

ZSwap by Stallion Labs: Shielded Atomic Swap Protocol for Zcash

Technical Architecture

2026-09-22

This document describes the cross-chain swap module: a non-custodial, privacy-preserving atomic swap protocol for shielded ZEC, built on adaptor signatures over RedPallas rather than a hash-timelock contract or a custodial/wrapped-asset bridge.

Goals & non-goals

Goals

- A trustless, two-party atomic swap between shielded ZEC and a counterparty asset (Bitcoin or Solana), with no custodian at any point.
- No on-chain link between the ZEC leg and the counterparty-chain leg — a chain observer sees two ordinary-looking transactions, not a correlatable pair.
- A mechanism built on Orchard's actual, confirmed signing internals (`orchard::pczt::Action::apply_signature`), not an invented primitive.

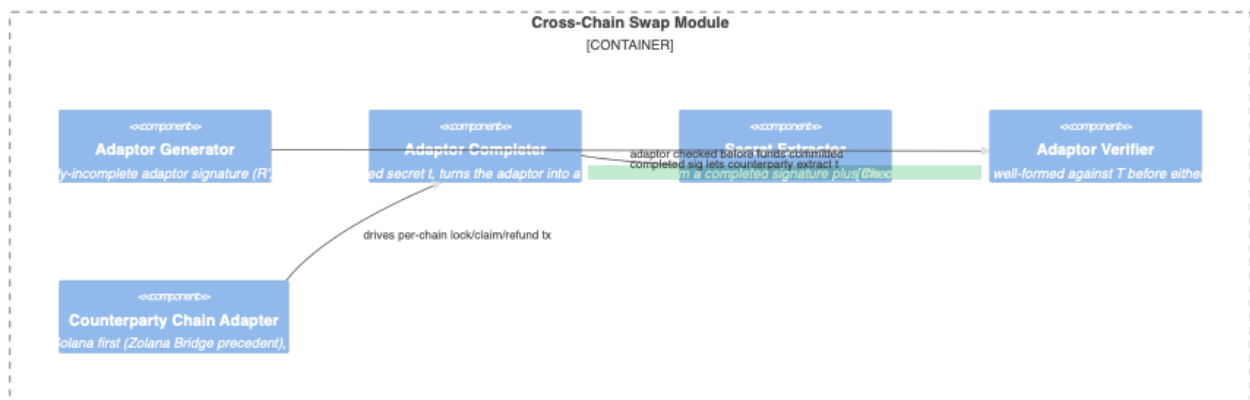
Non-goals (this phase)

- Multi-asset or multi-party swaps ($N > 2$).
- Built-in liquidity or order matching — counterparties find each other externally, or via a thin optional relay.
- A production, audited, mainnet-ready implementation. This phase is specification, feasibility verification, and a testnet demonstration — not a shipped product.

Cross-chain swap module — adaptor signatures, not a bridge

Instead of a wrapped-asset bridge, cross-chain transfer here is a trust-minimized atomic swap: no custodian, no on-chain link between the ZEC leg and the counterparty-chain leg, and no minting of a wrapped ZEC token anywhere.

The core primitive is an adaptor signature over RedPallas (Orchard's Schnorr-variant, on the Pallas curve). Rather than a hash-timelock contract visible on-chain — which would correlate the two legs — the swap logic lives entirely in the signature math, so a chain observer sees only a standard-looking Orchard transaction and a standard-looking counterparty-chain transaction, with nothing tying them together.



Component	Role
Adaptor Generator	Produces the deliberately-incomplete adaptor signature (R', s') from a public point T
Adaptor Completer	Given the shared secret t, turns the adaptor into a valid signature
Secret Extractor	Recovers t from a completed signature plus the original adaptor
Adaptor Verifier	Checks an adaptor is well-formed against T before either party commits funds
Counterparty Chain Adapter	Pluggable per chain, prioritized as Solana first with Bitcoin as fallback — Solana already has a live Orchard bridge path (NEAR Intents + OmniBridge, the "Zolana Bridge") and drew most 2025–2026 hackathon privacy-bridge activity, while Bitcoin's native HTLC scripting is the simpler backstop if Solana's lock-side program is harder to audit in time

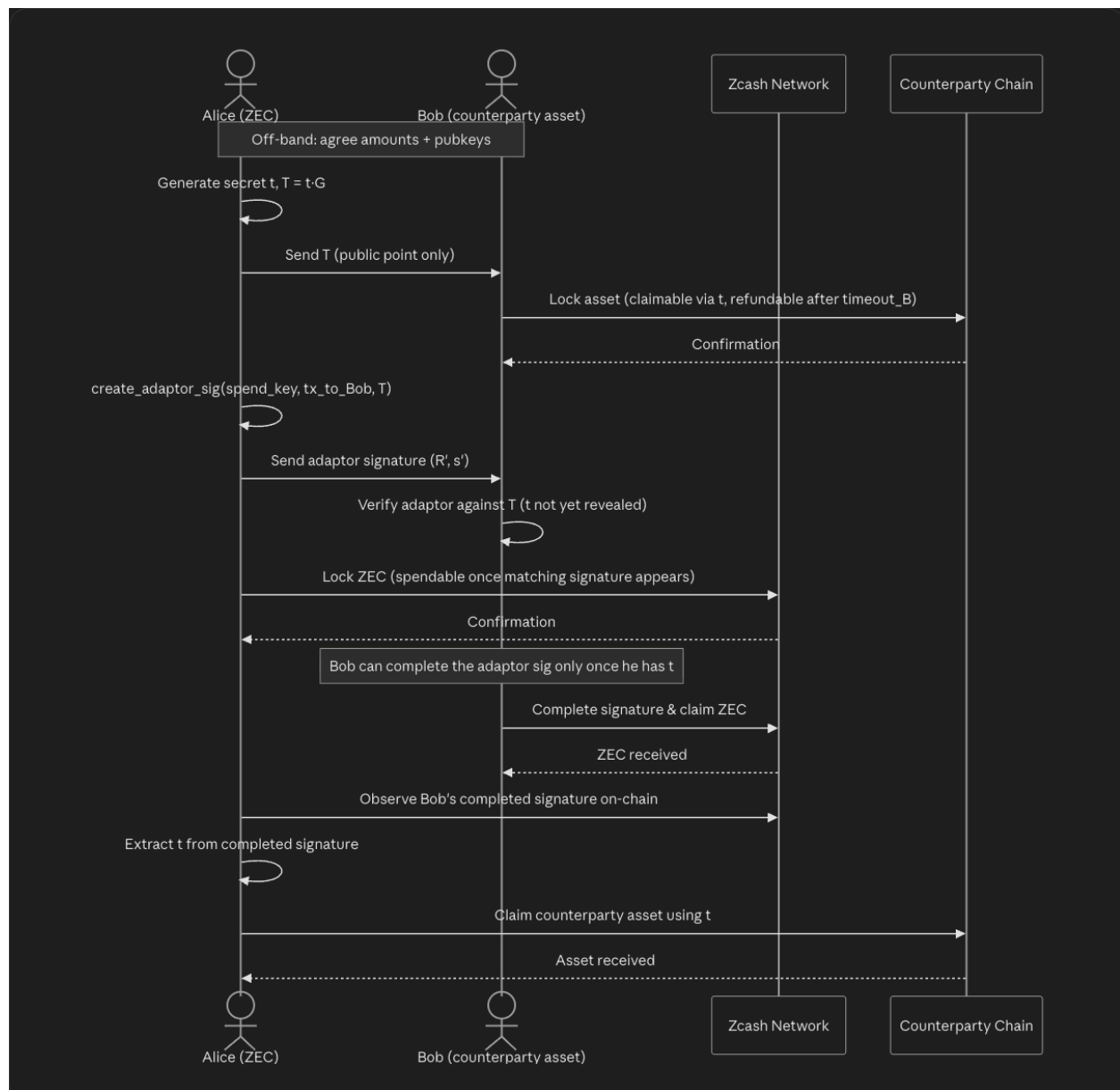
orchard-rs signing hook

This signing hook already exists in `orchard-rs` — no fork needed. The crate's PCZT module lets a spend be built normally, its sighash computed, and a completed adaptor signature injected via `Action::apply_signature`, which accepts an externally-produced `redpallas::Signature<SpendAuth>` and verifies it against the action's stored verification key before applying it.

Source: <https://docs.rs/crate/orchard/latest/source/src/pczt/signer.rs>

Protocol sequence

Alice holds ZEC, wants the counterparty asset; Bob holds it, wants ZEC.



Critical ordering rule

Bob must only be able to complete Alice's adaptor signature after his lock is already irrevocably claimable by Alice via the same secret. Reversing this lets Bob learn t , claim the ZEC, and never let Alice claim the counterparty asset. This ordering needs a formal proof before any real funds move — it is the same subtlety BTC↔XMR adaptor-signature swaps (Farcaster, UnstoppableSwap) had to solve carefully, and it should be treated as a research milestone, not an implementation detail.

Failure modes

If either party never locks, the other never locks either and no funds are at risk; if one party locks and the other stalls, a timeout refunds the locker ($\text{timeout_B} > \text{timeout_A}$, standard HTLC-style margin, hours not minutes to tolerate network delays).

This is deliberately the same design as the ZSwap protocol — the recommendation here is to build against `libzswap-core` (or an equivalent audited crate) as the swap engine behind a defined interface, rather than re-implementing the adaptor-signature math inside the wallet.