Zwap: A Cross-Chain Atomic Swap Protocol Using Multiplicative Key Aggregation

Aditya Bisht
Atheon
aditya@atheon.xyz

Yashwanth Reddy
Atheon
yashwanth@atheon.xyz

***Abstract*—We present Zwap, a cross-chain atomic swap protocol designed for a solver model. The protocol uses an ECDH shared point $P_{\mathsf{SB}} = s \cdot P_B = b \cdot P_S$ as the aggregate locking key, combined with a standard hash lock $\mathsf{H}(s)$ on the counterparty's chain. Multiplicative key aggregation eliminates rogue-key attacks without commitment ordering constraints. Alice provides a zero-knowledge proof, verified off-chain by Bob, binding her secret s to both $\mathsf{H}(s)$ and P_S. All redeem and refund paths are permissionless with hardcoded recipients, eliminating MEV. The protocol is applicable to swaps between any two chains that support the required primitives: EVM-to-UTXO, UTXO-to-UTXO (where at least one side supports hash-lock scripts), and EVM-to-EVM (using ECDSA signature verification on one side and hash verification on the other). The core construction ECDH lock on one leg, hash lock on the other is chain-agnostic; only the lock encoding differs per chain type. We prove protocol soundness under the discrete logarithm assumption, enumerate all edge cases including a novel hash-collision attack (Lemma 3), and show cross-chain unlinkability under ephemeral keys.**

I. Introduction

Cross-chain atomic swaps enable trustless exchange of assets between distinct blockchain networks. The canonical construction uses Hashed Time-Locked Contracts (HTLCs) [1], [2], where a hash $\mathsf{H}(s)$ appears on both chains and the preimage s serves as the atomicity mechanism. This creates a deterministic cross-chain correlation: any observer can match the two legs by comparing hash values.

We present a construction that avoids placing the same hash on both chains to achieve unsuitability. Although we focus exposition on EVM-to-UTXO swaps, the protocol is fundamentally a two-leg construction one ECDH-locked leg and one hash-locked leg that generalizes to any chain pair where each side can enforce the required lock type. UTXO-to-UTXO swaps (e.g., Bitcoin–Zcash) work when one chain enforces the ECDH lock via `OP_CHECKSIG` and the other enforces the hash lock via `OP_SHA256`. EVM-to-EVM swaps work analogously using `ecrecover` for the ECDH lock and `keccak256` for the hash lock.

A. Key Ideas

ECDH aggregation. Rather than additive key aggregation ($P_A + P_B$) as in MuSig [4], we use the ECDH shared point $P_{\mathsf{SB}} = s \cdot P_B = b \cdot P_S$. The signing key is $s \cdot b$ (scalar product). Neither party can manipulate P_{SB} without solving the discrete logarithm problem, eliminating rogue-key attacks regardless of commitment order.

Zero-knowledge binding proof. Alice proves in zero knowledge that her secret s is simultaneously the preimage of $\mathsf{H}(s)$ and the discrete log of P_S. This proof is verified off-chain by Bob before he commits capital. It binds the hash lock and the ECDH key to the same secret, preventing Alice from decoupling the two locks.

Permissionless redeem and refund. Redeem and refund paths have no caller-identity checks. Funds flow to hardcoded recipients. Any relayer can submit transactions on behalf of either party; MEV extraction on the redeem and refund paths is zero. The relayer pays gas (or transaction fees) and is compensated out-of-band.

II. Related Work

Atomic swaps. In the context of cross-chain exchange, atomic refers to the all-or-nothing guarantee: either both legs of the swap settle completely, or neither does there is no intermediate state in which one party receives funds while the other does not. The HTLC construction remains the dominant approach in production systems (e.g., Lightning Network [6], Submarine Swaps), but creates a deterministic cross-chain link via the shared hash, the primary limitation our work addresses.

HTLC incentive attacks. Recent work has demonstrated that HTLCs are vulnerable to incentive-driven attacks. Tsabary et al. [2] showed that a rational miner can bribe validators to censor HTLC timeout transactions at negligible cost (MAD-HTLC attack). Wadhwa et al. [1] proposed He-HTLC to mitigate this via penalty mechanisms. Winzer et al. [3] analyzed temporary censorship attacks more broadly. Zwap's permissionless redeem/refund paths with hardcoded recipients eliminate the MEV vector entirely, though the censorship vulnerability on L2 sequencers remains (§XII).

Cross-chain interoperability. Beyond atomic swaps, Zamyatin et al. [5] proposed XCLAIM, a framework for trustless cross-chain asset transfers using collateralized intermediaries rather than HTLCs. Their approach avoids hash correlation but introduces counterparty risk via collateral. Zwap avoids both: no shared hash and no collateral requirement in the core protocol (though solver bonding is recommended operationally).

Key aggregation. MuSig [4] introduced secure multi-signature key aggregation for Schnorr signatures, addressing rogue-key attacks via commitment ordering and proofs of possession. Our ECDH construction achieves rogue-key resistance through a different mechanism multiplicative (Diffie–Hellman)

aggregation rather than additive at the cost of requiring a zero-knowledge binding proof.

Privacy-preserving swaps. Deshpande and Herlihy [9] formalized privacy-preserving cross-chain atomic swaps and proposed constructions that avoid hash correlation. Thyagarajan and Malavolta [8] introduced lockable signatures, providing a cleaner abstraction for scriptless cross-chain swaps that generalizes adaptor signatures. Our construction differs by operating within the standard script model (no signature scheme modifications) at the cost of an off-chain ZK proof.

Point time-locked contracts (PTLCs). On Schnorr-capable chains (Bitcoin post-Taproot, Zcash NU5), PTLCs provide a strictly simpler path to unlinkable atomic swaps: the adaptor signature is native to the signature scheme, requiring no separate binding proof or hash lock. Zwap's primary advantage over PTLCs is compatibility with ECDSA-only chains, particularly the EVM's `ecrecover` interface, which does not support native Schnorr verification and workarounds are relatively expensive. On chain pairs where both sides support Schnorr, PTLCs are preferred.

III. Preliminaries

A. Notation

- G: generator of the secp256k1 elliptic curve group of prime order q.
- (s, P_S): Alice's ephemeral secret and public key, $P_S = s \cdot G$. The secret s must be sampled uniformly at random from $\mathbb{Z}_q^*$, fresh per swap.
- (b, P_B): Bob's ephemeral keypair, $P_B = b \cdot G$, with b sampled uniformly at random from $\mathbb{Z}_q^*$.
- $P_{\mathsf{SB}} = s \cdot P_B = b \cdot P_S = (s \cdot b) \cdot G$: ECDH shared point. The signing key is $s \cdot b \pmod q$.
- (p_{r_A}, P_{r_A}): Alice's ephemeral key for signing on one chain and/or receiving on the other.
- (p_{r_B}, P_{r_B}): Bob's ephemeral key for signing on one chain and/or receiving on the other.
- (p_{R_A}, P_{R_A}): Alice's refund key (chain-specific, ephemeral).
- (p_{R_B}, P_{R_B}): Bob's refund key (chain-specific, ephemeral).
- T: absolute timelock on the ECDH lock (longer duration).
- c: timelock divisor. The hash lock receives timelock T/c. See Definition 6.
- Δ_{safe}: chain-specific confirmation depth. Instantiated as Δ_{hash} (hash-lock chain) or Δ_{sig} (ECDH-lock chain). See §XI for recommended values.
- δ: execution margin for broadcast, mempool inclusion, and block variance.
- $t_{\mathsf{alice\_reveal}}$: timestamp at which Alice's reveal of s achieves finality on the hash-lock chain.
- $t_{\mathsf{bob\_latency}}$: upper bound on time for Bob to observe s, compute $s \cdot b$, and broadcast a redeem transaction.

Definition 1 (Key Separation)**.** *Each party generates* independent *ephemeral keys for each chain. No key or address derived from it appears on both chains. Furthermore, ephemeral keys must be computationally independent of any public derivation path (e.g., HD wallet derivation from a known master public key), since an observer who recovers P_B from Bob's ECDSA signature could otherwise verify the derivation. This is required for cross-chain unlinkability (§X).*

B. Participants

- **Alice**: end user / swap initiator. Generates secret s. Always locks first (Remark 5).
- **Bob**: solver. Generates (b, P_B). Known, reputable entity. Always locks second.

All keys and addresses are ephemeral.

C. Chain Requirements

The protocol requires one leg to enforce an ECDH signature lock and the other to enforce a hash lock. The specific primitives depend on the chain type:

UTXO chains. Must support: `OP_CHECKSIG` (signature verification), a hash opcode for preimage verification (e.g., `OP_SHA256`), and `OP_CHECKLOCKTIMEVERIFY` (CLTV) for absolute timelocks. These are available on Bitcoin (post-BIP 65), Litecoin, Zcash (transparent addresses), and most Bitcoin-derived UTXO chains. Chains that additionally support `OP_CHECKSEQUENCEVERIFY` (CSV), such as Bitcoin (post-BIP 112) and Litecoin, may use relative timelocks; see Remark 7.

EVM chains. Must support: `ecrecover` (ECDSA signature recovery), `keccak256` (hash function), and standard smart contract deployment.

Remark 2 (Hash Function Selection)**.** *The hash function* H *used in the hash-lock leg (and correspondingly in the binding proof circuit) should be chosen based on the combination of: (i) the opcodes available on the hash-lock chain, (ii) the proving system used for the binding proof, and (iii) on-chain verification cost. The hash must provide collision resistance at or above the curve's security level (*$\sim$*128 bits for secp256k1); see Lemma 3. Subject to this constraint, the cheapest hash for the given chain*$\times$*proving-system combination should be preferred. For example,* `OP_SHA256` *is natural on Bitcoin when using a SNARK over SHA-256;* `keccak256` *is natural on EVM. On UTXO chains offering multiple hash opcodes, the choice should balance on-chain script cost against the in-circuit cost within the chosen proof system.*

D. Public Key Validation

Upon receiving any public key from the counterparty (P_S, P_B, or any P_r, P_R key), the recipient **must** validate:

1) The point is in compressed format (prefix `0x02` or `0x03`) and decompresses to a valid point on the secp256k1 curve.
2) The point is not the identity element (point at infinity).

Since secp256k1 has cofactor 1, any valid non-identity curve point is in the correct subgroup. Failure to validate enables griefing (identity point makes P_{SB} unspendable, locking funds until timeout) and potentially leaks secret bits via invalid-curve attacks.

E. Asset Verification

The protocol is *asset-agnostic*: the locked value on each chain may be native currency (e.g., ETH, BTC), a fungible token (e.g., ERC-20), or any other on-chain asset that can be locked and conditionally released. The lock contracts and scripts are parameterized by the asset type; for ERC-20 tokens on EVM, the token contract address is an additional hardcoded parameter.

Required off-chain check. Before acting on the counterparty's lock, each party **must** independently verify that the lock holds the agreed asset at the agreed amount:

- **Bob (before locking)**: After Alice locks in Phase 1, Bob must verify not only the lock parameters (keys, timelocks, recipient addresses) but also that the correct asset and amount are present. On EVM, this means querying the contract's balance (for native currency) or the token contract's `balanceOf` (for ERC-20). On UTXO chains, this means verifying the output value and, if applicable, the asset identifier (e.g., Zcash transparent value pool).
- **Alice (before revealing s)**: After Bob locks in Phase 1, Alice must verify the same: correct asset, correct amount, correct script or contract parameters. Alice must not reveal s until this verification is complete and the lock has sufficient confirmations.

Failure to verify the asset allows a trivial attack: the counterparty locks a worthless or wrong-denomination token that satisfies the lock structure but not the economic intent. This check is purely off-chain and adds no on-chain cost.

IV. Zero-Knowledge Binding Proof

Before any capital is committed, Alice must prove to Bob that her secret s is simultaneously:

(i) the preimage of the published hash $h = \mathsf{H}(s)$, and
(ii) the discrete log of the published point P_S (i.e., $P_S = s \cdot G$).

Concretely, Alice produces a zero-knowledge proof π for the relation:

$$\mathcal{R} = \{(h, P_S; s) : \mathsf{H}(s) = h \ \wedge \ s \cdot G = P_S\}$$

Bob verifies π off-chain. If verification fails, Bob aborts. The proof must be non-interactive (sent asynchronously during Phase 0), computationally knowledge-sound (Alice knows a valid s), and zero-knowledge (no information about s leaks beyond what $\mathsf{H}(s)$ and P_S already reveal).

Lemma 3 (Hash Collision Attack)**.** *If the hash lock uses a hash function whose collision resistance is below the curve's security level, Alice can steal Bob's funds.*

Proof. The binding proof does not depend on collision resistance: since secp256k1 has prime order q, the constraint $s \cdot G = P_S$ uniquely determines s, and knowledge soundness guarantees Alice knows this s. However, the on-chain hash lock checks only $\mathsf{H}(s_{\mathsf{revealed}}) = h$, not $s \cdot G = P_S$.

Suppose Alice pre-computes a birthday collision (s, s') with $\mathsf{H}(s) = \mathsf{H}(s')$ and $s \neq s'$. She uses s in the binding proof (valid), then reveals s' on the hash-lock chain. The hash check passes. Bob reads s', computes $s' \cdot b$, but $P_{\mathsf{SB}} = (s \cdot b) \cdot G \neq (s' \cdot b) \cdot G$; Bob cannot redeem. Alice refunds at T. □

The hash function used for the hash lock must therefore provide collision resistance commensurate with the curve's security level ($\sim$128 bits for secp256k1). The specific hash is chosen per Remark 2.

Why the binding proof is necessary. Without it, Alice could publish $P_S = s' \cdot G$ and $\mathsf{H}(s)$ for different values $s' \neq s$. The ECDH lock would use $P_{\mathsf{SB}} = s' \cdot P_B$, while the hash lock uses $\mathsf{H}(s)$. When Alice reveals s on the hash-lock chain, Bob computes $s \cdot b$, but the ECDH lock requires $s' \cdot b$. Bob cannot redeem.

Why zero-knowledge. The proof must not reveal s, since knowledge of s would allow anyone to compute $s \cdot b$ (given P_B) and redeem the ECDH lock.

Why off-chain. The proof is verified by Bob during order matching, before either party locks funds. No on-chain verification is needed, keeping costs zero and avoiding any on-chain footprint during the matching phase.

A. Concrete Instantiation

For a direction where $\mathsf{H} = \mathsf{SHA256}$ (e.g., hash lock on a UTXO chain via `OP_SHA256`), the arithmetic circuit takes public inputs (h, P_S) and private witness $s \in \mathbb{Z}_q$, enforcing: (1) $\mathsf{SHA256}(s) = h$ via a Boolean decomposition of the SHA-256 compression function, and (2) $s \cdot G = P_S$ via double-and-add over the secp256k1 base field with non-native field arithmetic.

Performance (measured). Benchmarked on the $\mathsf{H} = \mathsf{SHA256}$, secp256k1 scalar-mul binding circuit:[1]

System	IR	Gates	Prove	Mem	Proof	Verification Time
Groth16 (Circom)	R1CS	197K	1.3 s	230 MB	128 B	150 ms
UltraHonk (BB)	ACIR	23K	1.4 s	380 MB	14.25 KB	200 ms
ProveKit	R1CS	128K	1.0 s	320 MB	550 KB	300 ms

Groth16 offers the smallest proof size (128 B) and is the natural choice when on-chain verification is needed (e.g., the ZK-verified initiation pool in §XII-A), since BN254 pairing precompiles are available on EVM. UltraHonk avoids a trusted setup at the cost of larger proofs ($\sim$14.25 KB); this is acceptable for the off-chain verification in the base protocol. ProveKit achieves the fastest proving time but produces large proofs ($\sim$550 KB), making it suitable for off-chain verification with high-bandwidth communication.

V. Protocol Specification

The protocol is **structurally symmetric**: the same construction applies in both directions, though the security profile differs due to chain-specific finality properties (see §IX-E). Alice always creates the ECDH lock (P_{SB}, timelock T). Bob always creates the hash lock ($\mathsf{H}(s)$, timelock T/c).

[1]Benchmarks performed on Apple M-series hardware. Proof generation is performed by Alice during Phase 0 and is not time-critical. Verification is performed by Bob off-chain and is near-instantaneous for all systems listed. Gate counts are not directly comparable across intermediate representations: ACIR gates in Barretenberg encode higher-level operations than R1CS constraints.

A. Design Principles

Definition 4 (Lock Ordering)**.** *Alice (initiator) always locks first. Bob (solver) always locks second, after verifying Alice's lock.*

Remark 5 (Solver Net-Fund Protection)**.** *Alice-locks-first ordering (Definition 4) ensures that Bob (the solver) never commits capital before independently verifying Alice's lock. If Alice's lock is absent, malformed, or underfunded, Bob simply does not lock and incurs zero loss. This protects the solver's net fund exposure: Bob's worst case before locking is wasted verification effort; his worst case after locking is bounded by the refund timeout. Reversing the order would expose Bob to griefing (Alice never locks, Bob's funds frozen until timeout) and undermine the solver model's viability.*

Definition 6 (Timelock Assignment)**.** *The ECDH lock (redeemed second, by Bob) receives the longer timelock T. The hash lock (redeemed first, by Alice) receives the shorter timelock T/c, where $c > 1$ is the* timelock divisor.

The divisor c controls the ratio of Bob's sweep window to Alice's reveal window. Specifically, Bob has $T - T/c = T \cdot (1 - 1/c)$ time after Alice's reveal deadline to observe s, compute $s \cdot b$, and broadcast his redemption. Larger c gives Bob a proportionally larger sweep window relative to T, but compresses Alice's reveal window, increasing the probability that Alice times out and both sides refund.

We suggest $c = 2$ as a practical default: it splits the total duration evenly between Alice's reveal window ($T/2$) and Bob's sweep window ($T/2$), providing balanced margins for both parties. For chains with slow finality or high-latency observation (e.g., Bitcoin), c closer to $3/2$ may be appropriate to give Bob more time. For chains with fast deterministic finality (e.g., post-merge Ethereum), c up to 3 remains viable since Bob's required sweep time is short.

Remark 7 (Timelock Encoding on UTXO Chains)**.** *UTXO chains vary in their timelock support. Bitcoin (post-BIP 112), Litecoin, and others support both absolute timelocks (`OP_CHECKLOCKTIMEVERIFY`, CLTV) and relative timelocks (`OP_CHECKSEQUENCEVERIFY`, CSV). When relative timelocks are available, T and T/c can be expressed as durations relative to the funding transaction's confirmation. When only absolute timelocks are available (e.g., some Bitcoin-derived chains, older Zcash transparent scripts), the concrete block height or Unix timestamp for T and T/c must be agreed upon during Phase 0, before any capital is committed. Both parties independently compute and verify the absolute values embedded in the scripts. Per BIP 65, the spending transaction must set `nLockTime` $\geq T$ (or T/c) and `nSequence` $<$ `0xFFFFFFFF` on the spending input.*

B. Protocol Overview

Figure 1 illustrates the message flow for Alice on EVM and Bob on UTXO.It is symmetric even with the chains swapped.

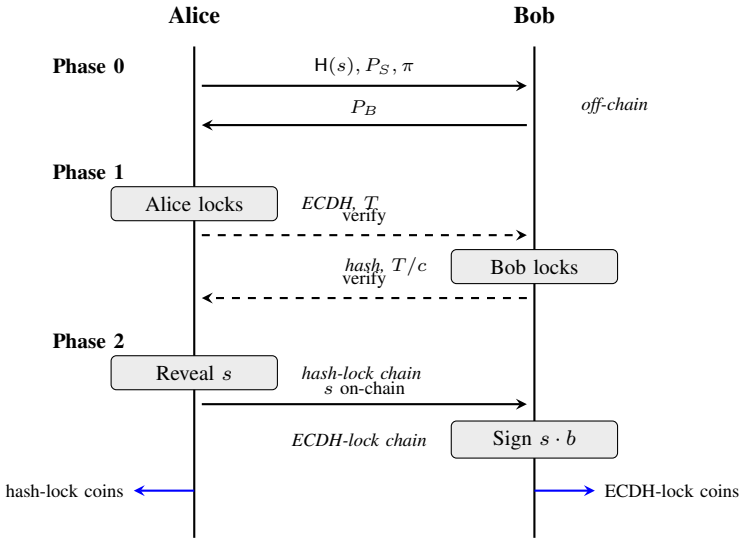

Fig. 1. Solid arrows denote messages; dashed arrows denote on-chain verification. Phase 0 is off-chain. Alice locks first (ECDH lock, timelock T; Remark 5), Bob locks second (hash lock, timelock T/c). Alice reveals s to claim hash-locked coins; Bob computes $s \cdot b$ to claim ECDH-locked coins.

C. Direction 1: Alice on EVM, Bob on UTXO

Alice holds assets on an EVM chain and wants assets on a UTXO chain. Bob (solver) holds assets on the UTXO chain and wants assets on the EVM chain.

1) Phase 0: Order Matching (Off-Chain):

0.1 Agree on: EVM asset and amount $v_{\mathsf{dep}}^{\mathsf{evm}}$, UTXO asset and amount $v_{\mathsf{dep}}^{\mathsf{utxo}}$, absolute timelocks T and T/c (as block heights or Unix timestamps), hash function H for the hash-lock chain (per Remark 2).

0.2 Alice generates (s, P_S), (p_{r_A}, P_{r_A}), and (p_{R_A}, P_{R_A}). Alice sends to Bob:

- $\mathsf{H}(s)$, P_S, P_{r_A}, P_{R_A}
- Zero-knowledge proof π binding s to $\mathsf{H}(s)$ and P_S (§IV)

0.3 Bob validates all received public keys (§III-D), then verifies π. If any check fails, abort. Otherwise, Bob generates (b, P_B), (p_{r_B}, P_{r_B}), and (p_{R_B}, P_{R_B}). Bob sends to Alice:

- P_B, P_{r_B}, P_{R_B}

0.4 Alice validates all received public keys (§III-D). Both compute $P_{\mathsf{SB}} = s \cdot P_B = b \cdot P_S$.

2) Phase 1: Locking: Per Definition 4 and Remark 5, Alice locks first:

1.1 **Alice locks on EVM**: sends $v_{\mathsf{dep}}^{\mathsf{evm}}$ to a contract (or address) encoding:

- Redeem: signature under P_{SB}, recipient $= \mathsf{addr}(P_{r_B})$
- Refund: after T, recipient $= \mathsf{addr}(P_{R_A})$

1.2 Bob verifies: the lock encodes the correct $P_{\mathsf{SB}} = b \cdot P_S$, $\mathsf{addr}(P_{r_B})$, $\mathsf{addr}(P_{R_A})$, and T; the lock holds the agreed asset at $v_{\mathsf{dep}}^{\mathsf{evm}}$ (§III-E).

1.3 **Bob locks on UTXO**: creates a transaction with a script output UTXO-HashLock (see §VI-C), funded with $v_{\mathsf{dep}}^{\mathsf{utxo}}$.

1.4 Alice verifies the UTXO: correct script ($\mathsf{H}(s)$, P_{r_A}, P_{R_B}, T/c), correct asset and amount $v_{\mathsf{dep}}^{\mathsf{utxo}}$ (§III-E), sufficient confirmations ($\geq \Delta_{\mathsf{safe}}$).

3) Phase 2: Redemption:

2.1 Alice reveals s on the UTXO chain and signs with p_{r_A}. Takes UTXO coins. Alice must reveal before the deadline $t_{\mathsf{alice\_deadline}}$ (Equation (1)).

2.2 Bob reads s from the UTXO blockchain.

2.3 Bob computes $s \cdot b \pmod q$ and produces an ECDSA signature under P_{SB}.

2.4 Bob redeems the EVM lock. Tokens flow to $\mathsf{addr}(P_{r_B})$. Bob must redeem before the deadline $t_{\mathsf{bob\_deadline}}$ (Equation (2)).

D. Direction 2: Alice on UTXO, Bob on EVM

Alice holds assets on a UTXO chain and wants assets on an EVM chain. Bob (solver) holds assets on the EVM chain and wants assets on the UTXO chain.

1) Phase 0: Order Matching (Off-Chain):

0.1 Agree on: UTXO asset and amount $v_{\mathsf{dep}}^{\mathsf{utxo}}$, EVM asset and amount $v_{\mathsf{dep}}^{\mathsf{evm}}$, absolute timelocks T and T/c, hash function H for the hash-lock chain (per Remark 2).

0.2 Alice generates (s, P_S), (p_{r_A}, P_{r_A}), and (p_{R_A}, P_{R_A}). Alice sends to Bob:

- $\mathsf{H}(s)$, P_S, P_{r_A}, P_{R_A}
- Zero-knowledge proof π binding s to $\mathsf{H}(s)$ and P_S

0.3 Bob validates all received public keys (§III-D), then verifies π. If any check fails, abort. Otherwise, Bob generates (b, P_B) and (p_{R_B}, P_{R_B}). Bob sends to Alice:

- P_B, P_{R_B}

0.4 Alice validates all received public keys (§III-D). Both compute $P_{\mathsf{SB}} = s \cdot P_B = b \cdot P_S$.

2) Phase 1: Locking: Per Definition 4 and Remark 5, Alice locks first:

1.1 **Alice locks on UTXO**: creates a transaction with a script output UTXO-SigLock (see §VI-D), funded with $v_{\mathsf{dep}}^{\mathsf{utxo}}$.

1.2 Bob verifies the UTXO: $P_{\mathsf{SB}} = b \cdot P_S$, P_{R_A}, T, correct asset and amount $v_{\mathsf{dep}}^{\mathsf{utxo}}$ (§III-E), sufficient confirmations ($\geq \Delta_{\mathsf{safe}}$).

1.3 **Bob locks on EVM**: sends $v_{\mathsf{dep}}^{\mathsf{evm}}$ to a contract (or address) encoding:

- Redeem: preimage of $\mathsf{H}(s)$, recipient $= \mathsf{addr}(P_{r_A})$
- Refund: after T/c, recipient $= \mathsf{addr}(P_{R_B})$

1.4 Alice verifies: the lock encodes the correct $\mathsf{H}(s)$, $\mathsf{addr}(P_{r_A})$, $\mathsf{addr}(P_{R_B})$, and T/c; the lock holds the agreed asset at $v_{\mathsf{dep}}^{\mathsf{evm}}$ (§III-E).

3) Phase 2: Redemption:

2.1 Alice redeems the EVM hash lock by revealing s. Tokens flow to $\mathsf{addr}(P_{r_A})$.

2.2 Bob reads s from the EVM blockchain.

2.3 Bob computes $s \cdot b \pmod q$ and signs under P_{SB}.

2.4 Bob redeems Alice's UTXO coins via the sig-lock path.

E. Structural Comparison

	D1: Alice on EVM	**D2: Alice on UTXO**
Alice locks	EVM (ECDH, T)	UTXO (ECDH, T)
Bob locks	UTXO (hash, T/c)	EVM (hash, T/c)
Alice reveals s	UTXO chain	EVM chain
Bob signs $s \cdot b$	EVM chain	UTXO chain

In both directions: Alice always locks the ECDH lock first (timelock T, per Remark 5). Bob always locks the hash lock second (timelock T/c). Alice redeems the hash lock by revealing s. Bob redeems the ECDH lock by signing with $s \cdot b$.

VI. Lock Specifications

This section specifies the lock semantics for each chain type. The protocol is agnostic to the deployment mechanism: on EVM chains, locks may be implemented as pre-deployed singleton contracts, deterministic `CREATE2` deployments (see §VII), or any other contract pattern that enforces the specified redeem and refund conditions. On UTXO chains, locks are standard P2SH scripts.

A. EVM: ECDH Lock (Alice locks)

EVM-ECDHLock:

Parameters: $\mathsf{addr}(P_{\mathsf{SB}})$, $\mathsf{addr}(P_{r_B})$, $\mathsf{addr}(P_{R_A})$, $v_{\mathsf{dep}}^{\mathsf{evm}}$, T

redeem *permissionless*:
1. Assert `block.timestamp` $< T$
2. Compute a domain-separation digest h from the contract's hardcoded parameters
3. Assert $\mathsf{ecrecover}(h, v, r, \sigma) = \mathsf{addr}(P_{\mathsf{SB}})$
4. Transfer balance to $\mathsf{addr}(P_{r_B})$

refund *permissionless*:
1. Assert `block.timestamp` $\geq T$
2. Transfer balance to $\mathsf{addr}(P_{R_A})$

The domain-separation digest h must bind the signature to all swap parameters (addresses, amount, timelock, chain ID) to prevent replay across swaps and chains. The exact encoding is implementation-defined.

Remark 8 (EVM Point Handling)**.** *The EVM has no native precompile for secp256k1 point decompression. The contracts therefore store* $\mathsf{addr}(P_{\mathsf{SB}})$ *(20-byte Ethereum address) rather than the raw curve point* P_{SB}*. Both parties compute* $\mathsf{addr}(P_{\mathsf{SB}}) = \mathsf{addr}((s \cdot b) \cdot G)$ *during Phase 0 from their shared knowledge of* P_{SB}*. The same applies to all public key parameters: only addresses appear on-chain.*

B. EVM: Hash Lock (Bob locks)

EVM-HashLock:

Parameters: $h_s = \mathsf{H}(s)$, $\mathsf{addr}(P_{r_A})$, $\mathsf{addr}(P_{R_B})$, $v^{\mathsf{evm}}_{\mathsf{dep}}$, T/c

redeem *permissionless*:
1. Assert `block.timestamp` $< T/c$
2. Assert $\mathsf{keccak256}(s_{\mathsf{preimage}}) = h_s$
3. Transfer balance to $\mathsf{addr}(P_{r_A})$

refund *permissionless*:
1. Assert `block.timestamp` $\geq T/c$
2. Transfer balance to $\mathsf{addr}(P_{R_B})$

Remark 9 (Permissionless Execution)**.** *Both EVM contracts have no `msg.sender` check on either the redeem or refund paths. Any party can invoke either function. Funds always flow to the hardcoded recipient. This provides: (i) zero MEV; (ii) the non-EVM party need not interact with EVM directly; (iii) any relayer can submit on behalf. The relayer pays gas and is compensated out-of-band. The timelock boundary is clean: at `block.timestamp` $= T$ (or T/c), redeem fails (requires $< T$) and refund succeeds (requires $\geq T$), so no race condition exists at the boundary.*

Remark 10 (Implementation Considerations)**.** *For native currency locks, implementations must use the checks-effects-interactions pattern (or a reentrancy guard) for balance transfers. If the recipient is a smart contract without a `receive()` or `fallback()` function, a plain native transfer reverts and funds become stuck; a pull-payment pattern is recommended. For ERC-20 token locks, the contract holds the token balance and uses `transfer()` to the hardcoded recipient on redeem or refund; the lock must specify the token contract address as an additional hardcoded parameter.*

C. UTXO: Hash Lock Script

Used when Bob locks on a UTXO chain.

UTXO-HashLock (P2SH):

```
OP_IF
⟨hash_opcode⟩ ⟨H(s)⟩ OP_EQUALVERIFY
⟨P_rA⟩ OP_CHECKSIG      // Alice redeems with s + sig
OP_ELSE
⟨T/c⟩ OP_CLTV OP_DROP
⟨P_RB⟩ OP_CHECKSIG      // Bob refunds after T/c
OP_ENDIF
```

Where hash_opcode is the chain's hash verification opcode matching H (e.g., `OP_SHA256`).

D. UTXO: ECDH Sig Lock Script

Used when Alice locks on a UTXO chain.

UTXO-SigLock (P2SH):

```
OP_IF
⟨P_SB⟩ OP_CHECKSIG      // Bob redeems with sig under P_SB
OP_ELSE
⟨T⟩ OP_CLTV OP_DROP
⟨P_RA⟩ OP_CHECKSIG      // Alice refunds after T
OP_ENDIF
```

VII. Deterministic Deployment on EVM

On EVM chains, the lock contracts can optionally use the `CREATE2` pattern [7] for deterministic deployment. Funds are sent directly to a precomputed address derived from the contract bytecode and swap parameters. The contract is not deployed until redemption, when the redeemer deploys and redeems atomically in a single transaction.

The contract address is deterministically computed as:

$$\mathsf{addr}_{\mathsf{contract}} = \mathsf{keccak256}(\texttt{0xff} \| \mathsf{addr}_{\mathsf{factory}} \| \mathsf{salt} \| \mathsf{keccak256}(\mathsf{bytecode}))[12:]$$

The salt encodes all swap parameters (lock key or hash, recipient addresses, amount, timelock, chain ID), ABI-encoded and hashed. Both parties compute this salt independently and verify agreement before locking.

Advantages. No on-chain contract exists before redemption: the funded address is indistinguishable from a regular EOA transfer, reducing the observable footprint and delaying structural fingerprinting until the moment of redemption. This benefit is time-limited (see §X-D).

Requirements. The `CREATE2` factory must be a well-known, immutable, non-upgradeable singleton. It must contain no `SELFDESTRUCT`, no admin functions, and no upgradability proxy. Its interface must support atomic deploy-and-call in a single transaction, and handle the case where a contract already exists at the target address (e.g., due to front-deployment by a third party; see EC-5 in §IX-C).

VIII. Rogue-Key Resistance

Theorem 11 (Rogue-Key Resistance)**.** *Under the discrete logarithm (DLog) assumption on secp256k1, and given that both parties validate received public keys (§III-D), neither party can manipulate P_{SB} to gain unilateral signing capability, regardless of commitment order.*

Proof. The signing key for P_{SB} is $s \cdot b \pmod q$.

Alice tries to bias P_{SB}: Suppose Alice wants $P_{\mathsf{SB}} = X$ for some target point X where she knows the discrete log x (i.e., $X = x \cdot G$). This requires $s \cdot b \equiv x \pmod q$, i.e., $b \equiv x \cdot s^{-1} \pmod q$. Alice knows s and $P_B = b \cdot G$, but extracting b from P_B requires solving the discrete logarithm problem. Alice cannot achieve this under the DLog assumption.

Bob tries to bias P_{SB}: Symmetrically, Bob wants $P_{\mathsf{SB}} = X$ with known x. This requires $s \equiv x \cdot b^{-1} \pmod q$. Bob knows b and $P_S = s \cdot G$, but computing s from P_S requires solving the DLog problem. The zero-knowledge property of π reveals

nothing about s beyond what $\mathsf{H}(s)$ and P_S commit to. Under the DLog assumption, Bob cannot recover s.

Unlike additive aggregation ($P_A + P_B$) [4], where Alice can set $P_A = X - P_B$ to force the aggregate to X without knowing b, multiplicative aggregation via ECDH requires knowledge of the counterparty's scalar to bias the result. □

Corollary 12. *No commitment ordering constraint is needed for rogue-key resistance. Alice can commit P_S first in both directions without security loss.*

Proof. Follows directly from Theorem 11: the DLog hardness argument is symmetric and does not depend on which party commits their public key first. □

IX. Security Analysis

A. Binding Proof Necessity

Lemma 13 (Decoupling Attack Without Binding Proof)**.** *If Alice does not prove that s binds $\mathsf{H}(s)$ and P_S, she can decouple the two locks and steal Bob's funds.*

Proof. Alice publishes $P_S = s' \cdot G$ and $\mathsf{H}(s)$ for $s' \neq s$. The ECDH lock uses $P_{\mathsf{SB}} = s' \cdot P_B$. The hash lock uses $\mathsf{H}(s)$. Alice reveals s on the hash-lock chain, takes Bob's funds. Bob reads s, computes $s \cdot b$, but the ECDH lock requires $s' \cdot b$. Bob cannot redeem. □

B. Timing Constraints

The following deadlines ensure each party has sufficient time to act:

$$t_{\mathsf{alice\_deadline}} = T/c - \Delta_{\mathsf{hash}} - \delta \tag{1}$$

$$t_{\mathsf{bob\_deadline}} = T - \Delta_{\mathsf{sig}} - \delta \tag{2}$$

Alice must reveal s before $t_{\mathsf{alice\_deadline}}$ to ensure finality before T/c. Bob must redeem the ECDH lock before $t_{\mathsf{bob\_deadline}}$ to ensure finality before T. For the protocol to be viable, Bob's sweep window must accommodate observation, computation, and broadcast:

$$T - T/c \geq 3 \cdot (\Delta_{\mathsf{sig}} + t_{\mathsf{bob\_latency}}) \tag{3}$$

The factor of 3 is a conservative engineering margin (not a formally derived bound) that accounts for δ overhead, variance in block production, and network delays. With recommended parameters ($T = 24$h, $c = 2$, $\Delta_{\mathsf{sig}} \leq 1$h, $t_{\mathsf{bob_latency}} \leq 30$min), the left side is 12h and the right side is ≤ 4.5h, satisfying the constraint with ample slack.

C. Edge Cases

EC-1: Bob never locks. Alice locked first (Remark 5). Bob vanishes. Alice waits until T and reclaims funds via the refund path. Loss: opportunity cost only.

EC-2: Alice never reveals s. Both locks exist. Alice does not reveal s before T/c. Bob reclaims the hash lock after T/c. Alice reclaims the ECDH lock after T. Both recover funds. Alice held a free option on the exchange rate during $[0, T/c)$; see the free-option analysis in §XII.

EC-3: Bob fails to sign before T. Alice revealed s and took hash-locked funds. Bob has s but does not redeem the ECDH lock before T. Alice reclaims the ECDH lock at T. Alice gets both sides; Bob loses. Bob's window: $[t_{\mathsf{alice\_reveal}}, t_{\mathsf{bob\_deadline}})$, where $t_{\mathsf{bob\_deadline}}$ is given by Equation (2). Mitigation: automated watchtower; Equation (3).

EC-4: Chain reorg after Alice reveals s. Alice reveals s on the hash-lock chain. Bob reads s, redeems the ECDH lock. A reorg removes Alice's reveal. If Alice re-submits before T/c: no loss. If T/c passes: hash lock refunds to Bob; Bob has already claimed the ECDH side. Alice loses. Mitigation: Alice reveals well before $t_{\mathsf{alice\_deadline}}$ (Equation (1)).

EC-5: Front-running / MEV on EVM. A searcher observes a redeem transaction in the EVM mempool and front-runs it. Since the redeem path sends funds to the hardcoded recipient (not the caller), the searcher gains nothing. If using `CREATE2` deployment (§VII) and a third party front-deploys the contract, the intended redeemer's deployment call will revert; the redeemer must detect this and fall back to calling `redeem()` directly on the already-deployed contract. In all cases, the intended recipient receives funds. Zero extractable value.

EC-6: Alice double-claims. Alice reveals s, takes hash-locked funds, attempts to reclaim ECDH lock. The ECDH lock refund requires time $\geq T$. Bob has until T to sign with $s \cdot b$. Bob reads s and signs. Alice cannot get both (assuming Bob meets his deadline per Equation (2)).

EC-7: UTXO double-spend. *Hash-lock UTXO*: Redeem requires $\mathsf{H}(s)$ preimage + P_{r_A} signature. Refund requires P_{R_B} signature after T/c. No other party can spend. *ECDH-lock UTXO*: Redeem requires signature under P_{SB} (needs $s \cdot b$). Refund requires P_{R_A} signature after T. Alice does not know b; cannot sign under P_{SB} before T (by DLog hardness).

EC-8: Third party observes s. s appears on-chain when Alice redeems the hash lock. A third party learns s but not b. The third party can compute the point $s \cdot P_B = P_{\mathsf{SB}}$, but signing under P_{SB} requires the scalar $s \cdot b$; recovering this scalar from $P_{\mathsf{SB}} = (sb) \cdot G$ is the DLog problem. Funds go to hardcoded recipients regardless. No attack.

EC-9: Invalid binding proof. Alice sends an invalid or missing proof π. Bob verifies π in Phase 0. Verification fails. Bob aborts. No capital is committed.

EC-10: Chain reorg after Bob redeems ECDH lock. Alice reveals s (finalized on hash-lock chain). Bob computes $s \cdot b$, redeems the ECDH lock. A reorg on the ECDH-lock chain removes Bob's redeem transaction. If Bob re-submits before T: no loss. If T passes: Alice refunds the ECDH lock; Alice gets both sides; Bob loses. Mitigation: Bob redeems well before $t_{\mathsf{bob\_deadline}}$ (Equation (2)) and waits for $\geq \Delta_{\mathsf{sig}}$ confirmations.

D. Security Model

We define security via three properties against a probabilistic polynomial-time (PPT) adversary $\mathcal{A}$.

Safety against malicious Alice: $\mathcal{A}$ controls Alice; Bob is honest. $\mathcal{A}$ wins if it redeems the hash-locked funds *and* reclaims

the ECDH-locked funds. The protocol is safe if $\Pr[\mathcal{A} \text{ wins}] \leq \mathsf{negl}(\lambda)$.

Safety against malicious Bob: $\mathcal{A}$ controls Bob; Alice is honest. $\mathcal{A}$ wins if it redeems the ECDH-locked funds *without* Alice having successfully redeemed the hash-locked funds. The protocol is safe if $\Pr[\mathcal{A} \text{ wins}] \leq \mathsf{negl}(\lambda)$.

Liveness: If $\mathcal{A}$ (controlling one party) aborts at any phase, the honest party must recover its locked funds by time $T+\Delta_{\mathsf{safe}}$ with overwhelming probability.

Theorem 14 (Protocol Soundness)**.** *Under the discrete logarithm (DLog) assumption on secp256k1, collision resistance of* H*, knowledge soundness of the binding proof system, an authenticated channel between Alice and Bob during Phase 0, chain finality within* Δ_{safe}*, chain liveness, and parties meeting their deadlines, the protocol satisfies correctness, safety, and liveness.*

Remark 15 (Authenticated Channel)**.** *Phase 0 requires an authenticated channel: each party must be certain that the public keys and proof it receives originate from its intended counterparty, not from a man-in-the-middle. Without authentication, an attacker could substitute keys and redirect funds. In practice, this is achieved by the solver's relay server authenticating sessions, or by both parties committing to a transcript hash of all Phase 0 parameters before proceeding to Phase 1. The specific authentication mechanism is outside the scope of the core protocol.*

Proof. **Correctness.** Follows by inspection of §V: in honest execution, each phase completes within the timing constraints and both parties receive their expected payout. We reduce to DLog and collision resistance of H.

Safety against malicious Alice. Suppose $\mathcal{A}$ (Alice) redeems the hash lock *and* reclaims the ECDH lock. $\mathcal{A}$ produces a binding proof π for $(h, P_S; s)$ (by knowledge soundness, we extract s). To redeem the hash lock, $\mathcal{A}$ reveals some s' with $\mathsf{H}(s') = h$. If $s' = s$: Bob reads s, computes sb, redeems the ECDH lock before T; $\mathcal{A}$ cannot also reclaim. If $s' \neq s$: (s, s') is a collision for H, contradicting collision resistance. Therefore $\Pr[\mathcal{A} \text{ wins}] \leq \mathsf{Adv}^{\mathsf{CR}}_{\mathsf{H}}(\lambda) + \mathsf{negl}(\lambda)$.

Safety against malicious Bob. $\mathcal{A}$ (Bob) produces a valid signature under $P_{\mathsf{SB}} = (sb) \cdot G$ without Alice having redeemed the hash lock. The signing key is $sb \pmod q$. $\mathcal{A}$ knows b and observes $P_S = s \cdot G$, but recovering s from P_S requires solving DLog; the zero-knowledge property of π leaks nothing beyond P_S and $\mathsf{H}(s)$. Without s, $\mathcal{A}$ cannot compute sb, so any valid signature under P_{SB} constitutes an ECDSA forgery. Therefore $\Pr[\mathcal{A} \text{ wins}] \leq \mathsf{Adv}^{\mathsf{DLog}}(\lambda) + \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{ECDSA}}(\lambda)$.

Liveness. If $\mathcal{A}$ aborts at any phase, the honest party recovers via the timelock refund path: if Bob never locks, Alice refunds at T (EC-1); if Alice never reveals, both refund (EC-2); if Alice reveals but Bob fails, Alice refunds at T (EC-3). By chain liveness, refund transactions are included within bounded time. □

E. Non-Atomicity Window

The interval W between the block in which s first becomes observable on the hash-lock chain and the block at which Alice's reveal achieves finality (depth $\geq \Delta_{\mathsf{hash}}$) is the *non-atomicity window*. During W, Bob can read s and redeem while Alice's reveal is not yet final. If a reorg invalidates Alice's reveal during W, Alice may lose (EC-4). The probability is bounded by $\Pr[\text{reorg at depth} \geq \Delta_{\mathsf{hash}}]$.

For Bitcoin at $\Delta_{\mathsf{btc}} = 6$ blocks, this is approximately $< 10^{-3}$ under honest-majority assumptions. On post-merge Ethereum ($\Delta_{\mathsf{evm}} = 64$ blocks ≈ 2 epochs), finality is deterministic barring a $\geq 1/3$ Byzantine quorum.

The non-atomicity risk differs by direction. When s is revealed on a PoW chain (probabilistic finality), W depends on the chain's reorg resistance. When s is revealed on a PoS chain with deterministic finality, W is effectively zero after finalization.

In the solver model, a rational solver waits for Δ_{hash} confirmations. However, if the exchange rate moves significantly during the wait, a profit-maximizing solver may redeem immediately. Alice's mitigation is to reveal well before $t_{\mathsf{alice\_deadline}}$ (Equation (1)).

X. Cross-Chain Unlinkability

A. Threat Model

A passive observer $\mathcal{O}$ monitors both chains: all transactions, scripts, contract state, and events. No access to off-chain communication.

The protocol is *cryptographically unlinkable* if no PPT $\mathcal{O}$ can distinguish a real swap pair from a mismatched pair (composed of legs from two independent swaps) with non-negligible advantage.

B. HTLC Baseline

Standard HTLCs: $\mathsf{H}(s)$ on both chains. Observer hashes the revealed preimage on one chain, matches against the other. Deterministic proof of linkage.

C. Zwap

Theorem 16. *Zwap with ephemeral per-chain keys (Definition 1) is cryptographically unlinkable in either direction.*

Proof. In each swap, $\mathsf{H}(s)$ and s appear on exactly one chain, while P_{SB} appears on exactly the other. By key separation (Definition 1), no key or address appears on both chains.

D1: The UTXO side reveals s and uses keys (P_{r_A}, P_{R_B}). The EVM side stores $\mathsf{addr}(P_{\mathsf{SB}})$; however, once Bob redeems, his ECDSA signature (v, r, σ) is on-chain and any observer can recover the full public key P_{SB} via `ecrecover`. The EVM side thus reveals P_{SB} (post-redemption) along with addresses $(\mathsf{addr}(P_{r_B}), \mathsf{addr}(P_{R_A}))$. Given s from the UTXO side and P_{SB} from the EVM side, $\mathcal{O}$ computes $P_S = s \cdot G$ and $P_B = s^{-1} \cdot P_{\mathsf{SB}}$. However, P_B is sampled uniformly from $\mathbb{Z}_q^*$ and appears nowhere on either chain. In a mismatched pair, $P_B^{(1)} = s_0^{-1} \cdot P_{\mathsf{SB}}^{(1)}$ is equally a uniformly random group element (since

b_1 is independent of s_0). Both distributions are identical. No PPT algorithm can distinguish them.

D2: Symmetric argument with chains swapped.

All receiving, refund, and ECDH keys are ephemeral and per-chain, so no auxiliary on-chain data correlates the two sides. □

D. Statistical and Structural Linkability

The cryptographic unlinkability result eliminates deterministic cross-chain correlation. In practice, statistical and structural side channels are the dominant deanonymization vectors.

Amount correlation. If Alice locks $v_{\mathsf{dep}}^{\mathsf{evm}}$ and Bob locks $v_{\mathsf{dep}}^{\mathsf{utxo}}$ within a narrow time window, $\mathcal{O}$ can check whether the ratio is consistent with the spot exchange rate.

Timing correlation. The protocol's phase structure creates observable temporal dependencies between the locking and redemption steps across chains. Jittered timing reduces correlation but is constrained by the T/c reveal window and the $T - T/c$ sweep window.

Solver deanonymization. Bob is a known entity. Bob's funding sources (UTXO set, EVM hot wallet) are not ephemeral and can be correlated across swaps and chains by graph analysis, even without cryptographic linkage.

UTXO graph analysis. Standard heuristics (common-input-ownership clustering, change output detection) can cluster Bob's swap-related outputs.

Vector	Type	Mitigation
Amount matching	Statistical	Fixed denominations
Timing correlation	Statistical	Batched settlement
Solver funding	Graph	Funding obfuscation
UTXO clustering	Graph	Isolated UTXO sets

The protocol eliminates cryptographic linkability. Statistical and graph-based linkability remains and is the dominant practical threat, particularly in the solver model. Mitigations are operational, not protocol-enforced.

XI. Recommended Parameters

Param	Value	Rationale	Param	Value	Rationale
T	24 h	Bob's sweep win.	δ (EVM)	15 min	Exec. margin
c	2	Balanced split	δ (UTXO)	2 blk	Exec. margin
T/c	12 h	Alice's reveal win.	t_{bob}	30 min	Observe+sign
Δ_{evm}	64 blk	Post-merge final.	Reveal	$T/c-1$h	Reorg+margin
Δ_{btc}	6 blk	BTC PoW depth	Sweep	$T-1$h	Bcast+margin
Δ_{ltc}	12 blk	LTC PoW depth			
Δ_{zec}	24 blk	ZEC PoW depth			

Remark 17 (Conservative Defaults)**.** *The parameters above are calibrated for very conservative security assumptions: deep confirmation requirements, wide execution margins, and generous latency bounds. Practical implementations operating under stronger liveness assumptions for example, a well-connected solver with dedicated full nodes and sub-second monitoring infrastructure can substantially tighten these bounds. Reducing T (and correspondingly T/c) compresses the free-option window (§XII) and reduces Alice's capital lockup in the event of solver failure (EC-1), both of which improve the user experience. For instance, with $\Delta_{\mathsf{btc}} = 1$ (acceptable under strong hashrate assumptions), $\delta = 5$ min, and $t_{\mathsf{bob\_latency}} = 10$ min, one can achieve $T = 6h$ and $T/2 = 3h$ while still satisfying Equation* (3)*, significantly reducing exposure to price fluctuation during the quote lifetime.*

XII. Discussion

Simplicity and gas costs. The EVM contracts use only `keccak256`, `ecrecover`, and basic control flow. The UTXO scripts use standard opcodes available on all major UTXO chains. Estimated EVM gas costs for native currency are as follows.[2] For a pre-deployed singleton contract: locking is a plain transfer ($\sim$21K gas); redeem requires `ecrecover` + storage reads + transfer ($\sim 50-70$K gas); refund requires a timestamp check + transfer ($\sim 40-60$K gas). For the CREATE2 pattern (§VII), locking remains a plain transfer ($\sim$21K gas), but redeem incurs contract deployment overhead ($\sim$32K base + bytecode storage at 200 gas/byte) in addition to the redemption logic, yielding $\sim 100-150$K gas total; refund under CREATE2 is similarly $\sim 80-120$K gas. For ERC-20 or other token standards, locking requires a `transferFrom` or `approve` + deposit ($\sim 50-80$K gas), and redeem/refund costs increase by $\sim 15-25$K gas due to the additional token transfer call.

ECDH vs. additive aggregation. Additive key aggregation ($P_A + P_B$) admits rogue-key attacks in one direction unless commitment ordering or proofs of knowledge are enforced [4]. ECDH aggregation ($s \cdot P_B$) eliminates this entirely under the DLog assumption. The cost is the zero-knowledge binding proof. A further tradeoff is that the relationship $P_{\mathsf{SB}} = s \cdot P_B$ is not publicly verifiable without knowledge of s or b, unlike additive aggregation where $Q = P_S + P_B$ can be checked by any observer. This precludes trustless third-party dispute resolution over aggregate key correctness.

Generality. The protocol works with any combination of chains that can enforce the required lock types. The ECDH lock requires signature verification (ECDSA); the hash lock requires hash preimage verification. Both require timelocks. This covers EVM-to-UTXO, UTXO-to-UTXO (e.g., Bitcoin–Litecoin, Bitcoin–Zcash), and EVM-to-EVM swaps. The protocol is also asset-agnostic: any on-chain asset that can be conditionally locked and released is supported, provided both parties verify the asset and amount off-chain before proceeding (§III-E).

Permissionless relay. The non-EVM party need never interact with EVM infrastructure. Any relayer can submit the redeem or refund transaction. Funds always route to the hardcoded recipient. Relayers must fund their own gas; out-of-band compensation is needed in practice.

Mutual refund before Bob locks. If Bob has not yet locked (Phase 1 is incomplete), Alice's funds are locked until T unless a cooperative cancellation path exists. We recommend

[2]Gas estimates assume post-EIP-1108 precompile pricing: `ecrecover` at 3,000 gas, `keccak256` at $30 + 6$ gas per 32-byte word. Storage costs follow EIP-2929 cold/warm access rules. All figures are for L1 Ethereum; L2 costs are dominated by calldata compression and differ substantially.

including an additional spending condition in Alice's lock: a 2-of-2 multisig (or, on EVM, a function requiring signatures from both $\mathsf{addr}(P_{R_A})$ and $\mathsf{addr}(P_{R_B})$) that allows immediate refund to Alice if both parties agree to cancel. This adds no trust assumption neither party can unilaterally invoke it and avoids the full T lockup when the swap is abandoned early. On UTXO chains this is a standard additional branch in the script; on EVM it is an additional function in the lock contract. If mutual refund is not implemented, Alice must wait until T, which is a significant UX cost under recommended parameters.

Privacy limitations. The protocol achieves cryptographic unlinkability: no deterministic cross-chain link exists under ephemeral keys. However, statistical correlation (amount, timing) and graph-based analysis (solver funding provenance, UTXO clustering) are the dominant practical threats (§X-D). The protocol does not provide forward secrecy: s is revealed on-chain permanently. If Bob's ephemeral key b is later compromised, $s \cdot b$ is retroactively recoverable. Bob should securely delete b immediately after redeeming.

Free option. Alice controls s and decides when (or whether) to reveal it. During $[0, T/c)$, Alice holds a free option on the exchange rate. For an at-the-money option with time to expiry τ and annualized volatility σ, the option value is approximately $V \approx 0.4 \cdot \sigma \cdot S \cdot \sqrt{\tau/365}$, where S is the notional value. For $\sigma \approx 65\%$ and $\tau = T/(c \cdot 24)$ days (e.g., 0.5 days at $T = 24$h, $c = 2$):

$$V \approx 0.4 \times 0.65 \times S \times \sqrt{\tau/365} \approx 0.96\% \cdot S$$

Mitigations: price the option into the spread ($\geq V/S$ markup); require a non-refundable upfront premium; reduce T/c ($V \propto \sqrt{T/c}$); rate-limit concurrent swaps per user. Tighter timelock parameters (Remark 17) directly reduce this optionality.

Alice-locks-first and capital risk. The protocol requires Alice to lock first (Remark 5). If Bob vanishes, Alice's capital is frozen until T. This is mitigated by the mutual refund path (above), solver bonding (below), and tighter timelock parameters (Remark 17).

Operational liveness. Protocol safety depends on Bob meeting his sweep deadline (EC-3). Automated watchtowers and threshold signing committees (e.g., k-of-n solvers sharing b) can strengthen this.

Operational assumptions. The protocol assumes a separate solver registration and bonding mechanism (not specified here) to deter griefing. The binding proof π is verified off-chain; if Bob claims π was invalid post-hoc, no on-chain record adjudicates. The relay server can log proofs for reputation, but trustless dispute resolution requires public verifiability of the ECDH relationship which is not available.

Sequencer censorship on L2. On L2s with centralized sequencers, redeem transactions face potential censorship [3]. The permissionless relay mitigates this: multiple independent relayers can attempt. For stronger guarantees, L1 forced-inclusion mechanisms are recommended.

A. ZK-Verified Initiation Pool on EVM

An extension for EVM-initiated swaps (Direction 1) is to replace individual lock contracts with a shared *initiation pool*: a single on-chain contract that holds deposits for multiple concurrent swaps. Alice deposits into the pool with a commitment to her swap parameters (amount, $\mathsf{addr}(P_{\mathsf{SB}})$, recipient, timelock). Redemptions and refunds are executed by submitting zero-knowledge proofs on-chain that demonstrate knowledge of the correct witness (the ECDH signing key for redeem, or timelock expiry for refund) without revealing which deposit is being claimed.

This design offers several advantages. First, it batches multiple swaps into a single contract address, reducing the amount-correlation and timing-correlation signals described in §X-D: an observer sees deposits into and withdrawals from a shared pool rather than 1-to-1 fund flows. Second, on-chain ZK verification enables trustless dispute resolution for the redemption step, since the proof is publicly verifiable. Third, the pool can enforce additional policy constraints within the proof circuit (e.g., compliance checks, rate limits) without revealing the underlying swap parameters.

The cost is on-chain proof verification. For Groth16 on BN254, verification costs approximately $(181 + 6\ell)$K gas for ℓ public inputs (via EIP-196/197 precompiles, post-EIP-1108 pricing): 6,000 gas per $\texttt{ecMul} \times \ell$ public inputs for the MSM, plus $45{,}000 + 34{,}000 \times 4 = 181{,}000$ gas for the pairing check with four Miller loops. For the binding proof circuit with $2-3$ public inputs, this yields $\sim 193-199$K gas per on-chain verification. This is viable on L2s with low gas costs but may be prohibitive on L1 Ethereum for small swaps. The pool contract must also manage the bookkeeping of deposits and nullifiers to prevent double-claims, adding state and complexity. A full specification of this extension is left to future work, but the binding proof infrastructure described in §IV provides the foundation: the same circuit can be extended to prove correct redemption within a Merkle tree of pool deposits.

XIII. Conclusion

We have presented Zwap, a cross-chain atomic swap protocol that replaces the standard HTLC hash correlation with ECDH-based multiplicative key aggregation. The protocol achieves: (i) rogue-key resistance without commitment ordering, under the discrete logarithm assumption; (ii) cross-chain cryptographic unlinkability, since no common value appears on both chains; and (iii) zero MEV via permissionless redeem and refund paths with hardcoded recipients. The construction is chain-agnostic, supporting EVM-to-UTXO, UTXO-to-UTXO, and EVM-to-EVM swaps using standard primitives. We have provided a complete specification for both swap directions, formal security proofs, an exhaustive edge-case analysis, and a candid assessment of statistical and structural linkability limitations inherent to the solver model.

REFERENCES

[1] S. Wadhwa, J. Stöter, F. Zhang, and K. Nayak, "He-HTLC: Revisiting Incentives in HTLC," in *Proc. IEEE S&P*, 2023.

[2] I. Tsabary, M. Yechieli, A. Manuskin, and I. Eyal, "MAD-HTLC: Because HTLC is Crazy-Cheap to Attack," in *Proc. IEEE S&P*, 2021, pp. 1230–1248.

[3] F. Winzer, B. Herd, and S. Faust, "Temporary Censorship Attacks in the Presence of Rational Miners," in *Proc. EuroS&PW*, 2019, pp. 357–366.

[4] G. Maxwell, A. Poelstra, Y. Seurin, and P. Wuille, "Simple Schnorr Multi-Signatures with Applications to Bitcoin," *Designs, Codes and Cryptography*, vol. 87, no. 9, pp. 2139–2164, 2019.

[5] A. Zamyatin, D. Harz, J. Lind, P. Panez, A. Gervais, and W. Knottenbelt, "XCLAIM: Trustless, Interoperable, Cryptocurrency-Backed Assets," in *Proc. IEEE S&P*, 2019, pp. 193–210.

[6] J. Poon and T. Dryja, "The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments," Technical Report, 2016.

[7] V. Buterin, "EIP-1014: Skinny CREATE2," Ethereum Improvement Proposals, 2018.

[8] S. A. K. Thyagarajan and G. Malavolta, "Lockable Signatures for Blockchains: Scriptless Scripts for All Signatures," in *Proc. IEEE S&P*, 2021, pp. 937–954.

[9] A. Deshpande and M. Herlihy, "Privacy-Preserving Cross-Chain Atomic Swaps," in *Proc. FC*, 2020, pp. 540–549.