

Skyhook: Trustless Cross-Chain Value Transfer with a Fully Scriptless Pool

Abhinav Vishnu
Strahn Cryptography

Abstract

Facilitating value transfers with a scriptless chain is a longstanding problem in cryptography. We present a cross-chain messaging and value transfer protocol that maintains game-theoretic atomicity even when one party is operating on a chain which is fully scriptless, such as shielded Zcash, which has no scripting capabilities at all and requires transactions to be virtually identical. The observation is that consensus protocols which utilise segregated witness schemes to ensure transaction ID unmalleability (such as ZIP 244 in Zcash, SegWit in Bitcoin) segregate transaction data from the transaction witness component (which is the part that is actually signed). The transaction identifier is, therefore, a virtually collision-resistant commitment to the effects intended by the transaction, computable before signing, and in many chains, even without the knowledge of the signer. A contract on another chain can therefore recompute a future transaction ID from the few fields it needs to see, such as the recipient, the value and the expiry, while every other part of the transaction arrives as an opaque digest. It learns that any transaction with this ID has the agreed effect, without learning anything about the sender and without the recipient signing anything. An optional zero-knowledge layer hides the recipient and value as well. Combined with a light client for the settlement chain on the Turing-complete chain, this yields fully trustless verification that a shielded payment occurred, with no validator consortium or multisig that can be hacked or stalled.

Contents

1	Introduction	4
1.1	Our approach	4
1.2	Technical overview	5
1.3	Contributions	5
2	Background and notation	6
2.1	Segregated witnesses: ZIP 244 and SegWit	7
2.2	Collapsing transaction-identifier structure	8
2.3	Notes, commitments, nullifiers	8
2.4	Light-client verification on R	9
3	Core insight: identifier as private pre-commit	9
3.1	Open verification	9
3.2	Two orderings of commit and effect	11
3.3	Why security is inherited	11

4	Atomic swap I: ETH→ZEC	11
4.1	Requests and intents	12
4.2	Expiry	12
4.3	Lifecycle	12
5	Atomic swap II: ZEC→ETH	13
6	Chain finality	14
6.1	Block store and oracles	14
6.2	Rollups	15
6.3	Order-level finality	15
6.4	Trust model	16
7	Lock-scripts: programmable payouts	17
7.1	Committing a script	17
7.2	Execution	17
7.3	The anonymous Entrypoint	18
8	Security model and games	19
8.1	Balance and payout games	20
8.2	Privacy and call games	21
8.3	Main theorem	21
9	Privacy	22
9.1	What the base mechanism reveals	22
9.2	Optional privacy layer	22
10	Complexity	24
11	Related work	24
12	Conclusion	25
A	Finality (G8): fork resistance	28
B	Payout unforgeability (G2) and binding (G3)	30
B.1	Binding of a toy identifier	30
B.2	Payout unforgeability	30
C	Frontrunning, replay, script integrity (G4/G5/G11)	31
D	Anonymity (G9), head indistinguishability (G10) and hiding (G15)	32
D.1	With the privacy layer	33
E	Extension: P2SC passive-recipient transfer (optional)	34
E.1	P2SC: pay-to-smart-contract	34

F	Expiry, refund, crowbar and scope (G6, G7, G12, G13)	35
F.1	Payer protection by expiry (G6)	35
F.2	Optional crowbar exit	35
F.3	Equivocation over a commitment	36
F.4	Refund (G7) and scope (G12/G13)	36
G	UC framework	36
G.1	Setting	37
G.2	Ideal functionality	37
G.3	Proof of Theorem G.2	38

1 Introduction

Cross-chain communication and seamless value transfer is pivotal in scaling the future of Web3. As shielded pools and private cryptocurrencies become more prevalent, the fundamental requirement for trustless bridges that can interface with them only increases over time. A fully scriptless shielded pool such as Zcash’s Ironwood (NU6.3) [Zca26] verifies zero-knowledge proofs of well-formed private transfers, but nothing else: it enforces no hash lock, no timelock, and no conditional release, in order to maintain a massive anonymity set with uniformity. Every transaction in the pool must look like every other transaction; any per-transaction programmability would partition the anonymity set by the program it carries. Despite the critical, longstanding need, there has been barely any major research with a fully scriptless counterparty, which breaks both classical constructions for atomic exchange at once.

Lack of script on the settlement side. HTLCs [Nol13, Her18] encode atomicity as script on each chain and link the legs by a shared preimage: whoever claims on one chain must reveal the preimage, which then unlocks the other. A shielded pool offers no script in which to encode such a condition, which makes auditing-style inclusion the only available enforcement point. This difference makes porting HTLC logic impossible, as there is nowhere to port it to.

Lack of an active counterparty for scriptless swaps. Adaptor signatures [Fou19, AEE⁺21, Gug20] replace on-chain script with interactive signing, which is crucial for ensuring atomicity without scripts: the completed signature on one chain leaks the discrete-log secret that completes the signature on the other. This makes single-sided or passive-recipient flows impossible, as both parties must co-sign and stay online. Worse, the secret is revealed in the mempool at claim time, so a colluding miner can censor the second leg after learning it.

Potential equivocation across chains. A party that pre-commits to a shielded payment may later equivocate, with equivocation being the act of presenting a different effect for the very same commitment, frontrunning an honest claim, or replaying one payment into two payouts. In extreme instances, it might commit anonymously and never reveal, or reveal selectively to split views between the settlement chain and the contract chain, because adaptor signatures have no concept of practical finality and can often be overridden by legitimate means.

1.1 Our approach

To address these challenges, we propose a segregated-witness based commitment framework for atomic cross-chain transfers with a fully scriptless pool. Our primary innovation lies in devising a strategy that turns the transaction identifier itself into a (programmable) contract:

- (i) *Identifier-as-commitment with open verification.* The segregated digest ($S.predict$) pre-computes $txid = H(tx_data)$ before signing, and the contract recomputes it from the revealed fields that determine the effect and opaque digests for everything else. An optional Groth16 [Gro16] layer moves the revealed recipient and value into a zero-knowledge proof.
- (ii) *Inclusion-gated execution.* The contract chain acts only on inclusion in a block finalized under the order’s own parameters, so equivocation reduces to hash collisions and finality is enforceable, unlike adaptor flows.

Lightning [PD16] was built on the same fact for pre-signed transactions on one chain: a transaction whose identifier is fixed before it is signed can be referenced, spent from, and reasoned about before it exists. We compose this fact cross-chain.

The construction reads as Taproot [WNT20]-like effect commitments transposed across chains with anonymity added: commit to a hash now, reveal only the executed path on claim. The analogy is structural, not an equivalence: Taproot verifies on the same chain via script with an ordinary signer, while here verification happens on the other chain via a light client and hash binding, the source has no script at all, and sender anonymity comes from the shielded pool itself.

1.2 Technical overview

Consider a requester A who wants shielded ZEC and holds an ERC-20 asset on a Turing-complete reference chain R , and a liquidity provider (LP) B holding shielded ZEC on the settlement chain S . The LP’s payment to A on S is a 1-in-2-out shielded transaction. Every field that determines *what this transaction does*—the recipient note commitment, the spent nullifier, the value balance, the memo, the anchor, the fee and the header—sits in tx_data , and its identifier $txid = H(tx_data)$ is computable before any signature or proof exists. What the transaction *does not* put in tx_data is the authorization: spend-authorization signatures, the Halo 2 proof, and randomizers live in the witness, which the identifier excludes by design (ZIP 244 [NHG21]).

This yields a three-step pattern. First, A locks its ERC-20 on R ; B proposes a payment by revealing the fields that determine its effect (recipient, value, expiry) and opaque digests for everything else, and R recomputes $txid_{\text{expected}}$ from them. Second, B signs and broadcasts the payment on S ; nothing about B ’s key or input note is revealed to R . Third, once the payment has k confirmations, anyone presents to R a proof that a transaction with identifier $txid_{\text{expected}}$ is in a block finalized under the order’s parameters. R checked at acceptance, by recomputing $txid_{\text{expected}}$ from revealed fields, that such a transaction pays A the agreed amount, and it needs nothing more. Since the identifier is collision-resistant, no different effect can be presented for the same identifier; since R only acts on finalized inclusion, there is no mempool secret to steal; and since everything R checks is data, not a signature, the recipient never needs to participate.

The same gate supports a reverse direction (pay ZEC first, claim ERC-20 later) and *lock-scripts*: the payment’s ciphertext head commits to the root of a tree of programs, and the escrow is distributed by whichever committed program runs. A head carrying a planted root is indistinguishable from ordinary ciphertext until the program is revealed, which yields an anonymous call channel as a special case.

1.3 Contributions

In the following, the contributions of this work are summarized:

- **Trustless atomic swap with a scriptless pool.** We present a fully symmetric two-direction atomic swap between shielded ZEC and ERC-20 assets: an ETH→ZEC direction (Section 4) and a relay-backed ZEC→ETH direction (Section 5) that share one escrow with claim, refund and cancel exits. Zcash’s expiry height protects the payer, so no fraud-proof exit is needed. Neither leg needs per-swap recipient signing or validator-multisig attestation. We establish game-based security G1–G14, emphasizing balance safety and inclusion-bound execution (Section 8).
- **Extension: P2SC passive-recipient transfer (optional).** We present a pay-to-smart-contract variant with zero contract-side lockup for AMM integration. It adds exactly one

trust assumption—threshold custody of backing funds—and no core result depends on it (Section E).

- **Lock-scripts and anonymous cross-chain calls.** We generalize the payout from a single transfer to a committed program: the payment’s ciphertext head commits to the root of a script tree, and deterministic, bounded execution on R distributes the escrow (Section 7). Its single-call special case is an anonymous Entrypoint that calls an arbitrary function with arbitrary calldata from a scriptless pool, in the spirit of EIP-4337 [BWT⁺21]. We establish anonymity, script-integrity and fee-pinning games (G9–G11) with explicit out-of-scope timing assumptions.
- **Per-order finality without fork choice.** We give a finality layer in which trustless oracles store every header with valid proof-of-work, rollups establish chain identity by FlyClient sampling over ZIP 221 history commitments, and each order fixes its own reference block, confirmation count and deadline (Section 6). We prove a parameterized fork-resistance theorem whose bound is explicit in the adversary’s hash share, the confirmation count, the deadline and the difficulty floor. We additionally state an ideal functionality $\mathcal{F}_{\text{bridge}}$ and UC-realization theorem (Section G).
- **Optional privacy layer.** Recipients who want to hide their address and value replace the revealed openings by a Groth16 proof whose relation contains only the identifier recomputation and one note commitment (Section 9). The base mechanism needs no proof system at all.

Roadmap. Section 2 fixes notation and the collapsing digest structure. Section 3 formalizes the pre-commit and open verification. Sections 4 and 5 give the transfer protocols. Section 6 gives the finality layer and Section 7 lock-scripts and the anonymous Entrypoint. Section 8 defines the security games and states the main theorem, and Section 9 gives the exact privacy leakage and the optional privacy layer. Proofs are in Sections A to D and F; the UC treatment is in Section G.

2 Background and notation

In this section, we start by introducing the ledger and party model and formalize the segregated-witness abstraction our protocols build on.

Conventions. λ (equivalently $\kappa = 256$ for the hash output length) is the security parameter. A function is negligible, $\text{negl}(\lambda)$, if it is eventually smaller than every inverse polynomial. PPT means probabilistic polynomial-time. For a hash family $\mathcal{H}$, $\text{Adv}_{\mathcal{H}}^{\text{CR}}(\mathcal{B})$ is the probability that $\mathcal{B}$ outputs $x \neq x'$ with $\mathcal{H}(x) = \mathcal{H}(x')$; for a κ -bit output the generic birthday bound gives $2^{-\kappa/2} = 2^{-128}$. We write Adv^X for the advantage of the best PPT adversary in experiment X when the adversary is clear from context.

Definition 2.1 (Ledgers and parties). A transfer configuration is a tuple $\Gamma := (R, S, A, B, G)$, where R is the reference (contract) ledger, S is the settlement ledger, A is the requester (who locks an asset on R and wants shielded value on S), B is the liquidity provider (who pays on S and receives the locked asset), and G is the guardian set controlling a shielded address on S (used only in the P2SC extension, Section E, and as the Entrypoint’s receiving address). Either side of an S transfer may submit it once signed; submission is permissionless but the effect is pinned by the commitment.

Adversary model. We consider a PPT adversary which may corrupt the counterparty, in which case it controls the internal state of the corrupted party and all following messages that party sends. Network scheduling is bounded by delay Δ , and the adversary controls a fraction $\alpha < 1/2$ of S 's hash power (Section A makes the mining model precise). R is assumed safe and live (its own consensus is outside our model). For any transfer, we assume that counterparties may equivocate (commit one effect, claim another, frontrun, or replay) but remain rational in the sense that they prefer a successful theft over a mutual abort; we do not consider irrational parties who destroy their own funds regardless of gain. Game prices are fixed and assets cannot be retraded inside the games, which removes optionality and speculation from the analysis. Timing, network-layer identity, and contact-channel metadata are out of scope throughout; the submission channel is assumed anonymous and self-relay is always permitted.

Definition 2.2 (Abstract segregated-witness ledger). A segregated-witness ledger is a tuple $\mathcal{L} := (tx_data, witness, H, S.predict)$ satisfying:

- (P1) *Data-only identifier.* $txid = H(tx_data)$ is computed over data only, excluding all witness data, so it is stable under witness malleability.
- (P2) *Predictability.* $S.predict$ exactly pre-computes the future $txid$ from agreed parameters before witnesses exist.
- (P3) *Avalanche.* H is collision-resistant, so any effect change mutates the identifier except with collision probability.

Bitcoin SegWit [LLW15, LW16] and Zcash ZIP 244 [NHG21] both satisfy P1–P3. Two further properties are used only in parts of this paper and are flagged where needed: *sender absence* (no identity key in tx_data), used for anonymity, and a *pseudorandom ciphertext field* inside the identifier, used for lock-scripts.

2.1 Segregated witnesses: ZIP 244 and SegWit

Both SegWit and ZIP 244 separate every transaction into data committed by the identifier and witness data that is not. In SegWit, the legacy $txid$ excludes the witness field introduced by BIP 141, so third-party malleation of signatures no longer changes the identifier; BIP 143 additionally fixes the signature-hash algorithm so that signatures commit to amounts. ZIP 244 applies the same principle to every component of a Zcash v5/v6 transaction and organizes the identifier as a tree of BLAKE2b-256 [SA15] digests with domain-separated personalization strings.

More concretely, in Zcash v6 the identifier is the digest tree $txid_digest_{v6}$, while signatures, proofs, and randomized nonces live in witness-only structures committed separately by the *authorizing data commitment*. Three consequences are load-bearing for this paper.

1. *The identifier is computable before witnesses exist.* Given note commitments, nullifiers, value commitments, memos, anchor, fee, and header fields, $S.predict$ outputs the exact future $txid$.
2. *Shielded inputs do not bind the signer in tx_data .* Binding is added afterwards via the witness zero-knowledge proof and spend-authorization signature, so tx_data forms without sender, recipient randomness, or authorizing key.
3. *The witness is malleable by design, the identifier is not.* Signature nonces and proof randomness re-randomize the witness, and the digest separation contains that malleability so tx_data immutability holds regardless of re-randomization.

Table 1: Digest classification for the 1-in-2-out Ironwood-only profile.

Subtree	Status on R	Who chooses
header_digest	Variable (branch ID, lockTime, expiry)	Requester/LP agreement
transparent_digest	Constant (zero-count)	—
sapling_digest_v6	Constant (zero-count)	—
orchard_digest_v6	Constant (zero-count)	—
ironwood compact	Variable, pins effect (cmx , nf , head)	Request + LP input
ironwood memos	Variable, opaque	Payer
ironwood noncompact	Variable, opaque (cv , rk)	LP freely
flags / valueBalance	Fixed by profile (fee const)	—
witness (proofs, sigs)	Excluded from $txid$	LP at broadcast

2.2 Collapsing transaction-identifier structure

For Zcash v6 the identifier digest [NHG21, HBHW] has the following shape, where (+) marks the Ironwood pool:

```
txid_digest_v6
|-- header_digest
    (version, branchId, lockTime, expiry, consensus params)
|-- transparent_digest
    (prevouts_digest, sequence_digest, outputs_digest)
|-- sapling_digest_v6
    (spends compact + noncompact [cv, rk], outputs, valueBalance)
|-- orchard_digest_v6
    (actions compact/memos/noncompact, flagsOrchard, valueBalanceOrchard)
|-- ironwood_digest_v6 (+)
    (actions compact/memos/noncompact, flagsIronwood, valueBalanceIronwood)
```

Table 1 classifies every subtree for our 1-in-2-out fully-shielded Ironwood-to-Ironwood profile. Because the contract knows the transfer uses no transparent, Sapling, or Orchard components, those digests collapse to precomputed constants for zero-count subtrees. With the fee fixed (constant for 1-in-2-out), *valueBalanceIronwood* is predetermined as well. On-chain computation therefore reduces to the revealed header fields and the compact fields of the paying action, with the memos and noncompact digests supplied as opaque digests. Any change in any variable digest avalanches through BLAKE2b and mutates the identifier, which is the entire basis of binding (G3).

Compact digests pin the *effect* while noncompact digests pin LP-chosen values such as cv and rk ; proofs and signatures are excluded. LPs may therefore supply noncompact digests, anchor, and fee freely: an invalid choice never becomes minable under S consensus, which the contract outsources to S validators. Failure ends in refund, never theft, because R releases nothing without finalized inclusion of exactly the committed identifier.

2.3 Notes, commitments, nullifiers

An Ironwood/Orchard-style note commitment [HGBN21, BH] is

$$cmx = H^{\text{Sinsemilla}}(pk_d, g_d, v, \rho, \psi, rcm), \quad (1)$$

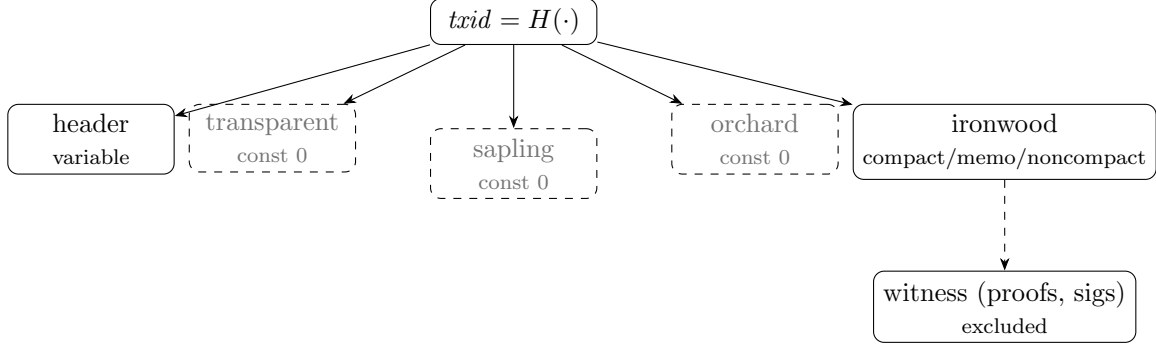


Figure 1: Collapsing digest for the Ironwood-only profile: grey dashed subtrees are precomputed zero-count constants; only header and Ironwood digests vary. The witness never enters $txid$. Complements Table 1.

with all six inputs revealed to R in the base mechanism (Section 3) and hidden by the optional privacy layer (Section 9). Here (pk_d, g_d) is the diversified recipient address, v the value, ρ the nullifier seed inherited from the spent note, ψ a sender-chosen randomizer, and rcm the commitment trapdoor. Spending a note reveals its nullifier, a pseudorandom function of (ρ, ψ, cmx) under the owner’s nullifier key nk , which is part of the full viewing key. For an output note, ρ equals the nullifier of the note spent in the same action. The anchor and fee are chosen by the LP; because they enter tx_data , grinding them grinds the identifier, so the identifier pins them automatically.

2.4 Light-client verification on R

R verifies statements about *mined* S blocks only, never mempool state:

1. inclusion of $txid$ via Merkle proof against a header that has been finalized for the order in question;
2. for the optional crowbar exit only, inclusion of a transaction carrying a given nullifier.

Section 6 describes how headers reach R and when one counts as final. Each order carries finality parameters $\Theta = (B_{\text{ref}}, k, T)$: a reference block, a confirmation count, and a deadline. We abstract verification as a predicate $\text{SPV}_k(txid, \text{proof}) \in \{0, 1\}$, with Θ left implicit, and write $\varepsilon_{\text{fork}}$ for the probability that it accepts a transaction that is not on the honest chain. Theorem A.3 bounds this term as an explicit function of Θ and the adversary’s hash share.

3 Core insight: identifier as private pre-commit

A ZIP 244 identifier is the root of a tree of hashes, and a root can be checked without seeing every leaf. A verifier that knows a few leaves (the recipient, the value, the expiry) and receives every other subtree as an opaque digest can recompute $txid$ exactly. It then knows that *any* transaction with this identifier has the revealed effect, and it has learned nothing about the subtrees it never opened.

3.1 Open verification

Definition 3.1 (Revealed leaves and opaque digests). For a transaction tx_data with identifier $txid = H(tx_data)$, an *opening* $\omega = (\Lambda, \Delta)$ consists of a set Λ of revealed leaves and a set Δ of

opaque digests, one for every maximal subtree of the ZIP 244 digest tree that contains no revealed leaf. We write $\text{Root}(\omega)$ for the root recomputed from ω . For the payment profile of Table 1, Λ contains:

- (L1) the header fields, in particular the consensus branch identifier and *expiryHeight*;
- (L2) the compact fields of the paying action: the spent nullifier nf , the output note commitment cmx , the ephemeral key, and the 52-byte ciphertext head;
- (L3) where a lock-script is bound, nothing further: its root sits in the ciphertext head of (L2);
- (L4) the opening of the output note, $(pk_d, g_d, v, \rho, \psi, rcm)$, which is not part of *tx_data* but is checked against cmx .

Every other subtree, including the noncompact digest that carries cv and rk and the digests of any change output, is opaque.

Definition 3.2 (Open check). Given an order that fixes a recipient (pk_d, g_d) , a value v , an expiry height E , and optionally a script root r , an opening ω *passes* iff:

- (C1) $cmx = \text{NoteCommit}_{rcm}(g_d, pk_d, v, \rho, \psi)$ (Equation (1)), with (pk_d, g_d, v) equal to the order's values;
- (C2) the revealed *expiryHeight* equals E ;
- (C3) if the order binds a script root r , the revealed ciphertext head contains r at an offset $o \in \{0, \dots, 20\}$ (Theorem 7.2).

On success, R stores $txid_{\text{expected}} = \text{Root}(\omega)$.

The check runs at acceptance, before the payment exists on S . It is deterministic and needs no proof system: it is one Sinsemilla commitment and a few BLAKE2b compressions over the revealed path. Its security rests on two binding properties, made precise in Section B. First, BLAKE2b collision resistance makes Root binding: a transaction whose identifier equals $txid_{\text{expected}}$ has exactly the revealed leaves. Second, Sinsemilla binding makes cmx open to exactly one note. A transaction on S with identifier $txid_{\text{expected}}$ therefore creates a note worth v to (pk_d, g_d) , expires at height E , and commits to the agreed script root, if any.

Remark 3.3 (The recipient cannot be denied the note). The opening (L4) gives the recipient everything needed to reconstruct and spend the note, whatever the ciphertext contains. A payer that writes a malformed ciphertext cannot deny the recipient its funds, because the recipient does not need to decrypt anything.

What the opening reveals. The recipient address, the value and the output note's openings become public on R , as does the identifier of the paying transaction. Nothing about the sender is revealed: the revealed spent nullifier nf is already public on S , the noncompact digest that carries cv and rk stays opaque, and the authorization lives in the witness, which the identifier excludes. The openings also do not link the output note to the transaction that later spends it. That spend reveals the note's nullifier, which is a pseudorandom function of (ρ, ψ) under the nullifier key nk ; without the recipient's full viewing key, which contains nk , the two transactions are unlinkable under the PRF's security. Recipients who want to hide their address and value as well can use the optional privacy layer of Section 9, which replaces the revealed openings by a zero-knowledge proof.

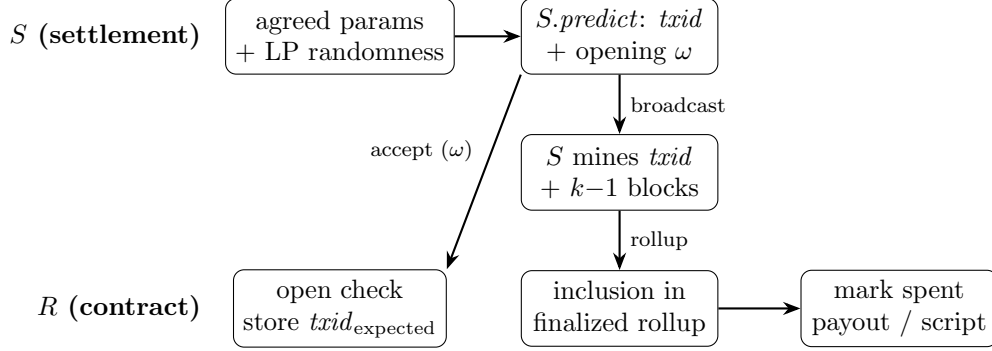


Figure 2: Primitive loop: predict $txid$ from agreed parameters, open it on R and store $txid_{\text{expected}}$, broadcast on S , then release on R once the transaction is included in a rollup finalized under the order’s parameters.

3.2 Two orderings of commit and effect

Two orderings of commitment and effect are possible.

Joint ordering. The committer binds $(txid, \text{effect})$ together at acceptance. This is the swap, where the effect is public from acceptance.

Anonymous-first ordering. The committer posts $txid$ alone and reveals the opening only on execution. Until then there is no on-chain footprint: the commitment is a bare hash, and the script root that will bind the effect sits in the ciphertext head, indistinguishable from ciphertext.

The anonymous-first ordering gives optional execution with full sender privacy, and it is the basis of the Entrypoint (and, in extension form, of P2SC).

3.3 Why security is inherited

Security is inherited, not assumed per construction. By avalanche, changing the committed effect while preserving $txid$ requires a BLAKE2b collision, and changing the note while preserving cmx requires a Sinsemilla collision (G3). The validity of the payment itself (that its input exists, is unspent and is authorized) is never checked by R : S ’s validators check it before including the transaction, and R acts only on finalized inclusion.

Recall the structural contrast with adaptor signatures: there the claim carries the secret in its witness, visible in the mempool at broadcast, so a colluding miner can censor the second leg *after* learning the secret and no confirmation depth helps. Here the claim artifact *is* the inclusion proof: R acts only on finalized inclusion, never on mempool observation, so the attack collapses to the fork race bounded by G8. Figure 2 summarizes the full loop.

4 Atomic swap I: ETH→ZEC

The requester A locks ETH or an ERC-20 asset on R and wants shielded ZEC; the LP B pays ZEC from its own shielded input and claims the escrow. The S -side payment is fully determined before it exists, so R can bind the escrow to its identifier with certainty about the effect.

Table 2: Request fields (requester) and intent fields (LP).

Field	Set by	Meaning
pk_d, g_d	Requester	Shielded address to receive ZEC
v, v_{tol}	Requester	Wanted value + slippage tolerance
$whatCanPay, howMuchPay$	Requester	Asset locked on R and amount
$\Theta = (B_{ref}, k, T)$	Requester	Finality parameters and deadline
r (optional)	Requester	Lock-script root
compact fields, header	LP	nf, cmx , ephemeral key, ciphertext head, E
(ρ, ψ, rcm)	LP	Openings of the output note
opaque digests	LP	Noncompact data, change output, other subtrees

4.1 Requests and intents

Definition 4.1 (Request and intent). A request is

$$O := (pk_d, g_d, v, v_{tol}, whatCanPay, howMuchPay, \Theta, r),$$

where $\Theta = (B_{ref}, k, T)$ are the finality parameters of Section 6.3 and r is an optional lock-script root (Section 7). An intent is an opening ω (Theorem 3.1) for a payment the proposer will make, together with the expiry height E it commits to. Selection (an open request filled by anyone, or a signed quote from a standing LP) outputs the winning pair (O, ω) ; safety depends only on the stored identifier, not on how the intent was selected.

The tolerance v_{tol} lets the requester accept any $v' \in [v - v_{tol}, v]$. Note what the LP is *not* asked to reveal: its input note, spending key, value commitment randomness or change output. These stay inside opaque digests and the witness, and R never needs them.

4.2 Expiry

The payment carries a Zcash expiry height E , enforced by S 's consensus rules: no block above height E may include it. The intent's E is checked at acceptance (C2) and pinned by $txid_{expected}$. Choosing E so that S is expected to reach height $E + k - 1$ comfortably before the deadline $t_{acc} + T$ gives the payer a clean outcome: either the payment is mined in time to be claimed, or it can never be mined at all and the payer keeps its ZEC. The gap between the expected time of height E and the deadline is a *buffer* that absorbs slow blocks. Theorem F.1 quantifies the payer's residual risk.

4.3 Lifecycle

Algorithm 1 gives the logic on R for one request.

1. *Request.* A locks the escrow and publishes O .
2. *Accept.* Anyone may propose an intent; the first whose opening passes the open check is accepted, which fixes $txid_{expected}$ and starts the deadline. Acceptance is single-use.
3. *Pay.* B signs and broadcasts the payment on S . Submission is permissionless because the effect is pinned: nobody can alter recipient, value, script root or expiry without changing the identifier.

Algorithm 1 Escrow on R for one request O .

```
1: procedure REQUEST( $O$ )
2:   lock howMuchPay of whatCanPay from  $A$ 
3: end procedure
4: procedure ACCEPT( $\omega, E$ )
5:   require no prior acceptance; open check of Theorem 3.2 passes for  $(O, \omega, E)$ 
6:    $txid_{\text{expected}} \leftarrow \text{Root}(\omega)$ ;  $t_{\text{acc}} \leftarrow \text{now}$ 
7: end procedure
8: procedure CLAIM(proof)
9:   require  $\text{SPV}_k(txid_{\text{expected}}, \text{proof}) = 1$  under  $\Theta$   $\triangleright$  finalized by  $t_{\text{acc}} + T$ 
10:  require  $\neg \text{spent}[nf]$ ;  $\text{spent}[nf] \leftarrow 1$   $\triangleright$  G5: before transfers
11:  release the escrow to the proposer, or place it under the script if  $r$  is set
12: end procedure
13: procedure REFUND
14:   require accepted and now  $> t_{\text{acc}} + T$  and no successful CLAIM
15:   return the escrow to  $A$ 
16: end procedure
17: procedure CANCEL
18:   require caller is  $A$  and no acceptance; return the escrow to  $A$ 
19: end procedure
```

4. *Claim.* Once the payment is included in the main block of a rollup that is valid and finalized under Θ , anyone may claim. R records the payment's nullifier and releases the escrow to the proposer, or, for a lock-script request, places it under the committed script (Section 7). Every check is against mined state only.

5. *Exits.* After the deadline with no claim, the escrow refunds to A (G7). Before any acceptance, A may cancel.

Why the LP cannot cheat the requester. After acceptance, any deviation in recipient, value, script root or expiry yields a different identifier, which R never pays against (G3). A transaction that S will not mine, or that the LP never broadcasts, ends in refund (G7). Spending the committed input elsewhere makes the committed payment permanently invalid, which also ends in refund; the optional crowbar exit of Section F only makes that refund faster.

Why the requester cannot cheat the LP. The escrow can leave only by claim, refund after the deadline, or cancellation before acceptance. Once the payment is finalized, anyone can claim, and the LP holds everything needed to do so. The LP's only residual risk is that the payment is mined but the claim misses the deadline, which the expiry buffer makes small (Theorem F.1).

5 Atomic swap II: ZEC $\rightarrow$ ETH

The reverse direction is the same protocol with the roles exchanged: the party that locks on R is now the LP, and the party that pays ZEC is the user. Both directions share one escrow logic (Algorithm 1), one open check and one finality rule, and no party needs the counterparty's signing key at any point. The P2SC passive-recipient variant lives separately as an optional extension (Section E).

Definition 5.1 (Advertisement and backing). A relay advertisement is $Adv := (\text{ratio}, pk_d, g_d, \text{fee})$: the rate served, the shielded address the user must pay, and the relay fee. An LP backing is an escrow on R for the advertised terms, accepted against the user’s intent exactly as in Section 4.

1. *Advertise.* A relay publishes Adv . The user checks the shielded address before paying; the identifier binds the recipient, so a wrong address voids every downstream guarantee.
2. *Back and accept.* An LP locks ETH against the terms. The user’s intent (an opening of the payment it will make, with expiry E) passes the open check, and $txid_{\text{expected}}$ is stored.
3. *Pay.* The user broadcasts the shielded payment.
4. *Claim.* Whoever holds the inclusion proof—user, LP or relay—claims once the payment is finalized under Θ ; the backing is released to the user and the relay fee is paid. Relay cost is about one L2 submission; the relay never takes custody of either asset.
5. *Exits.* After the deadline with no claim, the backing refunds to the LP. A user that never pays loses nothing but intent privacy.

Ordering risk. The user moves first on S , so it must pay only after the backing has been accepted on R with the matching $txid_{\text{expected}}$. Paying before acceptance, or to any other address, is outside the protocol and unprotected. After acceptance the user’s position is exactly the payer’s position of Section 4: expiry makes a late payment impossible, and Theorem F.1 bounds the residual risk.

Optional extension: P2SC. A passive-recipient variant with zero contract-side lockup (P2SC, pay-to-smart-contract) trades one added assumption—threshold custody of backing funds—for AMM integration with no per-order lock. It is fully specified, with its custody-separation lemma, in Section E; no core result depends on it.

6 Chain finality

An inclusion proof shows that $txid$ is in *some* block with a valid header. Before releasing funds, R must also know that the block belongs to the real S chain and will stay there. An Ethereum contract cannot observe S ’s peer-to-peer network or run its fork-choice rule, so this evidence must be submitted to it. It must also defeat an adversary that mines a private fork under exactly S ’s consensus rules, with genuine proof-of-work and well-formed commitments.

Our design has three layers. A *block store* admits every header that carries valid proof-of-work, forks included (Section 6.1). *Rollups* are views over the store that establish chain identity and burial for a chosen block (Section 6.2). *Order-level finality parameters* decide which rollups may release a particular escrow (Section 6.3). No layer runs a global fork choice: no rollup excludes another, and the store never decides which blocks are canonical. Fork resistance comes instead from a per-order reference block and deadline, and Theorem A.3 quantifies it.

6.1 Block store and oracles

Definition 6.1 (Block store). The block store BS is an append-only map from header hashes to headers. A header is admitted iff its Equihash [BK16] solution is valid and its difficulty is at least the deployment’s *difficulty floor* $D_{\min}$. A header may be admitted as an *orphan*, whose parent is unknown to R , and may later be *linked* to a stored parent, which records that it directly follows that parent.

Headers are submitted by *oracles*: any party that watches S and posts new headers. Oracles are trustless, because they report nothing that has to be believed; a header either carries valid proof-of-work or is rejected. Equihash is verified once per header at admission, and every later use references the header by hash. Because the store admits every valid header, it may contain several branches. This is intentional: the store records what work exists, not which work is canonical.

Header history is authenticated by the Merkle Mountain Range (MMR) that S already commits to in every header under ZIP 221 [LPK19]. An MMR proof shows that a header lies in the history committed by a later header, and its size is logarithmic in the history length.

The difficulty floor. S 's difficulty adjustment (DigiShield) cannot be replicated efficiently on the EVM, so R enforces only the lower bound $D_{\min}$. Participants who check S themselves can recompute the exact expected difficulty off-chain. For participants who do not, $D_{\min}$ is a trusted deployment parameter, and Theorem A.3 makes its effect on security explicit.

6.2 Rollups

Definition 6.2 (Rollup). A rollup is a tuple $\rho = (a, M, (H_1, \dots, H_{k-1}), \Sigma, \text{mode})$, where a is a previously finalized *anchor* block, M is the *main* block, $H_1, \dots, H_{k-1}$ are *head* blocks, Σ is a set of *tail samples*, and $\text{mode} \in \{\text{sampled}, \text{rooted}(\eta)\}$. ρ is *valid* iff:

- (V1) M and every H_i are in BS, and H_1 is linked to M and each H_{i+1} to H_i ;
- (V2) Σ is the set of blocks between a and M selected by FlyClient [BKLZ20] sampling, with the sampling randomness derived by Fiat–Shamir from the headers of M and its head;
- (V3) in mode **sampled**, every sample is in BS and lies under M 's ZIP 221 history commitment; in mode **rooted**(η), every sample is proven by an MMR proof against an agreed MMR root η ;
- (V4) in mode **sampled**, a was finalized by a rollup in mode **sampled**.

Sampling uses several independent FlyClient runs of about $5 \log_{10}(\text{gap})$ samples each: two runs below 100 blocks, three below 500, five below 1,000, and ten from 1,000 up. That gives about 30 samples for a 100-block gap, 150 for 1,000 and 250 for 100,000. A gap never needs more samples than it has blocks, so a short gap is verified in full. Because the samples are derived from the headers themselves, a builder cannot choose convenient ones; a forger who wants different samples must mine a different block.

Anyone may build a rollup; its builder posts no stake. Transactions *join* a rollup by proving inclusion in one of its blocks, and *finalizing* a valid rollup notifies every joined transaction. Two valid rollups may cover conflicting branches; neither invalidates the other. A swap may also build, validate and finalize its own rollup in a single transaction (*one-shot* finality).

Keeping the modes apart. A rooted rollup trades proof-of-work checks for an MMR proof against a root that its participants have checked against their own view of S . Condition (V4) prevents that root from leaking into the default guarantee: a **rooted** rollup never becomes the anchor of a **sampled** one, and R records the mode of every finalized rollup.

6.3 Order-level finality

Which rollups may release a given escrow is decided by the escrow's order, not by any global rule.

Table 3: Upper bound on the probability of release on a fork (Theorem A.3) for deadline $T = c k \tau$ and an accurate difficulty floor. The last row is the probability that an honest claim finalizes in time with no adversary present; when it fails, the escrow refunds.

$\alpha \setminus k$	$c = 1.25$				$c = 1.5$			
	4	10	20	40	4	10	20	40
0.1	4.6e−3	2.8e−6	1.4e−11	4.2e−22	7.3e−3	1.0e−5	2.3e−10	1.4e−19
0.2	5.3e−2	1.1e−3	1.7e−6	5.0e−12	7.6e−2	3.0e−3	1.6e−5	5.4e−10
0.3	0.20	2.5e−2	7.8e−4	7.9e−7	0.26	5.1e−2	3.8e−3	2.3e−5
0.4	0.49	0.20	4.3e−2	2.0e−3	0.56	0.29	0.10	1.2e−2
honest, in time	0.74	0.80	0.87	0.94	0.85	0.93	0.98	1.00

Definition 6.3 (Finality parameters). An order fixes, at acceptance time t_{acc} , parameters $\Theta = (B_{\text{ref}}, k, T)$:

- a *reference block* $B_{\text{ref}} \in \text{BS}$, ideally the tip of the escrow owner’s own view of S , or by default the most recent block finalized in mode **sampld** (a stale reference costs confirmations; see Section A);
- a *confirmation count* $k \geq 1$, counting the main block and its $k - 1$ head blocks;
- a *deadline* T .

$\text{SPV}_k(\text{txid}_{\text{expected}}, \cdot)$ accepts iff $\text{txid}_{\text{expected}}$ is included in the main block M of a valid rollup with at least $k - 1$ head blocks, M descends from B_{ref} , and that rollup is finalized by $t_{\text{acc}} + T$. Otherwise the escrow exits by refund (G7).

The reference block and the deadline close different holes. The reference block prevents *pre-mining*: an adversary cannot start a private fork carrying the committed payment before B_{ref} exists, so all of its private work on top of B_{ref} is bounded by its lead at acceptance plus what it mines within the deadline. The deadline then turns “the work exists” into “the work was done in time”: within T , an adversary with a minority share of hash produces fewer blocks than the k it needs, except with a probability that decays exponentially in k . All three parameters are chosen by the participants, so each transfer buys exactly the finality its value warrants.

Table 3 instantiates the bound with $\tau = 75\text{s}$, Zcash’s target block interval. Two readings matter. At a 10% adversary, four confirmations already push the bound below 1%. At a 30% adversary, four confirmations are not enough, but twenty confirmations within about 31 minutes ($c = 1.25$) push it below 10^{-3} . Raising k increases the work an attack requires; lowering c shortens the window in which that work must be done, at the price of more honest claims missing the deadline and refunding.

6.4 Trust model

Rollups involve no stakers, committees or votes: a rollup is valid, which anyone can check, or it is never joined and its builder has wasted gas. There are two trusted inputs, and both are explicit. The first is the difficulty floor D_{min} , for participants who do not recompute S ’s difficulty themselves; a floor below the real difficulty makes an adversary’s blocks cheaper, and Theorem A.3 charges for this through the adversary’s block rate $\lambda_{\mathcal{A}}$. The second, for rooted rollups only, is the agreed

MMR root η , which the party whose escrow is at risk must check against its own view of S before accepting the quote.

7 Lock-scripts: programmable payouts

In the swap, a successful claim pays the whole escrow to one party. A *lock-script* replaces that single transfer by a program: on a successful claim, the escrow passes to a program whose root is committed inside the payment’s ciphertext head, and the program decides how the funds are distributed. Conditional escrows, splits among several parties, time-dependent releases and calls into other contracts all reduce to deterministic programs over committed inputs. Because the commitment lives in a field the open check already reveals, lock-scripts add no trust assumption to the accept path.

7.1 Committing a script

Definition 7.1 (Script tree). A lock-script is a set of programs $P_1, \dots, P_n$. Each P_i is deterministic, declares a step budget, and may declare constants and the shape of external inputs supplied at execution. Its *root* is

$$r = \text{MerkleRoot}_{H_s}(H_s(P_1), \dots, H_s(P_n))$$

for a collision-resistant hash H_s (Keccak-256 on the EVM); a single program has root $r = H_s(P)$.

As in Taproot [WNT20], the tree commits to every branch, and execution reveals only the branch that runs.

Definition 7.2 (Head commitment). A payment *commits* to a root r if its 52-byte ciphertext head, revealed in the opening (L2), contains r at an offset $o \in \{0, \dots, 20\}$, the 21 positions at which a 32-byte root fits.

In the swap, the requester sets a lock-script flag and fixes r in its request, and acceptance additionally requires that the payment commit to r ((C3)); R records the offset. In the anonymous-first ordering, the payer chooses r itself and reveals it only at execution. Either way, the head is part of the compact digest, so $\text{txid}_{\text{expected}}$ binds the root exactly as it binds the recipient and value.

Remark 7.3 (Planted heads). A head that carries a planted root is not an encryption of the note plaintext, so the recipient’s wallet cannot find the note by trial decryption. It does not need to: the opening (L4) gives the recipient the full note. To anyone without the opening, a head with a planted root is indistinguishable from a genuine ciphertext head, because both are pseudorandom (G10).

7.2 Execution

For a lock-script request, the claim of Algorithm 1 is unchanged in every check, including the deadline, but does not pay out: it moves the escrow under the committed script and disables the refund. Execution then happens separately, at any later time.

Definition 7.4 (Execute). $\text{Execute}(P, \pi_P, \text{ext})$ succeeds iff:

- (X1) the request has been claimed, so its payment is finalized under the order’s parameters;
- (X2) r is read from the payment’s ciphertext head at the recorded offset; the head is bound by $\text{txid}_{\text{expected}}$;

Table 4: Fields of the reference Entrypoint call program, committed through its root $r = H_s(P)$.

Field	Purpose
entrypoint, chainId	Domain binding (cross-chain replay excluded)
target, calldata	Contract to call with data
token, minOut	Payout asset + slippage bound
zecValue v	Proven shielded value (may be 0 = pure signal)
relayerFee, feeRecipient	Relayer compensation, pinned
nonce, expiry	Replay window + single use
guardianCommit = $H(pk_d, g_d)$	Authorizing shielded address

(X3) π_P is a valid Merkle path from $H_s(P)$ to r ;

(X4) the request has not completed;

and P , run on its committed constants and the executor’s external inputs ext , terminates within its step budget without trapping. P starts with the whole escrow as its balance and can move value only through accounted transfers; any remainder goes to a destination fixed in advance by the script or the deployment. On success the request is marked completed. On failure, budget exhaustion or a trap, the whole execution reverts, because transactions on R are atomic, and the escrow stays under the script, to be executed again.

Three properties make execution checkable, and each is enforced by the execution environment rather than by individual programs:

Determinism. Identical inputs always produce identical outputs.

Boundedness. Every run is capped by the step budget the program declares.

Isolation. A program observes only its declared constants and external inputs, its own balance, the caller, and a single time source; it cannot end its host early, escape its sandbox, or read undeclared state.

Because the claim keeps its deadline and disables the refund, the payer protection of G6 holds unchanged: once the payment is finalized and claimed, the escrow can leave only through a committed program. A successful execution completes the request, so the program’s decision is final. A script tree in which no program can succeed leaves the escrow under the script indefinitely; avoiding this is the script author’s responsibility, exactly as for any contract. Execution is permissionless, so the external inputs are chosen by whoever executes. A program that must restrict them commits to a key in its constants and requires a signature over the inputs or the caller.

7.3 The anonymous Entrypoint

The Entrypoint is the special case of a lock-script whose program is a single call, used in the anonymous-first ordering. Its recipient is a shielded address (pk_d, g_d) on S controlled by a passive custodian that holds the spending key for backing redemption but never signs per call; this is the same address class used by the optional P2SC extension (Section E). Table 4 lists the fields of the reference call program. Any equivalent program works, but every field that can change the outcome must be inside it, or G11 fails.

The flow, Taproot-like and hash-only until reveal, proceeds as follows.

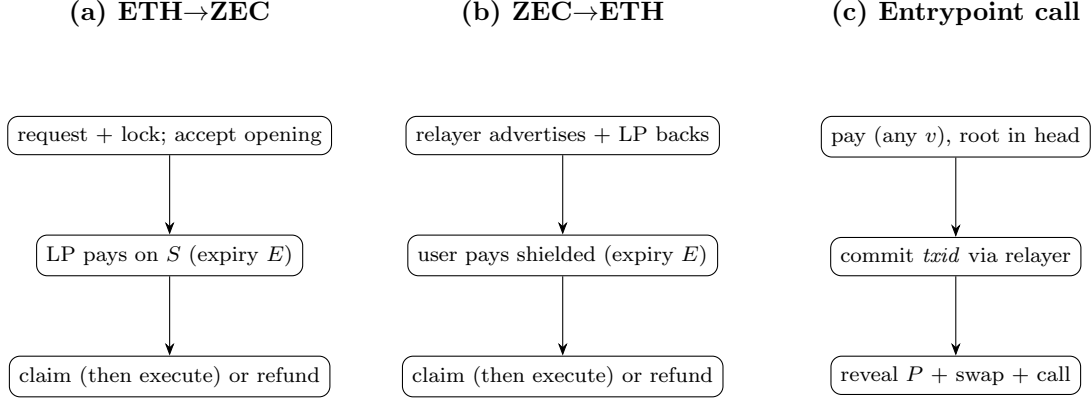


Figure 3: The core flows share one gate (open check + finalized inclusion + spent-marking): (a) and (b) the two swap directions, which differ only in who locks and who pays, and either of which may hand the claimed escrow to a lock-script; (c) the anonymous Entrypoint, a single-call lock-script committed hash-only and revealed at execution. P2SC is an extension of (b) with the R lock removed at the cost of guardian-backed custody.

1. The user pays the guardian (any v , including 0) with $r = H_s(P)$ planted in the ciphertext head. No signature is produced on S beyond the ordinary spend authorization; the head *is* the commitment.
2. The user commits $txid$ to the Entrypoint through any relaye.
3. To execute, the user hands (ω, P) to a relaye, which submits them to collect its fee. The Entrypoint checks the opening and finality, then (X2)–(X4) in the same transaction, swaps through the AMM within $minOut$, and calls the target, atomically and with spent-marking first.

Finality parameters. An anonymous-first commitment has no acceptance step at which a counterparty could choose finality parameters, so the deployment fixes them: a policy for the reference block B_{ref} , a confirmation count k and a deadline T , measured from the commitment of $txid$. Theorem A.3 then applies to every Entrypoint call with these parameters. A user who wants a fresher reference can fix one at rollup time by an MMR proof that it lies in the rollup’s history. Because that reference must itself descend from the deployment’s, it only narrows the set of branches on which a call can execute and never weakens the deployment’s guarantee.

The sender stays anonymous per G9: the recipient, the relaye and observers learn intent but not identity, and self-relay keeps even submission unlinked. Fee and call are pinned per G11. The guardian cannot decrypt a planted head, and a failed trial decryption looks exactly like a payment addressed to someone else, so the guardian cannot distinguish signaling payments from unrelated traffic before execution.

8 Security model and games

Security is game-based. Prices are fixed and there is no cross-asset trading inside games, which removes optionality and speculation; only definite fund or asset theft counts as breakage. The adversary is a malicious counterparty plus a bounded-delay network plus a hash share $\alpha < 1/2$ of

S ; R is safe and live. For the swap, including lock-script requests, security is additionally formalized in UC, with an ideal bridge functionality and a realization theorem, in Section G.

All games are stated for the base mechanism (open verification, expiry and refund). The optional privacy layer of Section 9 adds one game, G15, adds one term to the bounds of G2, G4 and G11, and adds $2\text{Adv}_{\text{Groth16}}^{\text{ZK}}$ to G9; Section 9 states the changes. Properties G1, G2, G4, G8 and G9 are *critical* and proved in full in Sections A to D. The others are lemmas, axioms on R , or scope statements, discharged in Sections C, D and F.

8.1 Balance and payout games

Game 1 (G1: Atomicity / balance safety). The challenger runs an honest party through either swap direction, as escrow owner or as payer. The adversary corrupts the counterparty, schedules messages within Δ , and mines with hash share α . The adversary *wins* if the honest party ends net-down: an honest escrow owner loses its escrow without the committed payment being on the honest chain, or an honest payer’s payment is on the honest chain while the escrow is refunded.

G1 has no independent bound; it holds by G2, G5, G6, G7 and G8 jointly (Theorem 8.1).

Game 2 (G2: Payout unforgeability). The adversary wins if R releases an escrow without a transaction on the honest chain whose identifier is the stored $\text{txid}_{\text{expected}}$ and whose output note opens to the request’s (pk_d, g_d, v) . This covers forged openings, unmined identifiers, and wrong value or recipient.

Bound: $\text{Adv}_{\text{Blake2b}}^{\text{CR}} + \text{Adv}_{\text{Sinsemilla}}^{\text{bind}} + \text{Adv}_{\text{tree}}^{\text{CR}} + \varepsilon_{\text{fork}} + \text{Adv}^{\text{replay}}$ (Theorem B.2).

Game 3 (G3: Binding). The adversary wins if it outputs two openings with the same root but different revealed leaves, or two different notes with the same cmx .

Bound: a win is a BLAKE2b collision on the digest tree or a Sinsemilla binding break (Theorem B.1).

Game 4 (G4: Frontrunning). The adversary observes an honest claim in the R mempool and wins if it steals or redirects the payout.

Bound: on the swap, zero: the payout recipient is fixed at acceptance and a copied claim pays the same party. On the anonymous path (Entrypoint; P2SC identically), redirection needs a second program under the same committed root, and replay dies on the spent set: $\text{Adv}_{H_s}^{\text{CR}} + \text{Adv}_{\text{Blake2b}}^{\text{CR}} + \text{Adv}^{\text{replay}}$ (Theorem C.2).

Axiom 1 (G5: Replay). R records the spent nullifier, and marks a lock-script request completed, on every claim or execution and before any external call (checks–effects–interactions). Hence $\text{Adv}^{\text{replay}} = 0$ by inspection; there is no game.

Game 5 (G6: Payer protection by expiry). The challenger runs an honest payer that broadcasts its payment immediately after acceptance. The adversary controls the escrow owner, schedules messages and mines with hash share α . It wins if the payment is mined on the honest chain but the escrow is refunded.

Bound: the probability that the honest chain has not reached height $E + k - 1$ in time for the claim to finalize before $t_{\text{acc}} + T$ (Theorem F.1). Above height E the payment can never be mined, so the payer keeps its ZEC.

Axiom 2 (G7: Refund). Without a claim finalized under the order’s parameters, no payout path exists, and the refund follows from correct evaluation of the deadline on the Turing-complete ledger R under bounded-delay liveness. Loss is bounded by gas and time; there is no cryptographic reduction.

Game 6 (G8: Finality). The challenger runs an honest order with finality parameters $\Theta = (B_{\text{ref}}, k, T)$ and B_{ref} on the honest chain. The adversary mines with block rate $\lambda_{\mathcal{A}}$ (blocks per target interval τ , at the difficulty floor), submits any headers it likes to the block store, and builds any rollups it likes. It wins if the escrow is released on a main block that is not on the honest chain.

Bound: $\varepsilon_{\text{fork}}(\lambda_{\mathcal{A}}, k, T) + \varepsilon_{\text{FC}}$, explicit in Theorem A.3 and tabulated in Table 3.

8.2 Privacy and call games

Game 7 (G9: Sender anonymity). The challenger flips b between two honest senders with identical intent $(v, E, \text{guardian})$. The adversary controls the guardian, the relayer and all R observers on an anonymous channel (timing out of scope) and wins if it guesses b .

Bound: $\text{Adv}^{\text{head-dist}} + \text{negl}$ under the segregation (A1), sender-absence (A2) and head-distribution (A3) axioms (Theorem D.1).

Axiom 3 (G10: Head indistinguishability). A genuine 52-byte ciphertext head and one carrying a planted root $r = H_s(\cdot)$ at a legal offset (v possibly 0) are indistinguishable random-looking strings without the viewing key; $\text{Adv}^{\text{head-dist}} \leq \text{negl}$ by the pseudorandomness of note encryption and of H_s . Zero-value signaling is therefore deniable.

Game 8 (G11: Script integrity). The adversary wins if R runs, on a committed lock-script, a program that is not a leaf of the committed tree, runs a program before the claim or after the request has completed, or refunds an escrow that has been claimed. For the Entrypoint this covers tampering with the target, calldata, token, *minOut*, fee recipient or nonce.

Bound: $\text{Adv}_{H_s}^{\text{CR}} + \text{Adv}_{\text{Blake2b}}^{\text{CR}} + \text{Adv}^{\text{replay}}$ (Theorem C.3).

G12/G13: scope. Relayer commit-bonds and rate-limits (G12) and AMM quotes and solvency (G13) are deployment policy, not protocol games. We state what the protocol does guarantee: commitments expire and prune, self-relay preserves liveness, P2SC locks nothing on R , and *minOut* fixed at S -send time bounds sandwich loss to ordinary slippage.

G14: custody (extension-only lemma). G14 applies solely to the P2SC extension (Section E): payout needs no guardian signature (G2/G4/G11), while backing redemption does (FROST assumption, cf. wrapped custody). The core swap and Entrypoint paths involve no guardian custody whatsoever.

8.3 Main theorem

Theorem 8.1 (Main). *Under collision resistance of BLAKE2b, the script hash H_s and the block Merkle tree, binding of Sinsemilla note commitments, the fork-resistance bound of Theorem A.3 for each order’s parameters, correct spent-marking and deadline evaluation on R , and—in the P2SC extension only—guardian honesty for backing redemption, the protocols of Sections 4, 5 and 7 satisfy G1–G14 with the bounds stated above.*

Table 5: Base leakage: what R observers learn vs. what stays hidden.

Public on R	Never revealed
recipient (pk_d, g_d) and value v	payer’s address, spending key, input note
output note openings (ρ, ψ, rcm)	payer’s viewing keys and change output
paying $txid, nf, cmx, E$	recipient’s viewing keys
script root r (in the head)	program P until execution; link to later spend

Proof sketch. G2–G4, G6, G8, G9 and G11 are proved individually in the appendices; G5, G7 and G10 are axioms; G12–G14 are scope statements or the custody lemma. For G1, consider the two honest roles. An honest escrow owner’s escrow leaves by exactly one of three exits of Algorithm 1. CANCEL and REFUND return it. CLAIM requires a transaction with identifier $txid_{\text{expected}}$ in the main block of a rollup finalized under Θ ; by G8 that block is on the honest chain, and by G2/G3 the transaction pays the committed note, so the owner received its leg; by G5 the exit fires at most once. For a lock-script request, the claim places the escrow under the committed script instead, and by Theorem C.4 and G11 it can then leave only through a committed program. An honest payer’s payment either is never mined, in which case it keeps its ZEC, or is mined on the honest chain at height at most E , in which case G6 bounds the probability that the claim misses the deadline. Summing the stated advantages gives the bound. $\square$

9 Privacy

9.1 What the base mechanism reveals

Table 5 states the exact leakage of the base mechanism. Request terms are public on R , exactly as in any intent-based exchange. Everything that identifies the payer stays on S , where it was already public or already hidden. The last entry of the right column needs the recipient’s nullifier key. The later spend of the output note reveals $\text{DeriveNullifier}_{nk}(\rho, \psi, cmx)$, which is pseudorandom under nk , so the openings alone do not identify it.

9.2 Optional privacy layer

Recipients who want to hide their address and value as well replace the revealed openings by a zero-knowledge proof. The request then carries a hiding commitment

$$c_O = \text{Com}(pk_d, g_d, v; r_O)$$

instead of (pk_d, g_d, v) in the clear, and the requester delivers the opening of c_O to the LP privately (for example encrypted to the LP’s key). The layer changes nothing else: the escrow, the deadline, the finality rule and every exit are as in the base mechanism.

The effect proof system. The proof system is a tuple of PPT algorithms $\Pi := (\text{Setup}, \text{Prove}, \text{Verify}, \text{Extract})$:

- $(\text{crs}, \text{td}) \leftarrow \text{Setup}(1^\lambda)$ outputs a common reference string and simulation trapdoor for Groth16 [Gro16] over BLS12-381 [WB19];
- $\pi/\perp \leftarrow \text{Prove}(\text{crs}, \vec{x}, \vec{w})$ outputs a proof or failure;

- $1/0 \leftarrow \text{Verify}(\text{crs}, \vec{x}, \pi)$ outputs 1 iff π is a valid proof for $\vec{x}$;
- $\vec{w}/\perp \leftarrow \text{Extract}(\text{crs}, \vec{x}, \pi)$, with oracle access to a prover, outputs a witness or failure.

Definition 9.1 (Properties of Π). (i) *Completeness*: for every $(\vec{x}, \vec{w}) \in \mathcal{R}_{\text{priv}}$, honest proofs verify: $\text{Verify}(\text{crs}, \vec{x}, \pi) = 1$ for $\pi \leftarrow \text{Prove}(\text{crs}, \vec{x}, \vec{w})$.

(ii) *Knowledge soundness*: for every PPT $\mathcal{A}$ there is a PPT extractor $\mathcal{E}$ with

$$\Pr \left[\text{Verify}(\text{crs}, \vec{x}, \pi) = 1 \wedge (\vec{x}, \vec{w}) \notin \mathcal{R}_{\text{priv}} : \begin{array}{l} (\vec{x}, \pi) \leftarrow \mathcal{A}(\text{crs}), \\ \vec{w} \leftarrow \mathcal{E}^{\mathcal{A}}(\text{crs}, \vec{x}, \pi) \end{array} \right] \leq \text{Adv}_{\text{Groth16}}^{\text{Sound}},$$

negligible in the algebraic group model plus random oracle model (AGM+ROM).

(iii) *Zero knowledge*: there is a simulator that, given td and $\vec{x}$ only, outputs proofs whose distribution differs from honest proofs by at most $\text{Adv}_{\text{Groth16}}^{\text{ZK}}$.

Definition 9.2 (Privacy relation). The statement and witness are

$$\vec{x} = (\text{txid}, c_O, E, r), \tag{2}$$

$$\vec{w} = (\omega, pk_d, g_d, v, \rho, \psi, rcm, r_O), \tag{3}$$

and $(\vec{x}, \vec{w}) \in \mathcal{R}_{\text{priv}}$ iff:

(Z1) $\text{Root}(\omega) = \text{txid}$;

(Z2) the cmx revealed in ω equals $\text{NoteCommit}_{rcm}(g_d, pk_d, v, \rho, \psi)$;

(Z3) $c_O = \text{Com}(pk_d, g_d, v; r_O)$;

(Z4) the *expiryHeight* in ω equals E , and, if r is set, the ciphertext head in ω contains r at a legal offset.

The relation is exactly the open check with the openings moved into the witness. It contains no spend-validity condition: whether the payment's input exists, is unspent and is authorized is checked by S 's validators before inclusion, and R still acts only on finalized inclusion. The circuit therefore consists of the BLAKE2b compressions on the revealed path and one Sinsemilla commitment. Concrete Sinsemilla arithmetization over BLS12-381 has no published baseline, so we state no constraint counts or prover timings.

Game 9 (G15: Recipient and value hiding, privacy layer only). The adversary chooses two requests that differ only in (pk_d, g_d, v) ; the challenger runs the request with the privacy layer for one of them, chosen by a bit b . The adversary sees everything on R and the block store, and wins if it guesses b .

Bound: $2\text{Adv}_{\text{Groth16}}^{\text{ZK}} + \text{Adv}_{\text{Com}}^{\text{hide}}$ plus the hiding of note commitments and note encryption (Theorem D.3). The escrowed amount on R remains public, so v is hidden only up to the exchange rate.

Effect on the other games. With the layer in use, G2, G4 and G11 each gain the term $\text{Adv}_{\text{Groth16}}^{\text{Sound}}$, because a verifying proof for a false statement replaces the forged opening (Theorem D.2). G9 gains $2\text{Adv}_{\text{Groth16}}^{\text{ZK}}$. All other bounds are unchanged.

10 Complexity

We state costs asymptotically.

Acceptance. The open check computes one Sinsemilla note commitment and the BLAKE2b compressions on the path from the revealed leaves to the root, a constant number for the payment profile. Every empty-subtree digest is a precomputed constant, so collapsing removes their hashing entirely. On the EVM, BLAKE2b with ZIP 244 personalization is evaluated through the BLAKE2 compression-function precompile of EIP-152 [HLDH16].

Header admission. Admitting a header costs one Equihash verification, whose cost depends on the size of the solution and not on the mining effort. It is paid once per header, by whichever oracle submits it, and never again by the rollups that reference it.

Rollups. A rollup references its main block, $k - 1$ head blocks and $O(\log g)$ tail samples for a gap of g blocks since its anchor. With every sample already in the store, each reference is a storage read, so the cost of a rollup is logarithmic in the gap and linear in k . A sample that is not stored costs one header admission. In mode *rooted*, each sample is instead an MMR proof of size $O(\log g)$ against the agreed root, so the rollup needs no stored blocks at all.

Execution. A lock-script execution verifies one Merkle path of length $O(\log n)$ in a tree of n programs and then runs the program, whose cost is bounded by its declared step budget.

Claims. A claim is a Merkle path from $txid_{\text{expected}}$ to the main block’s transaction root, of size logarithmic in the number of transactions in the block, plus a constant number of storage writes for spent-marking.

Privacy layer. A Groth16 proof is three group elements, and verification is a single pairing-product check whose cost is independent of circuit size. On the EVM, verification over BLS12-381 uses the precompiles of EIP-2537 [VOSS20]; on a chain without them, the same relation can be proven over BN254 at a lower security level.

Latency. End-to-end latency is about $k\tau$ plus R confirmation time, and is capped by the order’s deadline T .

11 Related work

Script-based and signature-based swaps. *HTLC swaps* (TierNolan [Nol13], formalized by Herlihy [Her18]) need the same hash on script-capable chains on both sides; a scriptless pool cannot participate, and inclusion plays no role. *Adaptor-signature swaps* [Fou19, AEE⁺21, Gug20] remove on-chain script but keep two-sided interactive signing and leak the secret in the mempool at claim, so finality cannot be enforced—Section 3 gives the full contrast, since our claim artifact is the inclusion proof itself.

Relay-based constructions. *XCLAIM* [ZHL⁺19] does use chain-relay inclusion proofs, but for over-collateralized vault-backed assets on script-capable chains, with vault collateral replacing our hash binding. *Light-client relays*—BTC Relay [Eth] as the canonical on-chain SPV construction, zkRelay [WE20] batching header checks into one SNARK, FlyClient [BKLZ20] sampling logarithmically, NIPoPoWs [KMZ20] with superblock proofs—are our verification substrate, not our contribution. We use FlyClient sampling over the history commitments that Zcash already includes in its headers [LPK19]. The difference is in how a fork is rejected: a FlyClient verifier compares competing provers and keeps the heavier chain, whereas our contract never compares chains. Each order instead fixes a fresh reference block and a deadline, and Theorem A.3 bounds the probability of release on a fork directly.

General bridges. zkBridge [XZC⁺22] with SNARK-proven header relay and Rainbow [NEA] with production light clients target script-capable counterparties and say nothing about scriptless shielded sources.

Positioning. Scoped precisely, our claim is the first trustless atomic swap and anonymous call channel from a fully scriptless shielded pool with private sender and input—not general bridging, where the cited systems remain the state of the art.

12 Conclusion

Segregated-witness identifiers are private pre-commits. Combined with open verification and per-order finality over a trustless block store, they give atomic cross-chain execution against a fully scriptless shielded pool: no per-swap recipient signing, no validator-multisig attestation, no mem-pool secret for colluding miners to steal. The same gate yields a fully symmetric atomic swap, programmable payouts through lock-scripts committed in the ciphertext head, and, as their single-call case, an anonymous general-purpose Entrypoint. An optional extension (P2SC, Section E) additionally offers passive-recipient transfer with zero contract-side lockup at the cost of one custody assumption.

Limitations. In the P2SC extension only, backing redemption remains guardian-trusted; a corrupt guardian set can rug or freeze backing funds (it cannot forge payout). AMM inventory dynamics are ordinary AMM dynamics: the pool ends long proven-credit and short payout asset, and LPs bear exactly the impermanent loss any pool bears. Finality is probabilistic: Theorem A.3 bounds the probability of release on a fork, and participants must choose k and T for the hash share they are willing to tolerate; at a 30% adversary, safe parameters take about half an hour. Participants who do not check S themselves trust the difficulty floor. Timing and contact-channel linkage are outside the anonymity model, and the recipient and value are public on R unless the privacy layer is used.

Future work. Concrete circuits for the privacy layer once Sinsemilla lookup arithmetization over BLS12-381 stabilizes, and measurements of S 's hash distribution to calibrate the finality parameters of Table 3.

Use of AI Tools

AI assistants were used to rapidly fuzz and debug software deployments, to check formatting and build issues, and to refine the language of the exposition. All protocol designs, security arguments, and claims were conceived, verified, and approved by the authors, who bear full responsibility for the correctness and originality of this work.

References

- [AEE⁺21] Lukas Aumayr, Oguzhan Ersoy, Andreas Erwig, Sebastian Faust, Kristina Hostakova, Matteo Maffei, Pedro Moreno-Sanchez, and Siavash Riahi. Generalized channels from limited blockchain scripts and adaptor signatures. In *Proc. ASIACRYPT 2021*, 2021.
- [AZR20] Hayden Adams, Noah Zinsmeister, and Dan Robinson. Uniswap v2 core, 2020.
- [BH] Sean Bowe and Daira-Emma Hopwood. *Sinsemilla Hash Function (Zcash Protocol Specification, Concretesinsemillahash)*. No standalone paper; Pedersen-like DLP-based PLONKish-lookup-efficient hash on Pallas.
- [BK16] Alex Biryukov and Dmitry Khovratovich. Equihash: Asymmetric proof-of-work based on the generalized birthday problem. In *Proc. Network and Distributed System Security Symposium (NDSS)*, 2016.
- [BKLZ20] Benedikt Bunz, Lucianna Kiffer, Loi Luu, and Mahdi Zamani. Flyclient: Super-light clients for cryptocurrencies. In *Proc. IEEE Symposium on Security and Privacy (S&P)*, pages 928–946, 2020.
- [BWT⁺21] Vitalik Buterin, Yoav Weiss, Dror Tirosh, Shahaf Nacson, Alex Forshtat, Kristof Gazso, and Tjaden Hess. ERC-4337: Account abstraction using alt mempool, 2021. Status: Final.
- [CKGW24] Deirdre Connolly, Chelsea Komlo, Ian Goldberg, and Christopher A. Wood. Flexible round-optimized schnorr threshold protocol (FROST), RFC 9591, 2024. IRTF CFRG.
- [Eth] Ethereum / ConsenSys contributors. BTC relay: Ethereum contract for bitcoin SPV. Live code and docs, 2015–2016, archived. No paper.
- [Fou19] Lloyd Fournier. One-time verifiably encrypted signatures a.k.a. adaptor signatures, 2019. Unpublished manuscript. Not a thesis.
- [GKL15] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Proc. EUROCRYPT 2015, LNCS 9057*, pages 281–310. Springer, 2015.
- [GKL17] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol with chains of variable difficulty. In *Proc. CRYPTO 2017, LNCS 10401*, pages 291–323. Springer, 2017.
- [Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In *Proc. EUROCRYPT 2016 (2), LNCS 9666*, pages 305–326. Springer, 2016.
- [Gug20] Joël Gugger. Bitcoin-monero cross-chain atomic swap, 2020.

- [HBHW] Daira-Emma Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. *Zcash Protocol Specification*. Living specification, versioned (e.g. Version 2024.5.1 [NU6]).
- [Her18] Maurice Herlihy. Atomic cross-chain swaps. In *Proc. ACM PODC*, pages 245–254, 2018.
- [HGBN21] Daira-Emma Hopwood, Jack Grigg, Sean Bowe, and Kris Nuttycombe. ZIP-224: Orchard shielded protocol, 2021. Final. Activated with NU5. Design docs at <https://zcash.github.io/orchard/>.
- [HLDH16] Tjaden Hess, Matt Luongo, Piotr Dyraga, and James Hancock. EIP-152: Add BLAKE2 compression function ‘F’ precompile, 2016. Status: Final.
- [KG21] Chelsea Komlo and Ian Goldberg. FROST: Flexible round-optimized schnorr threshold signatures. In *Proc. SAC 2020, LNCS 12804*, pages 34–65. Springer, 2021.
- [KMZ20] Aggelos Kiayias, Andrew Miller, and Dionysis Zindros. Non-interactive proofs of proof-of-work. In *Proc. Financial Cryptography and Data Security (FC) 2020, LNCS 12059*, pages 505–522. Springer, 2020.
- [LLW15] Eric Lombrozo, Johnson Lau, and Pieter Wuille. BIP-141: Segregated witness (consensus layer), 2015. Status: Deployed.
- [LPK19] Ying Tong Lai, James Prestwich, and Georgios Konstantopoulos. ZIP-221: FlyClient – consensus-layer changes, 2019. Final. Activated with Heartwood.
- [LW16] Johnson Lau and Pieter Wuille. BIP-143: Transaction signature verification for version 0 witness program, 2016. Status: Deployed.
- [NEA] NEAR / Aurora teams. Rainbow bridge: ETH–NEAR trustless bidirectional bridge. Docs and code; no paper.
- [NHG21] Kris Nuttycombe, Daira-Emma Hopwood, and Jack Grigg. ZIP-244: Transaction identifier non-malleability, 2021. Final. Activated with NU5.
- [Nol13] Tier Nolan. Alt chains and atomic transfers, 2013. Bitcointalk forum post, 21 May 2013. Non-peer-reviewed; pair with Herlihy PODC 2018.
- [PD16] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments, 2016. Self-published whitepaper, DRAFT v0.5.9.2.
- [PSS17] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Proc. EUROCRYPT 2017, LNCS 10210*, pages 643–673. Springer, 2017.
- [SA15] Markku-Juhani O. Saarinen and Jean-Philippe Aumasson. The BLAKE2 cryptographic hash and MAC, RFC 7693, 2015. IETF, Informational.
- [VOSS20] Alex Vlasov, Kelly Olson, Alex Stokes, and Antonio Sanso. EIP-2537: Precompile for BLS12-381 curve operations, 2020. Status: Final.
- [WB19] Riad S. Wahby and Dan Boneh. Fast and simple constant-time hashing to the BLS12-381 elliptic curve. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(4):154–179, 2019.

- [WE20] Martin Westerkamp and Jacob Eberhardt. zkrelay: Facilitating sidechains using zksnark-based chain-relays. In *Proc. IEEE EuroS&P Workshops (S&B)*, pages 378–386, 2020.
- [WNT20] Pieter Wuille, Jonas Nick, and Anthony Towns. BIP-341: Taproot: Segwit version 1 spending rules, 2020. Status: Deployed (activated Nov 2021, block 709632, with BIP-340/342).
- [XZC⁺22] Tiancheng Xie, Jiaheng Zhang, Zerui Cheng, Fan Zhang, Yupeng Zhang, Yongzheng Jia, Dan Boneh, and Dawn Song. zkbridge: Trustless cross-chain bridges made practical. In *Proc. ACM CCS*, pages 3003–3017, 2022.
- [Zca26] Zcash ecosystem (ECC, ZF, Shielded Labs et al.). NU6.3 network upgrade (ironwood; ZIP-258 et al.), 2026. Mainnet activated 2026-07-28, block 3428143. ZIP paperwork still Draft at time of writing; cite with spec snapshot.
- [ZHL⁺19] Alexei Zamyatin, Dominik Harz, Joshua Lind, Panayiotis Panayiotou, Arthur Gervais, and William J. Knottenbelt. XCLAIM: Trustless, interoperable, cryptocurrency-backed assets. In *Proc. IEEE Symposium on Security and Privacy (S&P)*, 2019.

A Finality (G8): fork resistance

Model. Blocks of S arrive as a Poisson process. At S ’s real difficulty the whole network produces one block per target interval τ in expectation, and the honest miners a fraction $1 - \alpha$ of these. The adversary produces blocks at rate λ_A per τ at the *difficulty floor* $D_{\min}$. With an accurate floor, $\lambda_A = \alpha$; with a floor below the real difficulty D , $\lambda_A = \alpha D / D_{\min}$. We assume difficulty is constant over the window of one order, which DigiShield’s response time makes accurate for windows of a few hours or less. Blocks that honest miners produce on branches they later abandon because of network delay Δ are counted towards λ_A , as in the backbone analysis [GKL15, PSS17]. The *honest chain* is the chain honest nodes hold at the end of the order’s window. By k -common prefix [GKL15, GKL17], all but its last few blocks are permanent except with probability exponentially small in their depth. We write $q = \lambda_A / (1 - \alpha)$ and require $q < 1$.

Lemma A.1 (Lead at acceptance). *Let B_{ref} be the honest tip at time t_{acc} , and let L be the length of the longest private chain the adversary has mined on top of B_{ref} by then. Then $\Pr[L \geq \ell] \leq q^\ell$ for all $\ell \geq 0$.*

Proof. Let W be the adversary’s private lead: the length of its best private chain minus that of the honest chain, both measured from the last block they share, floored at 0 because a rational adversary abandons a losing branch. Each block is adversarial with probability $q/(1+q)$ and honest otherwise, so W is a random walk reflected at 0 with up-step probability $q/(1+q)$. Its stationary distribution is geometric, $\Pr[W \geq \ell] = q^\ell$, and because the walk starts at its minimum state 0, its distribution at any fixed time is stochastically dominated by the stationary one. Since B_{ref} is the honest tip, a private chain on top of it of length ℓ is a private lead of ℓ , so $L \leq W$. $\square$

Remark A.2. If B_{ref} lies j blocks below the honest tip at t_{acc} , the adversary may already have up to j more private blocks on top of it, and every bound below holds with k replaced by $k - j$. The reference block should therefore be fresh: an escrow owner who checks its own node chooses $j \approx 0$, while the default of the most recent block finalized in mode `sampled` costs as many confirmations as blocks have passed since that finalization.

Theorem A.3 (Fork resistance). *Fix an order with parameters $\Theta = (B_{\text{ref}}, k, T)$, with B_{ref} the honest tip at t_{acc} . The probability that the escrow is released on a main block that is not on the honest chain is at most*

$$\varepsilon_{\text{fork}}(\lambda_{\mathcal{A}}, k, T) + \varepsilon_{\text{FC}}, \quad \varepsilon_{\text{fork}} = \sum_{\ell \geq 0} (1 - q) q^\ell \Pr[\text{Pois}(\lambda_{\mathcal{A}} T / \tau) \geq k - \ell], \quad (4)$$

where ε_{FC} is the FlyClient soundness error for the span from B_{ref} to the main block, which is 0 when the span is no longer than the number of samples.

Proof. Suppose the escrow is released on a main block M^* that is not on the honest chain. By the definition of SPV_k , M^* is the main block of a valid rollup with head blocks $H_1, \dots, H_{k-1}$, the rollup is finalized by $t_{\text{acc}} + T$, and M^* descends from B_{ref} .

1. **Every block of the claim is adversarial.** M^* is not on the honest chain, and every H_i descends from M^* , so none of these k blocks is on the honest chain. Each is in BS by (V1), so each carries valid proof-of-work at difficulty $\geq D_{\text{min}}$. By the model they were mined by the adversary, at rate $\lambda_{\mathcal{A}}$.
2. **They were mined after B_{ref} and before the deadline.** Descent from B_{ref} is proven either by linked blocks, each in BS, or by FlyClient samples. In the second case an adversary can skip proof-of-work for unsampled blocks with probability at most ε_{FC} [BKLZ20]; when the span has no more blocks than samples, every block is checked and $\varepsilon_{\text{FC}} = 0$. Blocks descending from B_{ref} cannot be mined before B_{ref} exists. The k blocks were therefore mined either before t_{acc} , as part of the adversary's private lead L , or during $[t_{\text{acc}}, t_{\text{acc}} + T]$, where their number N is $\text{Pois}(\lambda_{\mathcal{A}} T / \tau)$.
3. **Bound.** The event requires $L + N \geq k$. N is independent of L , because it counts only blocks mined after t_{acc} . By Theorem A.1, L is stochastically dominated by a geometric variable G with $\Pr[G = \ell] = (1 - q)q^\ell$, and $\Pr[N \geq k - \ell]$ is nondecreasing in ℓ , so $\Pr[L + N \geq k] \leq \Pr[G + N \geq k] = \sum_{\ell} (1 - q)q^\ell \Pr[N \geq k - \ell]$, which is Equation (4).

□

Proposition A.4 (Honest completion). *If the honest payment is included in the first honest block after t_{acc} and the adversary withholds all of its blocks, the honest claim finalizes in time with probability at least $\Pr[\text{Pois}((1 - \alpha)T' / \tau) \geq k]$, where T' is T minus the time needed to confirm the claim on R . If it does not, the escrow refunds (G7).*

Corollary A.5 (Choosing the deadline). *Let $T = ck\tau$ with $1/(1 - \alpha) < c < 1/\lambda_{\mathcal{A}}$. Then both $\varepsilon_{\text{fork}}$ and the probability that an honest claim misses the deadline decay exponentially in k . The interval for c is nonempty exactly when $\lambda_{\mathcal{A}} < 1 - \alpha$, which under an accurate floor is honest majority.*

Proof. Write $\mu = \lambda_{\mathcal{A}} ck < k$ and G, N as in the proof of Theorem A.3. For $\theta > 0$ with $qe^\theta < 1$, $\mathbb{E}[e^{\theta(G+N)}] = \frac{1-q}{1-qe^\theta} e^{\mu(e^\theta-1)}$. Take $e^\theta = k/\mu = 1/(\lambda_{\mathcal{A}} c)$; then $qe^\theta = q/(\lambda_{\mathcal{A}} c) = 1/((1 - \alpha)c) < 1$. Markov's inequality gives

$$\varepsilon_{\text{fork}} \leq \frac{1 - q}{1 - 1/((1 - \alpha)c)} e^{-\mu \left(\frac{e\mu}{k} \right)^k} = \frac{1 - q}{1 - 1/((1 - \alpha)c)} \exp(-k(\lambda_{\mathcal{A}} c - 1 - \ln(\lambda_{\mathcal{A}} c))),$$

which is exponentially small in k because $\lambda_{\mathcal{A}} c < 1$. The Chernoff bound for the lower tail of $\text{Pois}((1 - \alpha)ck)$, with $(1 - \alpha)c > 1$, gives the honest-completion statement. □

Remark A.6 (Effect of the difficulty floor). Theorem A.3 depends on the floor only through $\lambda_A = \alpha D / D_{\min}$. A floor at half the real difficulty doubles the adversary’s effective block rate. With $\alpha = 0.2$ this moves the row of Table 3 to roughly that of $\alpha = 0.4$. Participants who recompute the real difficulty off-chain do not depend on $D_{\min}$; everyone else should treat it as a trusted parameter.

B Payout unforgeability (G2) and binding (G3)

B.1 Binding of a toy identifier

Let $\mathcal{H}_i : \{0, 1\}^* \rightarrow \{0, 1\}^{256}$ for $i \in \{1, 2, 3, 4\}$ and $\mathcal{H}_{\text{outer}} : \{0, 1\}^{1024} \rightarrow \{0, 1\}^{256}$ be collision-resistant. Let $X = (pk_d, g_d, v, \text{memo})$ and

$$\text{gen_txid}(X) = \mathcal{H}_{\text{outer}}(\mathcal{H}_1(pk_d) \parallel \mathcal{H}_2(g_d) \parallel \mathcal{H}_3(v) \parallel \mathcal{H}_4(\text{memo})).$$

The toy captures the shape of the ZIP 244 tree: an outer digest over fixed-length inner digests.

Theorem B.1 (Binding). *If $\mathcal{H}_1, \dots, \mathcal{H}_4, \mathcal{H}_{\text{outer}}$ are collision-resistant, no PPT $\mathcal{A}$ finds $X \neq Y$ with $\text{gen_txid}(X) = \text{gen_txid}(Y)$ except negligibly.*

Proof. By reduction. Suppose $\mathcal{A}$ succeeds with non-negligible ϵ ; we build $\mathcal{B}$ using $\mathcal{A}$ as a subroutine.

1. $\mathcal{B}$ runs $\mathcal{A}$ and obtains distinct $X = (pk_d, g_d, v, \text{memo})$, $Y = (pk'_d, g'_d, v', \text{memo}')$ with equal outputs.
2. $\mathcal{B}$ forms intermediates $M_X = \mathcal{H}_1(pk_d) \parallel \dots \parallel \mathcal{H}_4(\text{memo})$ and M_Y likewise, so $\mathcal{H}_{\text{outer}}(M_X) = \mathcal{H}_{\text{outer}}(M_Y)$, and checks whether $M_X = M_Y$.
3. **Cases.** If $M_X \neq M_Y$, $\mathcal{B}$ outputs (M_X, M_Y) as an outer collision. If $M_X = M_Y$, fixed-length concatenation forces coordinate-wise equality, and since $X \neq Y$ some coordinate j differs with equal inner digests; $\mathcal{B}$ outputs this pair as an inner collision.
4. **Bound.** One of the five collision advantages is at least $\epsilon/5$, but each is at most 2^{-128} by the birthday bound, so ϵ is negligible.

□

The same argument applies node by node to the real digest tree: two openings with the same root but different revealed leaves yield a BLAKE2b collision at the first node where they differ. Fixed-length children are essential; ZIP 244 achieves them by hashing every subtree to 32 bytes with distinct personalization strings. Together with the binding of Sinsemilla note commitments, which makes *cmx* open to at most one note except with $\text{Adv}_{\text{Sinsemilla}}^{\text{bind}}$, this is G3.

B.2 Payout unforgeability

Theorem B.2 (Payout unforgeability). *No PPT adversary causes R to release an escrow without a transaction on the honest chain whose identifier is $\text{txid}_{\text{expected}}$ and whose output note opens to the request’s (pk_d, g_d, v) , except with probability*

$$\text{Adv}_{\text{Blake2b}}^{\text{CR}} + \text{Adv}_{\text{Sinsemilla}}^{\text{bind}} + \text{Adv}_{\text{tree}}^{\text{CR}} + \epsilon_{\text{fork}} + \text{Adv}^{\text{replay}}. \quad (5)$$

Proof. A release requires an accepted opening ω with $\text{Root}(\omega) = \text{txid}_{\text{expected}}$ that passed the open check, and a claim whose inclusion proof places $\text{txid}_{\text{expected}}$ in the main block of a rollup finalized under Θ . Consider the cases in which the adversary wins.

1. *The main block is not on the honest chain.* This is G8, bounded by $\varepsilon_{\text{fork}}$ (Theorem A.3).
2. *The inclusion proof is false:* the block does not contain $txid_{\text{expected}}$. The proof is a Merkle path to the block's transaction root, so this yields a collision in the block Merkle tree, $\text{Adv}_{\text{tree}}^{\text{CR}}$.
3. *The included transaction has a different effect.* The block contains a valid transaction tx_data' with $H(tx_data') = txid_{\text{expected}}$. If its leaves differ from those revealed in ω , the two give a BLAKE2b collision at the first differing node of the digest tree (Theorem B.1). If its leaves are the same, it contains the same cmx ; if that cmx commits to a note other than the one opened in ω , the two openings break Sinsemilla binding.
4. *The same payment is claimed twice.* The second claim fails the spent-nullifier check (G5), so $\text{Adv}^{\text{replay}} = 0$.

In every other case the honest chain contains a transaction with identifier $txid_{\text{expected}}$ whose output note opens to (pk_d, g_d, v) , so the adversary has not won. Summing gives Equation (5). $\square$

C Frontrunning, replay, script integrity (G4/G5/G11)

Lemma C.1 (G4 on the swap). *On the swap, the payout recipient is fixed at acceptance, so a mempool copier that submits an honest claim's inclusion proof triggers payout to the same party: $\text{Adv}_{\text{swap}}^{\text{front}} = 0$ by construction.*

Game 10 (FrontAnon). The challenger runs an honest sender: a shielded payment to the guardian whose ciphertext head commits to $r = H_s(P)$ for an Entrypoint program P with the fields of Table 4. The honest execution (ω, P) appears in the R mempool. $\mathcal{A}$ sees the mempool and submits competing executions (copy, reorder, or a modified program or opening). It wins by causing payment to a recipient, fee recipient or call different from those in P , or by executing one payment twice.

Theorem C.2. $\text{Adv}^{\text{FrontAnon}} \leq \text{Adv}_{H_s}^{\text{CR}} + \text{Adv}_{\text{Blake2b}}^{\text{CR}} + \text{Adv}^{\text{replay}}$.

Proof. Consider an accepted adversarial execution (ω_A, P_A) for the honest payment.

1. *Double execution:* the request was already completed but the execution was accepted. This contradicts (X4) and spent-set correctness (marked before external calls, reentrancy excluded), so $\text{Adv}^{\text{replay}} = 0$.
2. *Redirect:* $P_A \neq P$ was accepted. Acceptance requires $\text{Root}(\omega_A) = txid$ and a valid path from $H_s(P_A)$ to the root r_A found in the head revealed by ω_A . If that head differs from the honest one, ω_A and the honest opening give a BLAKE2b collision (Theorem B.1). Otherwise $r_A = r$, and a valid path from $H_s(P_A)$ to r with $P_A \neq P$ is a collision in H_s , either in the leaf hash or in the Merkle tree above it.

Copy-paste of identical executions (including relayer-fee resubmission) pays the same recipients once: the first confirmed copy wins and the rest revert, and reordering pays nothing new. $\square$

G5. Axiom 1: R records the spent nullifier and marks the request completed on every claim or execution, before any external call. $\text{Adv}^{\text{replay}} = 0$ by inspection; cited by G2, G4 and G11.

Game 11 (ScriptForge, G11). The challenger runs a lock-script request whose payment commits to the root r of a script tree $\{P_1, \dots, P_n\}$. $\mathcal{A}$ sees the opening and every executed program, and submits executions of its choice. It wins if R runs a program $P \notin \{P_1, \dots, P_n\}$, runs a program before the claim or after the request has completed, or refunds the escrow after the claim.

Theorem C.3. $\text{Adv}^{\text{ScriptForge}} \leq \text{Adv}_{H_s}^{\text{CR}} + \text{Adv}_{\text{Blake2b}}^{\text{CR}} + \text{Adv}^{\text{replay}}$.

Proof. A run of $P \notin \{P_1, \dots, P_n\}$ requires a valid path from $H_s(P)$ to the root found in the stored head. That root is r unless the opening of $\text{txid}_{\text{expected}}$ was forged, which is a BLAKE2b collision (Theorem B.1); and a valid path to r from a leaf outside the tree is a collision in H_s . A run before the claim contradicts (X1), and a run after completion contradicts (X4). A refund after the claim contradicts the claimed flag, which the refund checks first; this is $\text{Adv}^{\text{replay}} = 0$. For the Entrypoint, any change to the target, calldata, token, minOut , fee recipient, nonce or expiry changes P and hence $H_s(P)$, so it falls in the first case. $\square$

Lemma C.4 (Atomic and final execution). *Once a lock-script request is claimed, its escrow can leave only through a successful execution of a committed program. Each execution either completes the request, after which no further execution is possible, or leaves the escrow and the request exactly as they were.*

Proof. The claim disables the refund, and there is no other exit. On success, Theorem 7.4 marks the request completed, which (X4) checks. On a trap, budget exhaustion or a failed check, the transaction on R reverts, and transactions on R are atomic. Boundedness guarantees that one of the two outcomes is reached. $\square$

Program correctness itself is out of scope, exactly as for any contract: a lock-script distributes the escrow as its committed program dictates on the executor's external inputs, and a program that must constrain those inputs does so with a committed key.

D Anonymity (G9), head indistinguishability (G10) and hiding (G15)

Game 12 (AnonSender). The challenger samples $b \leftarrow \{0, 1\}$ with honest senders S_0, S_1 holding distinct input notes of equal value and identical intent: same guardian (pk_d, g_d) , same v , same program P , up to fresh note randomness and distinct nullifiers. The challenger runs the sender- b flow: shielded S payment, R commitment of txid_b , and execution (ω_b, P) . A PPT $\mathcal{A}$ controls the guardian (sees the ciphertext), the relayer (sees the opening on reveal), and all R observers. The channel is anonymous and self-relay is allowed; timing, network identity, and contact metadata are out of scope. $\mathcal{A}$ outputs b' , and $\text{Adv}^{\text{anon}} = |\Pr[b' = b] - 1/2|$.

Axioms on S .

(A1) Segregation. $\text{gen_txid}(\text{signed}(\cdot)) = \text{gen_txid}(\cdot)$: the witness (spend-authorization signature, proof, randomizers) never enters txid ; direct from ZIP 244.

(A2) Sender absence. There is no PPT $\text{recover}(\text{tx_data}, \text{txid}) \rightarrow \text{sender}$: tx_data carries no identity key, signer binding is witness-excluded, and the spent nullifier is a pseudorandom function of the spent note under the spender's key, so equal-value notes of different senders are unlinkable.

(A3) Head distribution (G10). A ciphertext head carrying a planted root is indistinguishable from a genuine one without the viewing key.

Theorem D.1 (Sender anonymity). $\text{Adv}^{\text{anon}} \leq \text{Adv}^{\text{head-dist}} + \text{negl}(\kappa)$.

Proof. Compare the adversary's views for $b = 0$ and $b = 1$. Everything on S is a pair of valid transactions with identical intent, which (A1)–(A3) make indistinguishable up to $\text{Adv}^{\text{head-dist}} + \text{negl}$. What R and the relayer additionally see is the opening ω_b . Its revealed leaves are the header and compact fields of tx_data_b , which are already part of the S -view, and the output note's openings (ρ, ψ, rcm) . These are not functions of the sender's identity: ρ is the spent nullifier, already in the S -view, and ψ and rcm are fresh randomness. Every other subtree is an opaque digest of data in the S -view. The opening therefore adds nothing that distinguishes S_0 from S_1 beyond the S -view, and the bound follows. The relayer and guardian learn intent by design, but by (A2) nothing mapping it to S_0 versus S_1 . $\square$

G10. Axiom 3: a genuine ciphertext head is an encryption under a key-private scheme and is pseudorandom without the viewing key; a head with a planted root is H_s output at a legal offset, padded with fresh randomness. $\text{Adv}^{\text{head-dist}} \leq \text{negl}$ by the pseudorandomness of note encryption and of H_s , so $v = 0$ signaling is deniable. Cited by G9.

D.1 With the privacy layer

Theorem D.2 (Payout unforgeability with the privacy layer). *With the privacy layer, the bounds of Theorems B.2, C.2 and C.3 each gain the term $\text{Adv}_{\text{Groth16}}^{\text{Sound}}$.*

Proof. With the layer, acceptance checks $\text{Verify}(\text{crs}, \vec{x}, \pi) = 1$ for $\vec{x} = (txid_{\text{expected}}, c_O, E, r)$ instead of an opening. Run the extractor on the adversary's accepted proof. If extraction fails or yields $\vec{w} \notin \mathcal{R}_{\text{priv}}$, the adversary has broken knowledge soundness. Otherwise $\vec{w}$ contains an opening ω with $\text{Root}(\omega) = txid_{\text{expected}}$ whose output note opens to the values committed in c_O , and the proofs of Theorems B.2, C.2 and C.3 apply to ω unchanged. A different note than the one committed in c_O additionally requires breaking the binding of Com, which we fold into $\text{Adv}_{\text{Sinsemilla}}^{\text{bind}}$ when Com is instantiated with Sinsemilla. $\square$

Theorem D.3 (Recipient and value hiding). *In G15, against an adversary that is not the LP and sees only R , the block store and S , before the escrow is claimed,*

$$\text{Adv} \leq 2\text{Adv}_{\text{Groth16}}^{\text{ZK}} + \text{Adv}_{\text{Com}}^{\text{hide}} + \text{Adv}_{\text{NoteCommit}}^{\text{hide}} + \text{Adv}_{\text{enc}}^{\text{ind}}.$$

Proof. By hybrids.

1. **H0:** the real game with $b = 0$.
2. **H1:** as H0, but the proof is produced by the zero-knowledge simulator for the same statement. Cost $\text{Adv}_{\text{Groth16}}^{\text{ZK}}$.
3. **H2:** as H1, but c_O commits to the $b = 1$ values. The simulated proof does not depend on the witness, so the change costs $\text{Adv}_{\text{Com}}^{\text{hide}}$.
4. **H3:** as H2, but the payment on S creates the $b = 1$ note. The adversary sees this payment only through $txid_{\text{expected}}$ and the S -view, where the note appears as cmx and a note ciphertext; the change costs $\text{Adv}_{\text{NoteCommit}}^{\text{hide}} + \text{Adv}_{\text{enc}}^{\text{ind}}$.

5. **H4:** as H3, but with a real proof for the $b = 1$ witness, which is the real game with $b = 1$.
 Cost $\text{Adv}_{\text{Groth16}}^{\text{ZK}}$.

Summing the steps gives the bound. $\square$

With the layer, the hybrid argument of Theorem D.1 additionally replaces the proof by a simulated one on each side, which adds $2\text{Adv}_{\text{Groth16}}^{\text{ZK}}$ to G9.

E Extension: P2SC passive-recipient transfer (optional)

This section is an optional extension. Everything before it is trustless: execution is gated only on proof-of-work inclusion with no custodian on any path. P2SC trades one additional trust assumption—guardian-threshold custody of backing funds—for a passive recipient that never signs and zero contract-side lockup. Deployments that do not need it should ignore this section; no core result depends on it.

E.1 P2SC: pay-to-smart-contract

P2SC removes the contract-side lock entirely. A contract—typically an AMM pool—specifies a shielded address (pk_d, g_d) to be paid. The address is controlled by a guardian threshold custodian (e.g. FROST [KG21, CKGW24]) that is *passive per swap*: unlike adaptor constructions, the recipient address never signs per-swap funding operations, so existing custody key management, including large threshold configurations, is reused unchanged. The guardian’s only job is to hold the spending key for eventual redemption of backing; it signs nothing on the swap path.

Definition E.1 (PayAndClaim). $\text{PayAndClaim}(v, E) \rightarrow 1/0$: any user pays the guardian address with a ciphertext head committing to a lock-script that pays *eth_recipient* subject to *minOut*; then anyone submits the opening, the program and the inclusion proof to the bridge.

The contract treats the proven value v as if wrapped ZEC had arrived (as Uniswap v2 [AZR20] treats ETH) and pays out the paired token at the current pool rate routed through the AMM. Pool accounting ends long proven-credit $pZEC$ and short payout asset; the swapper bears slippage, bounded by the *minOut* committed in the program at *S*-send time. Because that bound predates any *R*-mempool exposure, sandwiching can only move execution within the committed bound, exactly as with an ordinary AMM limit. Replay is prevented by single-use nullifier marking (G5); mempool copiers can only pay the same committed recipient (G4).

Lemma E.2 (Custody separation, G14). *Payout requires no guardian signature: R pays iff the opening and finalized inclusion verify per G2/G4/G11 with a fresh nullifier per G5/G8. Hence even fully corrupt guardians cannot forge payout or inflate R without S inclusion. Redemption of backing S funds, and hence the $pZEC$ peg, does require guardian honesty—the FROST custody assumption, directly analogous to wrapped-custody assets. Corrupt guardians can rug or freeze backing but cannot break unforgeability.*

Proof. The payout branch reads only the opening, the inclusion proof, E and contract storage; no guardian key or signature is an input. Any payout the guardian set could cause is therefore a payout an arbitrary PPT party could cause, which Theorems B.2, C.2 and C.3 bound. $\square$

No per-order EVM lock exists, so AMM liquidity providers suffer no additional locking-induced loss beyond ordinary AMM behavior. To our knowledge no other atomic swap construction offers passive-recipient swaps with zero contract-side lockup: HTLCs lock both sides, and adaptors require both sides to sign.

F Expiry, refund, crowbar and scope (G6, G7, G12, G13)

F.1 Payer protection by expiry (G6)

Let the payment be broadcast at acceptance, when the honest chain is at height h_{acc} , and let $e = E - h_{\text{acc}}$ be the expiry window in blocks. Let T' be the deadline T minus the time needed to build, finalize and confirm the claim on R .

Lemma F.1 (Expiry). *An honest payer ends with its payment mined on the honest chain and the escrow refunded with probability at most*

$$\varepsilon_{\text{exp}} = \Pr[\text{Pois}((1 - \alpha)T'/\tau) < e + k - 1].$$

Proof. The payment can be included only at a height of at most E , by S 's expiry rule. If it is never included, the payer keeps its ZEC and has not lost. If it is included at height $h \leq E$, the claim needs the honest chain to reach height $h + k - 1 \leq E + k - 1$, at which point the payer can build and finalize a rollup with $k - 1$ head blocks. Even if the adversary withholds all of its blocks, the honest chain grows by a Poisson number of blocks with mean $(1 - \alpha)T'/\tau$ in time T' , so it falls short of the required $e + k - 1$ blocks with the stated probability. Otherwise the claim finalizes before the deadline and the refund is impossible. $\square$

The payer thus chooses e small enough, and the requester T large enough, that this probability is negligible at the agreed k ; the buffer between the expected arrival of height E and the deadline is exactly the slack this requires. Larger T also lengthens the adversary's window in Theorem A.3, and Table 3 shows the resulting trade-off.

F.2 Optional crowbar exit

Expiry makes fraud proofs unnecessary: if the payer spends the committed input elsewhere, the committed payment becomes permanently invalid and the escrow refunds at the deadline. Deployments whose payments cannot carry a short expiry, or that want such refunds faster, may add a *crowbar* exit.

Fraud statement. The committed nullifier nf appears in a block finalized under the order's parameters inside a transaction with identifier $txid' \neq txid_{\text{expected}}$. The crowbar submits an inclusion proof for $txid'$ and an opening of $txid'$ that reveals nf ; R then refunds the escrow owner before the deadline.

Game 13 (CrowbarSound). The payer is honest (never spends the committed input elsewhere). A PPT $\mathcal{A}$ submits a crowbar proof and wins if R grants the early refund.

Game 14 (CrowbarComplete). The payer spent the committed input in $txid' \neq txid_{\text{expected}}$, finalized under the order's parameters. An honest challenger submits the genuine fraud proof before the deadline and wins if R accepts it, under bounded-delay R liveness.

Theorem F.2. $\text{Adv}^{\text{CrowbarSound}} \leq \varepsilon_{\text{fork}} + \text{Adv}_{\text{tree}}^{\text{CR}} + \text{Adv}_{\text{Blake2b}}^{\text{CR}}$. *Completeness holds with overwhelming probability given R liveness.*

Proof. Soundness. With an honest payer, no transaction on the honest chain other than the committed payment reveals nf , because a nullifier can be revealed only by spending its note. A winning crowbar therefore relies on a block off the honest chain ($\varepsilon_{\text{fork}}$), a false inclusion proof ($\text{Adv}_{\text{tree}}^{\text{CR}}$), or

an opening of $txid'$ that reveals an nf which the real $txid'$ does not contain ($\text{Adv}_{\text{Blake2b}}^{\text{CR}}$, by Theorem B.1). *Completeness.* With a genuine double spend finalized under the order's parameters, the challenger holds the real inclusion proof and opening, both verify, and by R liveness the transaction lands before the deadline. $\square$

F.3 Equivocation over a commitment

A payer that has fixed $txid_{\text{expected}}$ on R can deviate in exactly three mutually exclusive ways, each covered by a game already proved.

Case I (substitution). It presents a transaction whose effect differs from the committed one under the same identifier. This is a G3 binding break.

Case II (withholding). It never broadcasts the committed transaction, or broadcasts it too late. Above height E the payment can never be mined, so no claim succeeds and the escrow refunds (G7). In the anonymous-first ordering the commitment simply expires (G12).

Case III (diversion). It spends the committed input in a different transaction. The committed payment is then permanently invalid and the escrow refunds at the deadline (G7), or earlier through the optional crowbar exit.

F.4 Refund (G7) and scope (G12/G13)

G7. Axiom 2: without a claim finalized under the order's parameters there is no payout path. The refund depends only on R correctly evaluating its deadline, which the Turing-complete reference ledger enforces under bounded-delay liveness. Honest recovery with loss bounded by gas and time, by construction; there is no cryptographic reduction. Cited by G1.

G12 (relay grief, out of scope). Commit-no-reveal spam, commit-bond sizing, and rate-limiting are deployment policy, not protocol safety. We assume commitments expire and prune (no permanent bloat), self-relay is always available so liveness holds without any specific relay, and P2SC locks no R funds (grief bounded by commit gas, strictly less than HTLC or adaptor lockup). The swap inherits the same timeout-lock grief as HTLCs, bounded by the deadline T . No game.

G13 (AMM execution, out of scope beyond binding). Sandwich and stale-rate behavior is AMM policy. The bridge guarantees only: *minOut* committed in the program at S -send time (before any R -mempool exposure), atomic swap-and-call with reverts, spent-marking (G5), and call binding (G11). Residual slippage equals ordinary AMM slippage; pool solvency and price-feed choice are deployment decisions. No game.

G UC framework

This appendix covers the swap of Sections 4 and 5, including lock-script requests. The Entrypoint and P2SC pay out of pool liquidity rather than a per-request escrow and are covered by the game-based results only.

Functionality $\mathcal{F}_{\text{bridge}}$

State: per session: request O , opening ω , deadline t^* , status $\in \{\text{open}, \text{accepted}, \text{claimed}, \text{done}\}$; globally: USED.

Request. On (REQUEST, sid, O) from the escrow owner: instruct $\mathcal{G}_{\text{ledger}}$ to freeze the escrow, set status **open**, and leak O to $\mathcal{S}$.

Cancel. On (CANCEL, sid) from the escrow owner while status is **open**: return the escrow and set status **done**.

Accept. On (ACCEPT, sid, ω) from a proposer P while status is **open**: if ω passes the open check of Theorem 3.2 against O , record (ω, P) , set $t^* \leftarrow t_{\text{now}} + T$ and status **accepted**, and leak (ω, P) to $\mathcal{S}$.

Claim. On (CLAIM, sid) from any party while status is **accepted** and $t_{\text{now}} \leq t^*$: if the honest chain contains no transaction with identifier $\text{Root}(\omega)$, or its nullifier $nf \in \text{USED}$, return 0. Otherwise ask $\mathcal{S}$; if $\mathcal{S}$ allows, add nf to USED and either release the escrow to P and set status **done**, or, if O binds a script root, set status **claimed**. Return whether the claim succeeded.

Execute. On (EXECUTE, sid, P', ext) from any party while status is **claimed**: if P' is a leaf of the tree committed in ω , ask $\mathcal{S}$; if $\mathcal{S}$ allows, distribute the escrow as P' does on ext and set status **done**.

Refund. On (REFUND, sid) while status is **accepted** and $t_{\text{now}} > t^*$: return the escrow to the owner and set status **done**.

Figure 4: Ideal bridge functionality $\mathcal{F}_{\text{bridge}}$ for the base mechanism. The opening is leaked because the base mechanism publishes it by design.

G.1 Setting

Both ledgers are part of a global setup $\mathcal{G}_{\text{ledger}}$ that is identical in the real and ideal worlds: it holds R 's balances and storage, S 's chain, and a clock. In particular, payments on S and the block store on R are visible to the environment in both worlds and need not be simulated. The *honest chain* of S is defined as in Section A. Every request runs under its own session identifier sid , while the spent-nullifier set on R is shared by all sessions.

G.2 Ideal functionality

Figure 4 gives $\mathcal{F}_{\text{bridge}}$. It interacts with the escrow owner, proposers, any claiming party and the ideal adversary $\mathcal{S}$. Per session it holds the request, the accepted opening and its deadline, and a status; globally it holds the set USED of payments that have already released an escrow.

$\mathcal{F}_{\text{bridge}}$ enforces safety unconditionally and leaves liveness to the adversary: a claim can succeed only for a payment on the honest chain that has not released another escrow, but $\mathcal{S}$ decides whether a permitted claim actually succeeds. This mirrors the real protocol, in which a claim also needs someone to build and finalize a rollup in time, and an execution needs a program that runs within its budget.

Definition G.1 (UC security). Let λ be the security parameter and Π the protocol in the $(\mathcal{G}_{\text{ledger}}, \mathcal{F}_{\text{SPV}})$ -hybrid world, where $\mathcal{F}_{\text{SPV}}$ is the finality predicate of Section 6.3. Then Π UC-realizes $\mathcal{F}_{\text{bridge}}$ if for every PPT adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that no environment $\mathcal{E}$ distinguishes real from ideal:

$$\text{REAL}_{\Pi, \mathcal{A}, \mathcal{E}}^{\mathcal{G}_{\text{ledger}}, \mathcal{F}_{\text{SPV}}}(\lambda) \approx_c \text{IDEAL}_{\mathcal{S}, \mathcal{E}}^{\mathcal{F}_{\text{bridge}}, \mathcal{G}_{\text{ledger}}}(\lambda). \quad (6)$$

Theorem G.2 (UC realization). *If BLAKE2b, H_s and the block Merkle tree are collision-resistant and Sinsemilla note commitments are binding, the base swap protocol of Sections 4 and 5, including lock-script requests, UC-realizes $\mathcal{F}_{\text{bridge}}$ in the $(\mathcal{G}_{\text{ledger}}, \mathcal{F}_{\text{SPV}})$ -hybrid world, except with advantage*

$$\sum_{\text{sid}} (\varepsilon_{\text{fork}}(\Theta_{\text{sid}}) + \varepsilon_{\text{FC}}) + \text{Adv}_{\text{tree}}^{\text{CR}} + \text{Adv}_{\text{Blake2b}}^{\text{CR}} + \text{Adv}_{\text{Sinsemilla}}^{\text{bind}} + \text{Adv}_{H_s}^{\text{CR}}.$$

G.3 Proof of Theorem G.2

In the base mechanism every message the protocol sends is public data that $\mathcal{F}_{\text{bridge}}$ also leaks (the request, the opening and the executed program), and every secret stays on S , which is part of the shared global setup. The simulator is therefore straight-line: it never extracts or equivocates. $\mathcal{S}$ runs $\mathcal{A}$ internally, forwards messages between $\mathcal{A}$ and $\mathcal{E}$, and emulates $\mathcal{F}_{\text{SPV}}$ and the escrow logic of Algorithm 1 towards $\mathcal{A}$.

Simulator.

- *Honest parties.* On each leak from $\mathcal{F}_{\text{bridge}}$ (a request, an accepted opening, a claim, an execution, a cancellation or a refund), $\mathcal{S}$ performs the same step in its emulated escrow with the leaked values. Since the leaked values are exactly what the real party would publish, the emulation is perfect.
- *Corrupt parties.* When $\mathcal{A}$ sends the emulated escrow a request, cancellation, intent, claim, execution or refund on behalf of a corrupt party, $\mathcal{S}$ runs the escrow logic. If the step succeeds in the emulation, $\mathcal{S}$ sends the corresponding input to $\mathcal{F}_{\text{bridge}}$ on that party's behalf.
- *Permissions.* When $\mathcal{F}_{\text{bridge}}$ asks whether a claim or an execution may succeed, $\mathcal{S}$ answers yes exactly when the corresponding step succeeded in its emulated escrow.

Hybrids.

Hyb₀. The real execution.

Hyb₁. As **Hyb₀**, but the execution aborts if a claim succeeds on a main block that is not on the honest chain. The two hybrids differ only on that event, whose probability is at most $\varepsilon_{\text{fork}}(\Theta_{\text{sid}}) + \varepsilon_{\text{FC}}$ per session by Theorem A.3.

Hyb₂. As **Hyb₁**, but the execution also aborts if a claim succeeds although the block contains no transaction with identifier $\text{Root}(\omega)$ whose output note opens as in ω . By cases (2) and (3) of the proof of Theorem B.2, this event yields a collision in the block Merkle tree or in BLAKE2b, or breaks Sinsemilla binding.

Hyb₃. As **Hyb₂**, but the execution also aborts if an execution runs a program that is not a leaf of the committed tree. By Theorem C.3 this yields a collision in H_s or BLAKE2b.

Hyb₄. The ideal execution with $\mathcal{S}$. We show it is identical to **Hyb₃** by comparing every exit of every session.

- *Request, cancel, accept.* The escrow logic and $\mathcal{F}_{\text{bridge}}$ apply the same conditions (status and the open check), and $\mathcal{S}$ forwards corrupt inputs exactly when the emulation accepts them.

- *Claim.* In Hyb_3 , a claim that succeeds has its payment on the honest chain with the accepted effect, and its nullifier is fresh because the real spent set is shared by all sessions. Both are the conditions $\mathcal{F}_{\text{bridge}}$ checks, and $\mathcal{S}$ grants permission exactly when the emulated claim succeeds. A claim that fails in the emulation is denied by $\mathcal{S}$, so it fails in both.
- *Execute.* In Hyb_3 , an execution that succeeds runs a committed leaf after the claim and before completion, which are $\mathcal{F}_{\text{bridge}}$'s conditions, and both distribute the escrow as that program does on the same inputs, by determinism.
- *Refund.* Both refund exactly when the deadline has passed with no successful claim, by the same clock.

Summing the differences between consecutive hybrids gives the bound of Theorem G.2. □

With the privacy layer. The layer replaces the public opening by a hiding commitment and a proof, so the functionality must leak only $\text{Root}(\omega)$ and c_O , and the simulator must produce honest proofs without witnesses and obtain the effect of a corrupt party's accepted proof. Simulated proofs follow from zero knowledge. Obtaining the effect, however, needs a *straight-line* extractor, because a UC simulator cannot rewind the environment. Groth16's knowledge soundness rests on a non-black-box extractor in the algebraic group model, which does not provide this. We therefore claim UC security for the privacy layer only when it is instantiated with a straight-line extractable NIZK, for instance a simulation-extractable SNARK whose witness is also encrypted under a key in the common reference string. With plain Groth16, the privacy layer is covered by the game-based results of Section 9 (Theorems D.2 and D.3) and not by Theorem G.2.

Theorem G.3 (Game coverage). *Under the assumptions of Theorem 8.1, the protocols satisfy G1–G14, and with the privacy layer additionally G15.*

Proof. This is Theorem 8.1, together with Theorems D.2 and D.3 for the privacy layer. □