-authenticating, a malicious seed or replica cannot forge or alter a listing. It can withhold records, serve stale ones (§6), or, if it controls all of a node's peers, present a partial view of the network: an eclipse attack [26]. Remembering peers across sessions, drawing peers from several independent sources, and preferring a pasted link over seed-supplied data are the mitigations; no Sybil defense [18] is designed beyond the publishing cost of §6.

11.3 Private search at scale

On-device search stops scaling when the catalog no longer fits on a phone. The next step in the design is private information retrieval (PIR) [10] against a periodic snapshot of the header catalog, so that a replica answers a query without learning which records were fetched. Single-server lattice-based schemes such as SimplePIR [27] make this plausible for databases of the relevant size. The scheme is not designed in detail (§15.2). Until then, a client that fetches individual records from a replica reveals its interest in those records to that replica.

12 Security Analysis

We state each property with its assumptions. The arguments are informal; a game-based or machine-checked proof is future work. The escrow properties (P1–P4) refer to the protocol of §10.

Assumption 1 (Escrow)**.** (a) MuSig2 with BIP 327 key aggregation is unforgeable [35], and adaptor pre-signatures do not let anyone without the adaptor secret produce a valid signature. (b) The proofs of knowledge, the bit proofs, and the DLEQ proof are sound in the random-oracle model. (c) Each party runs the protocol checks before advancing and keeps its own secret key. (d) The funding transaction spends only native segwit inputs that do not signal replace-by-fee and is never replaced, so its transaction id cannot change after the presigned spends are created. (e) A party who needs to broadcast a transaction gets it confirmed within the relevant window (at least 144 blocks) at the presigned fee rate or with a fee bump through the anchor output. (f) Each party durably stores the presigned spends it holds (the buyer's refund, the vendor's claim) before the step that makes them necessary, and the moderator durably stores its rulings.

Property 1 (No custody)**.** *No party other than the buyer and the vendor holds a key sufficient to spend the escrow output.*

Argument. The output key Q is a tweak of $\mathsf{KeyAgg}(B, V)$; the moderator's key K is not aggregated. Under Assumption 1(a), producing any valid signature under Q requires partial signatures from both b and v. Every spend that can exist was either presigned by both parties before funding or is signed by both cooperatively (the release).

Property 2 (Bounded moderator)**.** *A moderator, alone or with any outside party, cannot move the funds, and cannot cause any payout other than one of the two dispute outcomes whose outputs both parties signed.*

Argument. The moderator learns u_X for at most the branch it rules on. A completed signature on tx_X needs $t_X + u_X$ and the pre-signature $\hat{\sigma}_X$. The moderator does not know t_X, and by the proof of knowledge (Assumption 1(b)) X could not have chosen T_X to cancel U_X. The outputs of tx_X, including the moderator fee and payout scripts, are fixed by the signed terms and committed

in its signature hash, so the moderator cannot redirect funds or change amounts. The terms carry exactly the moderator record each party already trusts, so a counterparty cannot substitute a different moderator, payout, or fee.

What the moderator can do: rule for the wrong party; refuse to rule, which leaves the parties to the claim and refund; or rule for both parties, which lets both broadcast and makes the outcome a race. The last is provable from two signed rulings over the same terms digest and outpoint, but not preventable by the parties; an honest moderator's ruling store refuses a second branch. A moderator colluding with one party (A5) can make that party win the dispute, which is the same economic trust as in a 2-of-3 escrow.

Property 3 (Buyer exit)**.** *If the vendor disappears after funding, the buyer recovers $F - f$ at height $h_R + 1$ without anyone's cooperation. If the vendor stays but does not deliver, the buyer recovers $F - f - \varphi$ provided it sends a dispute letter by h_D, the moderator rules for it, and the outcome confirms by h_C.*

Argument. The buyer holds a complete refund signature before it signs the funding transaction (refund before funding), and the refund needs nothing but height $h_R + 1$. If the vendor never broadcasts the claim, nothing else can spend the output before then except the release (which the buyer must sign) and the dispute outcomes (which need a ruling). For the second case, the buyer holds the pre-signature for $\mathtt{tx}_\mathrm{B}$ and the moderator can reveal u_B because the buyer verified its encryption in round 3. *Limits:* against a vendor who stays online and does not deliver, the buyer depends on the moderator ruling in time; if the moderator does not, the vendor claims at $h_C + 1$. Both parties require h_D to lie at least a delivery window above the tip at funding, so a vendor cannot propose heights that are already past.

Property 4 (Vendor protection against a stalling buyer)**.** *If the buyer stops responding after funding, the vendor collects $F - f$ at height $h_C + 1$ without anyone's cooperation.*

Argument. The vendor holds a complete claim signature from round 3 with `nLockTime` $h_C < h_R$. The buyer's only competing unilateral spend is the refund, which is not valid until $h_R + 1$, and a buyer dispute needs a ruling. *Assumptions:* the vendor kept the claim (Assumption 1(f)) and confirms it inside the claim window (Assumption 1(e)). A vendor that loses the claim lets the buyer take the refund at $h_R + 1$.

Property 5 (Catalog integrity)**.** *A party that does not hold a store's secret key cannot produce a store record or listing that verifies under that store's key, and cannot alter a listing, or its content label, without changing its address. The same holds for a labeler's sets under the labeler's key.*

Argument. BIP 340 signatures are existentially unforgeable under the discrete-log assumption in the random-oracle model. The tagged hash separates record types. The canonical encoding is injective and rejects trailing bytes, so two different records cannot share a signed body. The listing address is SHA-256 of the body, so changing any field changes the address under collision resistance. *Limits:* no freshness and no completeness (§6).

Property 6 (Letter confidentiality)**.** *Only the holder of the vendor's mailbox secret can read an order letter, and only the holder of the buyer's per-order reply secret can read a reply.*

Argument. HPKE base mode is IND-CCA2 secure for the recipient under the Gap Diffie–Hellman assumption on X25519 and AEAD security of ChaCha20-Poly1305 [4]. Distinct `info` strings separate orders, replies, and escrow messages. *Assumption:* the buyer seals to the key in the vendor's signed store record (§7); a buyer who takes the key from anywhere else gives whoever supplied it the ability to read the letter.

12.1 Attacks the escrow checks stop

Each check in §10.3 exists because a protocol without it is attackable. We list the attacks.

Past or near heights. If a party accepted any $h_D < h_C < h_R$, a vendor could propose heights at or below the current tip: the buyer's dispute window would already be closed and the vendor's claim already valid, so the vendor could take the funds immediately after funding. Requiring $h_D \geq \mathit{tip} + 1008$ at funding, checked by both parties against their own view of the chain, and bounding h_R by a maximum lock time, prevents this.

Unspendable or substituted moderator payout. If the dispute outputs paid the moderator fee to any script the terms named, a vendor could name a non-standard script or a dust amount, making both dispute outcomes non-relayable, so that no ruling could ever be broadcast and the buyer's only exit became the refund after h_R; or the vendor could name its own script as the "moderator" payout and collect the fee on a buyer win. Every payout must be a standard type above dust, the three payout scripts must differ, and the moderator's script and fee must equal those in its published record. The moderator itself refuses letters whose terms name a different payout or fee.

Replayed rulings. If a ruling only named a branch, a ruling for the buyer in one escrow could be presented as the moderator's statement in another with the same moderator. Rulings sign the terms digest, the output key, the funding outpoint, the branch, and the adaptor point, and the moderator checks on its own chain view that the outpoint exists and pays the terms' output key and amount before ruling.

Ruling both ways. A stateless moderator that receives a dispute letter from each party could reveal both u_B and u_V, turning the escrow into a broadcast race. The moderator keeps a durable ruling store keyed by terms digest and outpoint and refuses a second branch; two signed rulings over the same digest and outpoint are a transferable proof of equivocation.

Fee pinning and stale fee rates. Presigned spends pay a fee fixed at funding time. If fee rates rise, a dispute outcome or refund could be stuck below the relay floor until the window closes. Every spend except the cooperative release carries a small pay-to-anchor output that either party can spend to attach a child paying more (CPFP); the presigned fee is bounded below by a floor and above by a cap so that neither party can burn the other's funds to fees.

Funding replacement. A replaced funding transaction would have a new transaction id and strand every presigned spend, including the buyer's refund. Funding inputs that signal replace-by-fee are rejected, and the buyer must not double-spend its own funding inputs.

Secret moderator policy. If the moderator's acceptance rules (minimum amount, minimum windows, fee) were private, a moderator could refuse a dispute on grounds neither party could check beforehand. The policy is part of the moderator record inside the signed terms, and both parties check the terms against it before funding.

12.2 Other attacks considered

Rogue keys. BIP 327 key aggregation weights each key with a coefficient that depends on all keys, which prevents a party from choosing its key to cancel the other's [33]. The party halves T_X are protected by proofs of knowledge, as above.

Nonce reuse. MuSig2 secret nonces are single-use: a party's state is consumed when it advances and cannot be copied, and the release draws new nonces, distinct from those of the dispute outcome that pays the same party the same amount.

Garbage moderator share. A losing party that encrypts a value not matching U_X is caught by the favored party's check before that party signs anything leading to funding.

Late dispute letters. The moderator refuses letters received after h_D, so a ruling always has the full ruling window before the vendor claim becomes valid. The moderator's notion of "received" is its own view of the chain height, and it applies that check even to letters it has verified before.

Denial of service on the moderator. Each letter carries a 50 KB share whose verification costs about 50 ms (§14). The moderator caches letters it has verified and does not repeat the verification on decryption, but an attacker can still send many distinct letters; rate limits per source and batched verification (§15.2) bound the cost.

Client impersonation on the same machine. A local process that can read the pairing token can act as the client (§5). This is out of scope (A6).

13 Privacy Analysis

Table 2 lists, for each party or observer, what it learns, what it does not, and which mechanism the second column relies on. We assume the deployment of §5: nodes talk to each other only over Tor, and the node gets chain data from its own Bitcoin node or compact block filters unless stated otherwise.

Table 2: Who learns what.

Observer	Learns	Does not learn	Relies on
Seller	Items, quantities, and options in its own package; the ship-to for physical goods or the delivery note for digital goods; the payment and its amount; with Payjoin, the buyer's inputs.	The buyer's IP address; the buyer's orders with other sellers; any long-term buyer identity (none is sent).	Tor delivery; one letter and one payment per seller; coin selection without shared ancestry.
Carrier	Ship-to, parcel, and delivery timing.	Price paid, payment method, other orders.	Nothing: this is the ceiling for physical goods.
Moderator, no dispute	That it is named in public listings.	That a given order used it; its key is not on chain.	Key-path escrow; the moderator is contacted only on a dispute.

Observer	Learns	Does not learn	Relies on
Moderator, dispute	Signed terms (per-order keys, amounts, heights, payout scripts), the outpoint and therefore the on-chain funding and spends, and whatever the claimant puts in its memo.	The order contents and ship-to, unless the claimant includes them; other orders of either party.	Dispute letters carry no order plaintext; fresh per-order keys and payout scripts.
Labeler	That some node fetched its sets, and when.	Which buyers follow it, if sets are fetched over Tor; what they bought.	Tor; the private follow list.
Replica	Which records a node fetches or publishes, and when.	The node's IP; orders; search terms (search is on the device).	Tor; PIR for record fetches (future).
Seed	That a node speaks the protocol, when, and which peers it asks for.	The node's IP; records fetched elsewhere; orders.	Tor.
Chain observer	Amounts and timing of every payment; for escrow, a single-key-looking Taproot output and spend; a dispute outcome with a fixed moderator fee; a pay-to-anchor output marking refund, claim, and dispute spends as contract spends; links through shared inputs or change. Prices are public, so a distinctive amount can identify a listing.	Moderator keys, escrow scripts, that a dispute occurred (beyond output pattern), ship-to.	Key-path escrow; Payjoin, where used.
Block explorer, if used	Every address the node watches, with timing; without Tor, the node's IP. This links all of a user's orders together.	Order contents.	Not using one: own node or compact block filters.
Tor relays	Guard: the node's IP and that it uses Tor. Rendezvous and middle relays: neither endpoint. Approximate message sizes.	Letter contents; the vendor's IP; with one relay, who talks to whom.	Onion services on both ends.
Lightning hops	Forwarded amounts, timing, and adjacent channels. Without a blinded path, the payer learns the recipient's node key.	Order contents.	Blinded paths.
Exchange-rate source	That a node asks for rates, and when.	What was bought; the node's IP, if asked over Tor.	Tor.
Local network or ISP	That the user runs Tor.	Client–node traffic, which stays on the device or inside the pairing channel.	Tor; loopback client boundary.
Global passive adversary	Can correlate a buyer node's Tor circuit timing with a vendor onion service, and both with chain timing, linking buyer to vendor.	Letter contents.	Out of scope; a mixnet for letters (future).
Same-machine process	The pairing token, then everything the client can see.	—	Out of scope (A6).

The ship-to ceiling. For physical goods, the seller and the carrier learn where the buyer receives parcels. No protocol mechanism removes this, and we do not claim to. What the design does is keep the ship-to out of every other place: it is not in the catalog, not in any public record, not in the buyer's own order history on disk (which stores a label), not in a dispute letter unless the buyer puts it there, and, because delivery runs over Tor, not exposed to network observers.

Linking a buyer's orders. Separate letters and separate transactions are necessary for unlinkability across sellers, not sufficient. Three channels can still link one buyer's orders: a block explorer, if the node uses one, which sees all watched addresses together; the chain, if payments share inputs or change; and timing, if a buyer places several orders at once and an observer sees them confirm together. The first is removed by using one's own chain data, the second by coin selection (§9), and the third only partly, by spacing payments.

14 Evaluation

We measured the operations whose cost determines whether the design is practical on ordinary hardware, using the reference implementation. All numbers are from one machine (Apple M2 Max laptop, 32 GB, macOS 26.7, Rust 1.94 release builds, Node.js 24.6), averaged over the stated number of repetitions after a warm-up. They are microbenchmarks of local computation; we did not measure anything over a network, Tor, or a phone. Table 3 lists them.

Operation	Result	Repetitions
Encoded listing (a paperback)	268 bytes	—
Encoded sample catalog (1 store, 38 listings)	9,940 bytes	—
Sign one listing (BIP 340)	101 μs	2,000
Verify one listing	48 μs	2,000
Verify the whole sample catalog	1.9 ms	200
Order plaintext with a four-line US ship-to	173 bytes	—
Sealed letter for that order	230 bytes	—
HPKE seal one order	115 μs	2,000
HPKE open one order	75 μs	2,000
Verifiable moderator share, size (by the serialization format)	49,761 bytes	—
Encrypt one share	44–50 ms	20, two runs
Verify one share	48–50 ms	20, two runs
Decrypt one already-verified share	5–8 ms	20, two runs
On-device search, 1,000 headers	0.2 ms/query	100
On-device search, 10,000 headers	5 ms/query	100
On-device search, 100,000 headers	49–67 ms/query	100, two runs

Table 3: Measured costs. Search headers were synthesized by repeating the sample headers with distinct titles; queries were a mix of matching and non-matching terms.

The share size follows from the format: one 33-byte point, 256 ciphertext pairs of two 33-byte points each, 256 OR-proofs of four 32-byte scalars each, and a 64-byte DLEQ proof: $33 + 256 \cdot (66 + 128) + 64 = 49{,}761$. An escrow funding exchange carries two shares (one per dispute branch), so about 100 KB, and a dispute letter carries one. Decryption tests each of the 256 bits against the identity and G; because the moderator verifies a letter once and caches the result, decryption costs a few milliseconds rather than a second full verification.

These numbers support three conclusions. Signing and verifying the catalog and sealing orders cost well under a millisecond each, so they are not a bottleneck. On-device search over tens of thousands of headers is interactive on a laptop; we have not measured it on a phone. The verifiable share dominates the escrow's cost in both bytes and time, which is why smaller proofs are the first item of future work (§15.2).

15 Limitations and Future Work

15.1 Limitations

We collect the limitations of the design, ordered roughly by how much they matter to a user.

1. **The moderator is trusted to rule, and to rule in time.** A moderator that does not rule leaves a non-delivered buyer to lose at $h_C + 1$. Equivocation is provable, not preventable.
2. **Moderator and party collusion.** A moderator colluding with one party decides the dispute for that party. This is the same economic trust as a 2-of-3 multisignature escrow, and the protocol does not reduce it; it only removes the moderator's ability to take funds for itself or redirect them. Reputation, published equivocation proofs, and moderator panels (below) are the only counterweights.
3. **Moderator fee bias.** The fee is paid only from a dispute outcome. A vendor who wins a ruling near h_C can wait for the fee-free claim, while a buyer who wins must broadcast tx_{B} and pay. In expectation the moderator is paid more for buyer wins than for vendor wins, an incentive toward ruling for the buyer. Two remedies preserve the rest of the protocol: put the moderator fee into the claim $\mathsf{tx}_{\mathrm{clm}}$ as well, at the cost of charging it on undisputed escrows that end by claim; or lock a separate fee bond at funding, paid to the moderator by any dispute outcome and refunded to its owner otherwise, at the cost of a second output and its own presigned spends. Neither is part of the protocol above.
4. **The buyer must act before h_D.** The default outcome after h_C is the vendor's, so a buyer who does not dispute non-delivery in time loses the payment.
5. **Proof size and verification cost.** Each verifiable share is about 50 KB and costs about 50 ms to verify. Funding carries two shares, and an attacker can make a moderator verify many letters. The cost is tolerable on a laptop and heavy on a phone.
6. **Presigned fees and pinning.** Presigned escrow spends have fees fixed at funding and rely on a child-pays-for-parent bump through the anchor. As version-2 transactions they lack TRUC's pinning protection, and a fee spike near h_C can let the vendor's claim race the buyer's dispute outcome.
7. **Escrow outcomes are recognizable.** The anchor output marks refund, claim, and dispute spends as contract spends, and a moderator that reuses its payout script links its disputes.
8. **No catalog freshness or completeness.** A replica can serve stale listings or omit them without detection.
9. **Amounts can identify listings.** Public prices let a chain observer guess what was bought from the payment amount.
10. **Letter sizes leak.** Letters are not padded, so their sizes leak approximate content length.

11. **Timing correlation.** Tor does not resist an adversary who can watch both a buyer's node and a vendor's onion service, or who correlates letter timing with chain timing.
12. **Labels are only advice.** Self-labels are only as honest as the vendor, and labelers only as good as the buyer's choice of them; nothing in the design removes content from the network.
13. **The device is trusted.** A process on the same machine can act as the client, and at-rest encryption is only as strong as the protection of its key.
14. **Informal analysis.** The security arguments in §12 are informal, and the protocol has had no external cryptographic review.

15.2 Future work

Smaller verifiable encryption. Bitwise ElGamal is simple and relies only on the discrete-log setting of secp256k1, but it is large. Batch verification of the 256 bit proofs would cut the moderator's verification cost substantially without changing the format. Verifiable encryption in a group with an easy discrete-log subgroup, such as the class-group or Paillier-based constructions used for two-party ECDSA and scriptless swaps [48], could reduce the share to a few kilobytes, at the cost of a second hardness assumption.

Moderator panels. Replacing the single moderator with a t-of-n panel that shares the decryption key with FROST-style key generation [30] would require collusion between a party and t moderators rather than one.

Fee neutrality. Choosing between the two fee remedies above, or finding one that charges the moderator fee exactly when a dispute was opened, is open.

Pinning-resistant presigned spends. Moving the presigned spends to TRUC transactions with an ephemeral anchor would give them the pinning protection that version-2 transactions lack.

Catalog freshness and spam. A per-store sequence number and a signed index of current listings would let a buyer detect stale and withheld records. A publishing cost that replicas check would make open replication safe against spam; its form (a small payment, a proof of a funded output, or a rate limit tied to store age) is not designed.

Private retrieval and transport. Private information retrieval for catalog records at scale; padding letters to fixed size buckets; carrying letters over a mix network; and a pairing transport that lets a phone reach its owner's node on another machine with mutual authentication.

Formal analysis. A game-based proof of the escrow properties, and a machine-checked model of the message rounds and timeline.

16 Related Work

Peer-to-peer marketplaces. The original OpenBazaar [39] used IPFS-style content addressing [5] for stores, Ricardian-style signed contracts [24] for orders, and 2-of-3 multisignature escrow with a moderator. This design keeps the signed-contract idea and replaces the rest: catalogs are

signed records rather than a node's public content, orders are sealed letters rather than messages between online peers, and escrow is a two-party key with an encrypted moderator share rather than a moderator-held key. Particl [43] also runs a decentralized marketplace and uses two-party deposits in which both sides lock funds, which removes the moderator but requires the buyer and vendor to over-collateralize. Bisq [6] and Haveno [25] are peer-to-peer exchanges for currency trades rather than goods; they use 2-of-2 or 2-of-3 deposits with arbitrators and security deposits, and their arbitration roles inform our moderator design. Nostr's NIP-15 [38] defines marketplace stalls and products as signed events on relays [22]; it shares our signed-catalog model but carries orders as encrypted direct messages and has no escrow.

Earlier anonymous markets. Measurement studies of Silk Road and its successors [12, 49] document how centralized escrow, operator custody of funds, and server seizure shaped those markets. We cite them for the custody and single-operator lessons; OpenBazaar's scope is ordinary legal goods, and its privacy mechanisms are those any retail buyer has reason to want.

Payments. Lightning [45] and BOLT 12 offers [46] give fast payments whose privacy depends on routing and blinded paths; Kappos et al. [29] measure how much Lightning payments leak in practice. Silent payments (BIP 352) [3] let a payer derive a fresh output for a static public code; payjoin (BIP 78, BIP 77) [17, 23] breaks the common-input heuristic [34]. All three appear in this design as encoding or library code only (§9).

Escrow and signatures. The two-party key uses MuSig2 [35, 36], a successor to MuSig [33], on BIP 340/341 outputs [54, 55]. Adaptor signatures [1] let a completed signature reveal a secret, which is how the moderator's share finishes one branch. Discreet Log Contracts [19] use an oracle's signature as an adaptor secret to select among pre-signed outcomes; our moderator plays a similar role but is told the dispute only if one is raised and learns its secret through verifiable encryption rather than publishing an attestation. The verifiable encryption follows Stadler's publicly verifiable secret sharing [50], built from exponential ElGamal [20], Chaum–Pedersen equality proofs [9], and CDS OR-composition [14] under Fiat–Shamir [21]. Camenisch–Shoup verifiable encryption [8] gives much smaller proofs for discrete logarithms but needs a different group setting, and JugglingSwap [48] gives a segmented variant closer to ours. A threshold moderator panel could use FROST [13, 30].

Anonymous transport and private retrieval. Tor onion services [16, 51] hide the server's location; Johnson et al. [28] quantify what a realistic adversary learns about Tor users over time. Mix networks such as Loopix [44] and Nym [15] add cover traffic and delay to resist a global passive adversary, at a latency cost an order mailbox can afford. Oblivious HTTP [52] separates who asks from what is asked for a single request. Private information retrieval [10], now practical at the scale of a catalog index with SimplePIR [27], is the natural next step for discovery (§11). The replication design must also contend with eclipse [26] and Sybil [18] attacks, for which proof-of-work stamps [2] are one partial answer.

17 Conclusion

We have described a marketplace in which the public catalog contains only what a seller chooses to publish, a buyer's shipping address is sealed to one seller and never enters the catalog, filtering of what a buyer sees happens on the buyer's own device, payments are non-custodial, and escrow

is a two-party key in which a moderator can decide a dispute without ever being able to take the funds. The escrow's safety rests on a short list of checks that each party runs before funding: the moderator record is the one it trusts, the heights lie in the future, every payout is standard and above dust, fees fit a floor and a cap and can be bumped, the funding cannot be replaced, and the refund is in hand before anything is signed. The moderator, in turn, binds each ruling to the signed terms and the funding output and never rules twice on one escrow.

The limits are as important as the mechanisms. The seller and the carrier must learn the ship-to. The chain records every payment. A network observer who can watch both ends can correlate timing. A moderator colluding with one party decides the dispute, and the moderator's fee as designed rewards buyer wins more than vendor wins. The verifiable encryption that keeps the moderator honest costs 50 KB per share. What the design changes is who else learns these things, and that no party can hold the buyer's or seller's funds alone. The most important next steps are a fee-neutral moderator payment, smaller verifiable encryption, and an external review of the escrow protocol.

References

[1] Lukas Aumayr, Oguzhan Ersoy, Andreas Erwig, Sebastian Faust, Kristina Hostáková, Matteo Maffei, Pedro Moreno-Sanchez, and Siavash Riahi. Generalized channels from limited blockchain scripts and adaptor signatures. In *Advances in Cryptology – ASIACRYPT 2021*, volume 13091 of *LNCS*, pages 635–664. Springer, 2021.

[2] Adam Back. Hashcash – a denial of service counter-measure. `http://www.hashcash.org/papers/hashcash.pdf`, 2002.

[3] Josie Baker, Ruben Somsen, and Sebastian Falbesoner. BIP 352: Silent payments. Bitcoin Improvement Proposal, 2023.

[4] Richard Barnes, Karthikeyan Bhargavan, Benjamin Lipp, and Christopher A. Wood. Hybrid public key encryption. RFC 9180, 2022.

[5] Juan Benet. IPFS – content addressed, versioned, P2P file system. arXiv:1407.3561, 2014.

[6] Bisq contributors. Bisq: A decentralized bitcoin exchange network. `https://bisq.network`, source at `https://github.com/bisq-network/bisq`.

[7] Blockstream. Esplora: Block explorer and HTTP API. `https://github.com/Blockstream/esplora`.

[8] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *LNCS*, pages 126–144. Springer, 2003.

[9] David Chaum and Torben Pryds Pedersen. Wallet databases with observers. In *Advances in Cryptology – CRYPTO '92*, volume 740 of *LNCS*, pages 89–105. Springer, 1992.

[10] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 41–50, 1995.

[11] Ava Chow. BIP 86: Key derivation for single key P2TR outputs. Bitcoin Improvement Proposal, 2021.

[12] Nicolas Christin. Traveling the Silk Road: A measurement analysis of a large anonymous online marketplace. In *Proceedings of the 22nd International Conference on World Wide Web (WWW)*, pages 213–224, 2013.

[13] Deirdre Connolly, Chelsea Komlo, Ian Goldberg, and Christopher A. Wood. The flexible round-optimized Schnorr threshold (FROST) protocol for two-round Schnorr signatures. RFC 9591, 2024.

[14] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology – CRYPTO '94*, volume 839 of *LNCS*, pages 174–187. Springer, 1994.

[15] Claudia Diaz, Harry Halpin, and Aggelos Kiayias. The Nym network: The next generation of privacy infrastructure. Nym Technologies white paper, 2021.

[16] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, pages 303–320, 2004.

[17] Nicolas Dorier. BIP 78: A simple payjoin proposal. Bitcoin Improvement Proposal, 2019.

[18] John R. Douceur. The Sybil attack. In *Peer-to-Peer Systems: First International Workshop (IPTPS 2002)*, volume 2429 of *LNCS*, pages 251–260. Springer, 2002.

[19] Thaddeus Dryja. Discreet log contracts. `https://adiabat.github.io/dlc.pdf`, 2017.

[20] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[21] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology – CRYPTO '86*, volume 263 of *LNCS*, pages 186–194. Springer, 1986.

[22] fiatjaf and Nostr contributors. NIP-01: Basic protocol flow description. Nostr Implementation Possibilities, `https://github.com/nostr-protocol/nips`.

[23] Dan Gould and Yuval Kogman. BIP 77: Async payjoin. Bitcoin Improvement Proposal, 2023.

[24] Ian Grigg. The Ricardian contract. In *Proceedings of the First IEEE International Workshop on Electronic Contracting*, pages 25–31, 2004.

[25] Haveno contributors. Haveno: A decentralized Monero exchange. `https://github.com/haveno-dex/haveno`.

[26] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on Bitcoin's peer-to-peer network. In *Proceedings of the 24th USENIX Security Symposium*, pages 129–144, 2015.

[27] Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. One server for the price of two: Simple and fast single-server private information retrieval. In *Proceedings of the 32nd USENIX Security Symposium*, 2023.

[28] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. Users get routed: Traffic correlation on Tor by realistic adversaries. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 337–348, 2013.

[29] George Kappos, Haaroon Yousaf, Ania M. Piotrowska, Sanket Kanjalkar, Sergi Delgado-Segura, Andrew Miller, and Sarah Meiklejohn. An empirical analysis of privacy in the Lightning network. In *Financial Cryptography and Data Security (FC 2021)*, volume 12674 of *LNCS*, pages 167–186. Springer, 2021.

[30] Chelsea Komlo and Ian Goldberg. FROST: Flexible round-optimized Schnorr threshold signatures. In *Selected Areas in Cryptography – SAC 2020*, volume 12804 of *LNCS*, pages 34–65. Springer, 2020.

[31] Hugo Krawczyk and Pasi Eronen. HMAC-based extract-and-expand key derivation function (HKDF). RFC 5869, 2010.

[32] Adam Langley, Mike Hamburg, and Sean Turner. Elliptic curves for security. RFC 7748, 2016.

[33] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple Schnorr multi-signatures with applications to Bitcoin. *Designs, Codes and Cryptography*, 87(9):2139–2164, 2019.

[34] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. A fistful of bitcoins: Characterizing payments among men with no names. In *Proceedings of the 2013 Internet Measurement Conference (IMC)*, pages 127–140, 2013.

[35] Jonas Nick, Tim Ruffing, and Yannick Seurin. MuSig2: Simple two-round Schnorr multi-signatures. In *Advances in Cryptology – CRYPTO 2021*, volume 12825 of *LNCS*, pages 189–221. Springer, 2021.

[36] Jonas Nick, Tim Ruffing, and Elliott Jin. BIP 327: MuSig2 for BIP340-compatible multi-signatures. Bitcoin Improvement Proposal, 2022.

[37] Yoav Nir and Adam Langley. ChaCha20 and Poly1305 for IETF protocols. RFC 8439, 2018.

[38] Nostr contributors. NIP-15: Nostr marketplace. Nostr Implementation Possibilities, `https://github.com/nostr-protocol/nips`.

[39] OpenBazaar contributors. openbazaar-go: OpenBazaar 2.0 server daemon in Go. `https://github.com/OpenBazaar/openbazaar-go`. MIT license; last commit on master 20 November 2020.

[40] Olaoluwa Osuntokun and Alex Akselrod. BIP 158: Compact block filters for light clients. Bitcoin Improvement Proposal, 2017.

[41] Olaoluwa Osuntokun, Alex Akselrod, and Jim Posen. BIP 157: Client side block filtering. Bitcoin Improvement Proposal, 2017.

[42] Marek Palatinus, Pavol Rusnak, Aaron Voisine, and Sean Bowe. BIP 39: Mnemonic code for generating deterministic keys. Bitcoin Improvement Proposal, 2013.

[43] Particl contributors. Particl marketplace. `https://particl.io`, source at `https://github.com/particl`.

[44] Ania M. Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. The Loopix anonymity system. In *Proceedings of the 26th USENIX Security Symposium*, pages 1199–1216, 2017.

[45] Joseph Poon and Thaddeus Dryja. The Bitcoin Lightning network: Scalable off-chain instant payments. `https://lightning.network/lightning-network-paper.pdf`, 2016.

[46] Rusty Russell. BOLT #12: Negotiation protocol for lightning payments. Lightning Network specifications, `https://github.com/lightning/bolts`, 2024.

[47] Nils Schneider and Matt Corallo. BIP 21: URI scheme. Bitcoin Improvement Proposal, 2012.

[48] Omer Shlomovits and Oded Leiba. JugglingSwap: Scriptless atomic cross-chain swaps. arXiv:2007.14423, 2020.

[49] Kyle Soska and Nicolas Christin. Measuring the longitudinal evolution of the online anonymous marketplace ecosystem. In *Proceedings of the 24th USENIX Security Symposium*, pages 33–48, 2015.

[50] Markus Stadler. Publicly verifiable secret sharing. In *Advances in Cryptology – EUROCRYPT '96*, volume 1070 of *LNCS*, pages 190–199. Springer, 1996.

[51] The Tor Project. Tor rendezvous specification – version 3. `https://spec.torproject.org/rend-spec/`. Accessed November 2026.

[52] Martin Thomson and Christopher A. Wood. Oblivious HTTP. RFC 9458, 2024.

[53] Pieter Wuille. BIP 32: Hierarchical deterministic wallets. Bitcoin Improvement Proposal, 2012.

[54] Pieter Wuille, Jonas Nick, and Tim Ruffing. BIP 340: Schnorr signatures for secp256k1. Bitcoin Improvement Proposal, 2020.

[55] Pieter Wuille, Jonas Nick, and Anthony Towns. BIP 341: Taproot: SegWit version 1 spending rules. Bitcoin Improvement Proposal, 2020.

[56] Gloria Zhao. BIP 431: Topology restrictions for pinning. Bitcoin Improvement Proposal, 2024.

A Wire Formats

All integers are little-endian. `str` is a 2-byte length followed by UTF-8 bytes. Every format begins with a 4-byte magic and a 1-byte version, and decoders reject unknown versions and trailing bytes.

Magic	Record	Fields after magic and version
`OBST` v2	Store	name (str), region (str), BOLT 12 offer (str), BIP 352 code (str), store key (32 bytes), store rating (u8: 0 general, 1 sensitive, 2 adult), then a 64-byte BIP 340 signature over the preceding bytes with tag `ob/store/v1`. Version 1 omits the rating. The mailbox key and onion address of §7 are added at the end in a later version.
`OBLS` v5	Listing	store key (32), title (str), category (str), price (u64), ship-from (str), description (str), seller (str), stock (u32), listed-on date (u32), moderated (u8), kind (u8), product (str), shipping (u64), free-over (u64), rating (u8), tags (u16 bit set), then a signature with tag `ob/listing/v1`. Address: SHA-256 of everything before the signature. Version 4 omits the rating and tags.
`OBLB` v1	Label set	labeler key (32), name (str, at most 64 bytes), sequence (u64), issued-on date (u32), entry count (u16, at most 4,096), then entries sorted by target with no repeats: target kind (u8: 0 listing, 1 store), target (32), flags (u8; bit 0 retracts, others zero), rating (u8), tags (u16), reason (str, `[a-z0-9-]`, at most 32 bytes). A retraction has rating 0, tags 0, and no reason. Signature with tag `ob/labels/v1`; address: SHA-256 of the body.
`OBOD` v1	Order plaintext	listing address (32), quantity (u32), option count (u16) and name/value pairs (str, str), ship-to (str), rail (u8), reply key (32).
`OBSL` v1	Sealed letter	HPKE encapsulated key (32), ciphertext length (u32), ciphertext. At most 64 KiB. Orders and escrow messages use different HPKE `info` strings, so neither opens as the other.
`OBSR` v1	Sealed reply	As `OBSL`, sealed to the order's reply key; at most 8 KiB.
`OBML` v1	Onion frame	Letter length (u32), then one `OBSL` letter.
`OBET` v2	Escrow terms	The keys, moderator record (keys, payout, fee, policy), amounts, heights, and payout scripts of §10.1.
`OBEM` v1	Escrow mail	Message kind (u8), escrow id (16), field count (u8), then fields as u32 length and bytes; sealed as an `OBSL` letter between nodes.
`OBDL` v3	Dispute letter	The fields of the dispute letter in §10, then the claimant's signature.
`OBRU` v2	Ruling	Terms digest (32), output key (32), funding outpoint, favored party (u8), adaptor point (33), moderator share (32), signature (64) by the moderator's signing key over all preceding fields.