ELSA: Efficient Layer-2 Schnorr-based Apparition

Strahn Cryptography
avi@strahn.xyz

Confidential draft, November 13, 2026

Abstract

ELSA is a Lightning-style payment channel for Zcash's Ironwood pool, which has no scripts. It is built from Violet's Constrained Locks (VCLs), which provide three things:

- shielded addresses under joint control;
- pre-signed transactions whose outputs and timing are checked before signing;
- release conditions set by the choice of threshold signing scheme.

Lightning relies on relative timelocks. Zcash has only absolute ones, and enforces them only for transactions with a transparent input. ELSA therefore gives each channel a fixed lifetime that ends at a resolution deadline. It revokes old states with hash locks, so updates are unbounded. One-time viewing keys let a watchtower see a single state rather than the whole channel. We prove balance security under synchrony, assuming the unforgeability of the underlying signing schemes, and machine-check its timing core in Lean. We also analyse what the channel leaks on chain and what it costs. Routing across channels and an implementation are future work.

Contents

1 Introduction **2**

2 What ELSA uses from VCLs **3**

3 The channel **4**
3.1 Parameters and keys 4
3.2 Transactions 4
3.3 Opening 5
3.4 Updating 6
3.5 Closing 6
3.6 Renewal 7

4 Security **7**
4.1 Model 7
4.2 Balance security 8

5 Per-state viewing keys **8**
5.1 Watchtowers 9

6 Privacy **10**

7 Costs **11**

8 Deployment notes **12**

9 Related work **12**

10 Conclusion **13**

A Proof of balance security (Theorem E1) **14**

B Claim status **16**

1 Introduction

A Lightning channel [PD16] needs four script features:

- a 2-of-2 multisignature that holds the funds;
- commitment transactions whose outputs are timelocked relative to when they confirm;
- revocation keys that let the honest party take everything from a stale commitment;
- hash locks for routing payments across channels.

Zcash's shielded pools offer none of them as script. A shielded note is spent with a single spend-authorization signature, and its value and recipient are hidden.

Violet's Constrained Locks [VCL26] supply the pieces that ELSA needs.

- A 2-of-2 FROST key generation gives a shielded address under joint control.
- Transaction prediction [Vis26], with the anchor outside the identifier in the v6 transactions that carry Ironwood bundles, lets the parties sign transactions that spend notes not yet on chain, after checking what those transactions pay and when they can be mined.
- The signing scheme sets what releases a lock. ELSA uses plain FROST for the transactions a party may always broadcast, and plain hashFROST, which completes only with a hash preimage, for revocation.

One feature has no counterpart. Zcash has no relative timelocks, and enforces absolute ones only when a transaction has a transparent input. An ELSA channel therefore has a fixed lifetime. Every commitment expires at a deadline D, and the dispute window after it has a fixed length W.

Contributions.

- The channel, fully specified (Section 3). It has two-level commitments, revocation by hash locks with unbounded updates, an update order that keeps both parties safe when the other stops, a resolution deadline, a backstop that never races a commitment, and renewal.
- Balance security under synchrony (Theorem 4.1). It is proved in Appendix A, assuming the unforgeability of FROST and of plain hashFROST. For plain hashFROST the VCL paper gives only a proof plan (its P2). The timing core is machine-checked in Lean.

- Per-state viewing keys (Section 5). Diversified addresses serve the base protocol. One-time viewing keys, which follow ZIP 32 [ZIP] and Tucker [Tuc26], are independent (Lemma 5.1).
- Watchtowers (Section 5.1). A watchtower learns nothing before a breach, and with one-time viewing keys only the breached state after it (Lemma 5.2).
- What the channel leaks on chain (Section 6), and its fees, rounds and storage (Section 7).

Status. Balance security is proved modulo the VCL paper's P2. Routing across channels, a simulation-based treatment and an implementation are future work (Section 10). Appendix B lists the status of every claim.

2 What ELSA uses from VCLs

We use the VCL paper's definitions and results [VCL26] as black boxes. This section recalls what ELSA relies on.

Joint control. The parties run a 2-of-2 FROST key generation for the channel's spend-authorizing key and share the full viewing key $(\mathsf{ak}, \mathsf{nk}, \mathsf{rivk})$. Either can derive every diversified address of the channel, and decrypt every note sent to it.

Lock transactions. A lock transaction spends notes of a lock together with the lock's transparent *state output*. The state output is a pay-to-script-hash 2-of-2 output of at least 54 zatoshis. The party that did not build a lock transaction runs the VCL check before signing. It recomputes the identifier and the signature digest from the transaction's header, its transparent part and every action's compact data, taking the memos and non-compact digests as given (Skyhook's open check [Vis26]). It also checks the Action-Lock and constraint proofs for the outputs, and the header's timelock fields.

Results we use.

- *Payment binding* [VCL26, V3]: every valid transaction carrying a signature on a checked digest pays exactly the checked outputs.
- *Mutual exclusion* [VCL26, V5]: at most one transaction spending a given state output is mined.
- *Exclusivity* [VCL26, V6]: the notes of a lock leave only through transactions the counterparty checked.
- *Timelock windows* [VCL26, V8]: a transaction with a non-final state input, `lock_time` L and `expiry_height` E is valid exactly at heights η with $L < \eta \leq E$, within its network-upgrade epoch.
- *Adaptation and extraction* [VCL26, P1, H4]: a plain hashFROST pre-signature completes with the preimage. The completed signature reveals the trimmed preimage, from which the full preimage is recovered with at most four hash evaluations.

Completion. A lock transaction can be completed only by a party that knows its outputs' openings and every value-commitment randomness rcv. These form its completion data. Each ELSA transaction is built by the party that must be able to complete it alone.

3 The channel

3.1 Parameters and keys

Heights.

- a deadline D by which the channel must be resolved;
- a dispute window of W blocks, typically one to three days (2000 blocks is about 42 hours at 75-second blocks);
- the height $h^{\text{clean}} = D + W$;
- the activation height U of the next network upgrade, with $h^{\text{clean}} + W \leq U$.

Every pre-signed transaction becomes invalid at U, because it commits to the current branch identifier. The margin of W after h^{clean} lets a party that is online only periodically still get its CleanRedeem mined (Theorem 4.1).

Fees. Each transaction pays a fixed fee agreed at opening. All balances below are net of the fees of the transactions that pay them.

Keys.

- *Channel key.* Alice and Bob run a 2-of-2 FROST key generation, obtaining shares x_A, x_B of the spend-authorizing key with $\mathsf{ak} = [x]\mathcal{G}$. They fix nk and rivk jointly, so both hold the full viewing key.
- *Addresses.* The channel address a_0 is the diversified address with index 0. The lock of state n on Alice's side, L_A^n, is the diversified address with index $2n + 1$, and Bob's, L_B^n, the one with index $2n + 2$. Both parties derive them, and with the shared viewing key both can decrypt every note sent to them. Section 5 replaces these with one-time viewing keys.
- *State outputs.* Q_A, Q_B are secp256k1 keys, and st is the pay-to-script-hash 2-of-2 script of Q_A and Q_B. Every state output holds at least 54 zatoshis under st.
- *Payout addresses.* Each party uses fresh addresses of its own wallet for every payout.
- *Revocation locks.* For each state n, Alice runs plain hashFROST's Lock in reveal mode. She samples a 254-bit preimage $t_{A,n}$, and sends Bob $H_{A,n} = \mathsf{H}_{\text{lock}}(t_{A,n})$, $T_{A,n} = [t_{A,n}]\mathcal{G}$ and a proof $\pi_{A,n}$ of the link. Bob does the same.

3.2 Transactions

Table 1 lists Alice's transactions for state n; Bob's are the mirror image. Each is a lock transaction in the sense of the VCL paper [VCL26]. Its builder knows its completion data. The other party runs the VCL check before signing, and additionally trial-decrypts every output paid to it or to a lock, checking that the recovered opening matches the output's commitment (*recoverability*).

The header fields that produce these windows are:

Transaction	Spends	Pays	Valid at	Built by	Signed by
C_A^n	channel notes, u_0	V_B^n to Bob; V_A^n to L_A^n; new state output u_A^n	$\leq D$	Alice	FROST, ECDSA
CR_A^n	note in L_A^n, u_A^n (non-final)	V_A^n to Alice	$\geq h^{\text{clean}}$	Alice	FROST, ECDSA
WR_A^n	note in L_A^n, u_A^n	V_A^n to Bob	$< h^{\text{clean}}$	Bob	hashFROST on $T_{A,n}$, ECDSA
HE (once)	channel notes, u_0 (non-final)	the original deposits	$> D$	either	FROST, ECDSA

Table 1: Alice's transactions for state n. CR is CleanRedeem, WR WitnessRedeem, HE HardExpire.

- C_A^n: `expiry_height` D;
- CR_A^n: `lock_time` $h^{\text{clean}} - 1$, with a non-final state input;
- WR_A^n: `expiry_height` $h^{\text{clean}} - 1$;
- HE: `lock_time` D, with a non-final state input.
- The commitment C_A^n and CR_A^n carry full signatures, which Alice holds. She can publish them whenever they are valid.
- WR_A^n carries a plain hashFROST pre-signature for $T_{A,n}$, which Bob holds, together with Alice's ECDSA signature for the state input. Bob can complete it only with $t_{A,n}$.
- Alice, as a signer of WR_A^n, sees the transaction she signs; this is why revocation uses the plain variant rather than a blind one.
- WR_A^n spends an output of C_A^n, so it can only be mined after C_A^n.
- HE is built by either party, which hands the other its completion data, so both can complete it.

3.3 Opening

1. Run the key generation, and exchange Q_A, Q_B and the revocation locks for state 0.
2. Agree on the effecting data of the funding transaction F. F spends notes of both parties (or of the funder only) and creates the channel notes at a_0, worth $V_A^0 + V_B^0$, together with the state output u_0. Its identifier is now fixed, although no one has signed it.
3. Build and sign the state-0 sets of both parties and HE, as in step 3 of an update (Section 3.4).
4. Each party signs its own inputs of F, and either broadcasts it.
5. Each party confirms arrival as in the VCL paper: it trial-decrypts the channel notes, matches their commitments, and checks that u_0 exists. The channel is open once both have confirmed.

A party never signs its inputs of F before it holds its own commitment and CR for state 0, and HE. It can therefore always recover its deposit.

3.4 Updating

To move from state n to state $n+1$ with balances (V_A^{n+1}, V_B^{n+1}):

1. *Locks.* Each party sends its revocation lock for state $n+1$ and checks the other's proof.
2. *Build.* Alice builds C_A^{n+1} and CR_A^{n+1}, and sends Bob their data and the opening of the note that C_A^{n+1} pays to L_A^{n+1}. Bob builds WR_A^{n+1}. The mirror steps build Bob's set.
3. *Sign.* For each of C_A^{n+1}, CR_A^{n+1} and WR_A^{n+1}, the non-builder runs the VCL check and the recoverability check. Then:
   - Alice and Bob run the hashFROST pre-signing session for WR_A^{n+1}, and Alice sends her ECDSA signature for its state input. Bob now holds everything for WR_A^{n+1} except $t_{A,n+1}$.
   - They run the FROST sessions for CR_A^{n+1} and then for C_A^{n+1}, and Bob sends his ECDSA signatures for their state inputs. Alice now holds both fully signed.

   Bob's set is signed the same way, with the roles swapped.
4. *Revoke.* Each party, once it holds its own new commitment and CR fully signed, sends its old preimage. Alice sends $t_{A,n}$; Bob checks $\mathsf{H}_{\text{lock}}(t_{A,n}) = H_{A,n}$ and $[t_{A,n}]\mathcal{G} = T_{A,n}$, and stores it. Bob does the same.

Each party enforces the order locally.

- *Alice's rule (R1).* She releases $t_{A,n}$ only after holding a fully signed C_A^{n+1} and CR_A^{n+1}.
- *Bob's rule (R2).* He signs C_A^{n+1} only after holding the pre-signature for WR_A^{n+1}, and considers state $n+1$ final only after receiving a valid $t_{A,n}$.
- *Sequential updates (R3).* A party starts an update only after the previous one is final for it. That is, it must hold its own new commitment and CR fully signed, and have received the counterparty's old preimage.

If the counterparty stops in the middle of an update, the party's guarantee is the minimum of its balances in its last final state and in the state being negotiated (Theorem 4.1).

Updates are unbounded, because old states are revoked rather than outrun as in Decker and Wattenhofer's channels [DW15]. Revocation preimages can be derived from a seed, as with Lightning's shachain, which keeps their storage logarithmic. An off-chain signature on a Merkle root of the state's transactions can authenticate the state; validity on chain always comes from the per-transaction signatures.

3.5 Closing

- *Cooperative.* Both sign one transaction that spends the channel notes and u_0 to the final balances, with `expiry_height` D.
- *Unilateral.* A party broadcasts its latest commitment by $D - \Delta$. The counterparty's balance is paid at once. Its own is paid by CR once h^{clean} is reached.
- *Dispute.* Suppose Alice publishes a revoked commitment C_A^m. Bob sees u_0 spent, or sees the note in L_A^m through the shared viewing key. He completes WR_A^m with $t_{A,m}$ and broadcasts it, together with his ECDSA signature.
- *Backstop.* HE is valid only after D. No commitment can be mined after D, so it never races one. It matters only if nobody closed in time.

3.6 Renewal

A channel cannot outlive U, and should be resolved well before D. To keep it running, the parties *renew*: one transaction spends the channel notes and u_0 and is the funding transaction of a fresh channel, with new keys, new D', W' and U', and the current balances. As in opening, the new channel's first commitments, its CRs and its HE are signed before the renewal transaction, whose identifier is known in advance. If the counterparty refuses to renew, the party closes unilaterally by $D - \Delta$.

4 Security

4.1 Model

Parties. One party, P, is honest. The adversary controls the other party, Q, and every message between them: it may delay, drop or reorder messages, and stop at any point of any protocol. P follows Section 3, including rules R1 and R2 of Section 3.4. For cooperative closes and renewals, P's policy is to sign only if it receives at least its current guarantee (defined below).

Ledger and synchrony. Validity follows the consensus rules recalled in the VCL paper [VCL26]: signatures and proofs, no double spends, the branch epoch, the expiry height, and the lock time with a non-final transparent input. We assume synchrony with delay Δ. A transaction that P broadcasts at height η is in a block of height at most $\eta + \Delta$, provided it is valid at every height from η to $\eta + \Delta$ and no transaction spending one of its inputs is mined first.

Liveness of the honest party. With $K = W - \Delta - 1$, which requires $\Delta < W - 1$:

(L1) P is online at least once in every K consecutive heights, from opening until its last payout is mined.

(L2) If u_0 is still unspent at height $D - \Delta$, P broadcasts its newest fully signed commitment at some height at most $D - \Delta$.

Guarantee. A state f is *final* for P if P holds a fully signed commitment and CleanRedeem for f, and, for $f \geq 1$, has received Q's valid preimage $t_{Q,f-1}$. Let f be the newest final state. If an update to $f + 1$ is in progress, meaning P has signed any transaction of state $f + 1$, P's *guarantee* is $g_P = \min(V_P^f, V_P^{f+1})$. Otherwise it is $g_P = V_P^f$. A guarantee defined from P's newest own commitment instead would be wrong: until Q revokes $f - 1$, Q can still publish C_Q^{f-1}.

Assumptions.

- The VCL results [VCL26]: payment binding (V3), mutual exclusion (V5), exclusivity (V6) and the timelock windows (V8).
- Property U of FROST, which the VCL paper cites, and of plain hashFROST, which is the VCL paper's P2 and has only a proof plan there.
- Unforgeability of ECDSA.
- Correctness of trial decryption.
- The parameter constraint $h^{\text{clean}} + W \leq U$.

4.2 Balance security

Theorem 4.1 (E1: balance security; proved, assuming P2)**.** *Under the model and the assumptions above, except with negligible probability, the honest party P ends up with notes it can spend, worth at least its guarantee g_P at the time the channel closes. All of these notes are mined by height $h^{\text{clean}} + W - 1 < U$.*

The proof is in Appendix A. Its structure is as follows.

1. *Nothing unchecked leaves the channel.* Every transaction that spends the channel notes, a lock note or a state output is one of the pre-signed transactions in Table 1, or a cooperative close or renewal that P signed. This follows from V6, ECDSA unforgeability and property U.
2. *The newest own commitment is never revoked.* By R1, P releases $t_{P,m}$ only after holding a fully signed commitment for $m + 1$.
3. *The channel resolves by D.* By L2, synchrony and V8, u_0 is spent by a commitment, a cooperative close or a renewal by height D. HardExpire is never valid while u_0 is unspent and a commitment still could be.
4. *Case analysis.* The cases are: P's commitment, an unrevoked commitment of Q, a revoked commitment of Q, or a close or renewal P signed. Each pays P at least g_P. The timing of the dispute case is the arithmetic $b + K + \Delta = b + W - 1 < D + W$ for $b \le D$. That arithmetic is machine-checked (`penalty_in_time`).

What the theorem does not cover.

- Fees beyond the fixed per-transaction fees.
- The adversary's ability to delay P's broadcasts beyond Δ, which is the synchrony assumption.
- Griefing: Q can force a unilateral close, so P waits until h^{clean} for its own balance.

The theorem is game-based. A simulation-based treatment in the style of the UC analysis of generalized channels [AEE+21] is future work.

5 Per-state viewing keys

The locks L_A^n need fresh addresses.

Base protocol: diversified addresses. The base protocol uses diversified addresses of the channel's viewing key. Both parties derive them from the state index, Bob can decrypt the notes he needs, and outsiders cannot link them. One viewing key then covers the whole channel, so giving a watchtower viewing access exposes every state.

Extension: one-time viewing keys. Following ZIP 32's derivation of internal keys [ZIP] and the one-time viewing keys of [Tuc26], each state can have its own commitment randomness:

$$\mathsf{rivk}_n = \mathsf{ToScalar}(\mathsf{PRF}^{\mathsf{expand}}_{\mathsf{rivk}}([\mathtt{purpose}] \,\|\, \mathsf{I2LEOSP}_{64}(n))),$$

which gives $\mathsf{ivk}_n = \mathsf{Commit}^{\mathsf{ivk}}_{\mathsf{rivk}_n}(\mathsf{Extract}_{\mathbb{P}}(\mathsf{ak}), \mathsf{nk})$, and with it per-state dk_n and ovk_n. Only rivk changes, so the signing schemes are untouched. The single point of contact is the Action proof, in which the spender supplies rivk_n as the witness for $\mathsf{pk_d} = [\mathsf{ivk}_n]g_{\mathsf{d}}$; both parties can derive it.

Lemma 5.1 (E2: per-state viewing keys; proved)**.** *If* $\mathsf{PRF}^{\mathsf{expand}}$ *is a pseudorandom function, then for an adversary that does not hold* rivk*, the keys* $(\mathsf{ivk}_n)_n$ *are indistinguishable from independent viewing keys for the same* $(\mathsf{ak}, \mathsf{nk})$ *with uniform randomness. In particular, learning some* ivk_n*, for instance by computing a discrete logarithm from a published address, reveals nothing about* ivk_m *for* $m \neq n$*.*

Proof. Replace $\mathsf{PRF}^{\mathsf{expand}}_{\mathsf{rivk}}$ by a random function; by pseudorandomness this is indistinguishable. The values rivk_n are then independent and close to uniform in $\mathbb{Z}_q$, because $\mathsf{ToScalar}$ reduces a 512-bit value. $\mathsf{Commit}^{\mathsf{ivk}}$ adds $[\mathsf{rivk}_n]$ times a fixed generator to a point that depends only on $(\mathsf{ak}, \mathsf{nk})$, so it is perfectly hiding and the ivk_n are independent. Indices with $\mathsf{ivk}_n \in \{0, \bot\}$ are skipped, as in ZIP 32. □

5.1 Watchtowers

A watchtower lets P go offline for longer than K blocks. For each revoked state m of Q, P sends the watchtower a hint and an encrypted package, keyed by the identifier $\tau_m = \mathsf{txid}(C^m_Q)$ of the revoked commitment:

$$\mathsf{hint}_m = \mathsf{H}(\texttt{"hint"}, \tau_m)_{[0..16]}, \qquad \mathsf{blob}_m = \mathsf{AEnc}_{\mathsf{KDF}(\texttt{"key"},\tau_m)}(\mathsf{pkg}_m).$$

The package pkg_m holds everything needed to complete and broadcast WR^m_Q:

- the hashFROST pre-signature and the preimage $t_{Q,m}$;
- Q's and P's ECDSA signatures for the state input. The signature digest does not depend on the anchor, so P can sign in advance.
- the completion data;
- the opening of the note in L^m_Q;
- the spend witness keys ak and nk, and rivk_m.

For every transaction in every new block, the watchtower computes the hint of its identifier. On a match it derives the key, decrypts, picks a fresh anchor, proves the Action, completes the signature with $t_{Q,m}$, and broadcasts.

Lemma 5.2 (W1: watchtower privacy; proved in the random-oracle model)**.** *Model* H*,* KDF *and the identifier's top node as random oracles, and assume* AEnc *is IND$-CPA secure.*

1. *Until a revoked commitment is published, the watchtower's view of its hint and package is indistinguishable from random strings of the same lengths.*
2. *With one-time viewing keys, a package opened after a breach of state* m *reveals* ivk_m *and nothing about the notes of any other state.*

Proof. 1. The identifier τ_m hashes, among other things, the commitment's output commitments and ciphertexts, which carry fresh randomness the watchtower never sees. As a random-oracle output it is unpredictable to the watchtower until the commitment appears on chain. The hint and the key are therefore fresh random-oracle outputs on an unpredictable input: the hint is uniform, and the key is uniform and independent of everything the watchtower sees. IND$-CPA security then makes the package look random.

Transaction	Visible	Hidden
Funding	one P2SH output (script hash only), Ironwood actions	amounts, depositors
Commitment	spend of u_0 revealing the 2-of-2 script; new P2SH output	balances, which output is whose
CleanRedeem / WitnessRedeem	spend of u^n; the height, relative to h^{clean}	amounts, recipient
Cooperative close	spend of u_0	final balances

Table 2: What a chain observer learns from an ELSA channel.

2. The package contains rivk_m, not rivk. By Lemma 5.1, the other states' $\mathsf{ivk}_{m'}$ are independent of ivk_m, so the watchtower can decrypt no other state's notes. It also learns nk. Computing another note's nullifier from nk would need that note's opening, which the watchtower does not have.

□

In the base protocol, without one-time viewing keys, the package would have to contain the channel's rivk. A single breach would then let the watchtower derive the viewing key of the whole channel. This is the practical reason to adopt the extension. Paying watchtowers, and deciding when the package of a revoked state may be deleted, are left open.

6 Privacy

We describe what each observer learns. The analysis is qualitative; a formal leakage statement is future work.

Outside observers. Table 2 lists what an observer of the chain sees for each channel transaction.

- *Hidden.* Ironwood amounts and addresses stay shielded, including which output of a commitment belongs to whom. Updates happen off chain and leave no trace.
- *Visible.* The state outputs are transparent. Each holds a dust value under the same 2-of-2 script, and spending one reveals the script, with Q_A and Q_B. They therefore link the funding transaction, the commitment and the second-level transaction into one chain.
- *Timing.* A CleanRedeem at h^{clean} reveals the channel's dispute window. A WitnessRedeem before h^{clean} reveals that a dispute took place.

A cooperative close is a single transaction with one 2-of-2 transparent input. It stands out from ordinary shielded transactions, but reveals neither amounts nor parties.

The counterparty. Both parties hold the channel's viewing key, so each sees every note of the channel. This is inherent to a two-party channel and is not a leak.

Comparison.

- *Lightning.* Funding and closing amounts are public, and so is the 2-of-2 script unless Taproot key-path spends hide it.

Transaction	transparent	Ironwood actions	fee (zatoshis)
Funding	1	2–3	15 000–20 000
Commitment	2	2	20 000
CleanRedeem, WitnessRedeem	2	2	20 000
HardExpire, cooperative close	2	2	20 000

Table 3: Logical actions and conventional fees under ZIP 317, as we read it.

- *Bolt* [GM17]. Payments within a channel are unlinkable, which ELSA does not aim for; opening and closing remain visible.
- *ELSA.* Amounts and parties are hidden, but the channel's on-chain structure is visible through the state outputs.

Without state outputs. If lock times bound for shielded-only transactions (Section 8), the state outputs could be dropped. Mutual exclusion between a lock's transactions already follows from nullifiers, since they all spend the same note. The channel would then leave only Ironwood actions on chain, indistinguishable from other shielded transactions apart from timing.

7 Costs

Fees. ZIP 317 charges $5000 \cdot \max(2, \ell)$ zatoshis, where ℓ is the number of logical actions. A transparent part counts as the larger of its inputs' size in units of 150 bytes and its outputs' size in units of 34 bytes; each Ironwood action counts one (Revision 1, a draft). A 2-of-2 pay-to-script-hash input is about 260 bytes, so it counts two; a state output of 32 bytes counts one. With Ironwood bundles padded to two actions, as wallets do, Table 3 gives the fees.

A cooperative close costs one transaction, and a unilateral close or a dispute two: the commitment, then CleanRedeem or WitnessRedeem. The counterparty's direct output needs no further transaction.

Rounds per update. The signing sessions of an update are independent, so their rounds can be batched. That gives four message rounds: the locks, one round of nonce commitments for all sessions, one round of responses, and the revocation. Each round goes in both directions.

Proofs. Every lock transaction needs its Ironwood Action proofs at broadcast, as any shielded transaction does. At update time it also needs an Action-Lock proof per output and a constraint proof. These are computed off chain and dominate an update's latency. We have not measured them.

Storage. A party must keep, for every revoked state of its counterparty, the data needed to complete the corresponding WitnessRedeem:

- the pre-signature;
- the ECDSA signature;
- the completion data;
- the preimage, which can be derived from a seed.

This is on the order of a kilobyte per state, so storage grows linearly in the number of updates. Lightning instead derives its penalty data from revocation secrets, and needs no per-state pre-signed transactions.

Alternative: revocation keys. ELSA could do the same. Give the lock of state n on Alice's side the spend key $\mathsf{ak}_{A,n} = X_B + T_{A,n}$, where X_B is Bob's key, and make Alice's share of it her revocation secret $t_{A,n}$.

- *Before revocation.* Signing needs both shares, so CleanRedeem is pre-signed jointly with 2-of-2 FROST on these shares.
- *After revocation.* Once Alice reveals $t_{A,n}$, Bob holds the whole key, and he builds and signs the penalty himself when a breach happens. He only needs to store the preimages, which a seed derivation keeps logarithmic.
- *Proof of knowledge.* Alice's lock must come with a proof of knowledge of $t_{A,n}$. Otherwise she could choose $T_{A,n} = [y]\mathcal{G} - X_B$ for a y she knows, hold the whole key herself, and spend the lock before Bob can punish. The hash-lock proof $\pi_{A,n}$ already proves knowledge of $t_{A,n}$.

This replaces the hash-locked WitnessRedeem pre-signature, and with it ELSA's dependence on hashFROST's unforgeability (P2) for revocation. The cost is a fresh spend key, with its own viewing key, per lock. We have not adopted it, but it is the more storage-efficient design.

8 Deployment notes

Lifetime. Every pre-signed transaction is valid only in its network-upgrade epoch, so a channel's deadline D, its h^{clean} and its backstop must all fall before the next upgrade. A long-lived channel must close, or re-sign its latest state, before then. Revoked states need not be re-signed, since their transactions die with the epoch too.

Relative timelocks. If a consensus change made `lock_time` bind for shielded-only transactions, the state outputs could go. If it also added relative lock times in the style of BIP 68 [FBDk15], the dispute window could restart at each commitment as in Lightning, and the fixed deadline would disappear. The VCL paper proposes the change [VCL26].

Privacy. Amounts and addresses stay shielded. The state outputs are public, however. They link the funding transaction to the commitment, and the commitment to the transaction that spends its lock, and each reveals a 2-of-2 script when spent.

9 Related work

Channels. Lightning [PD16] uses revocable commitments with relative timelocks. Decker and Wattenhofer's duplex channels [DW15] use only absolute timelocks and bound the number of updates by decrementing them; ELSA revokes instead, so updates are unbounded and only the lifetime is fixed. Generalized channels [AEE+21] build Lightning-style channels from adaptor signatures and minimal scripts.

Chains without scripts. PayMo [TMSS20] builds channels for Monero from verifiable timed signatures. Bolt [GM17] gives anonymous payment channels for Zcash-like currencies, with stronger unlinkability between a channel's payments than ELSA aims for. ELSA keeps Lightning's structure and works with Zcash's existing v6 transaction format, through VCLs [VCL26], whose transaction prediction comes from Skyhook [Vis26].

10 Conclusion

VCLs are enough to build a Lightning-style channel on a pool without scripts, at the price of a fixed channel lifetime, a public state output, and storage linear in the number of updates. The next steps are:

- a full proof of the VCL paper's P2, on which balance security rests;
- revocation keys (Section 7), which would make storage logarithmic and remove that dependence for revocation;
- routing across channels with point-time-locked contracts built from preFROST. With only absolute timelocks and fixed deadlines, ordering the timeouts hop by hop is the main problem.
- a simulation-based treatment;
- an implementation, with measured proving times and latency.

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 *Advances in Cryptology – ASIACRYPT 2021*, LNCS. Springer, 2021.

[DW15] Christian Decker and Roger Wattenhofer. A fast and scalable payment network with Bitcoin duplex micropayment channels. In *Stabilization, Safety, and Security of Distributed Systems (SSS 2015)*, LNCS. Springer, 2015.

[FBDk15] Mark Friedenbach, BtcDrak, Nicolas Dorier, and kinoshitajona. BIP 68: Relative lock-time using consensus-enforced sequence numbers. `https://github.com/bitcoin/bips/blob/master/bip-0068.mediawiki`, 2015.

[GM17] Matthew Green and Ian Miers. Bolt: Anonymous payment channels for decentralized currencies. In *ACM Conference on Computer and Communications Security (CCS 2017)*, 2017.

[PD16] Joseph Poon and Thaddeus Dryja. The Bitcoin lightning network: Scalable off-chain instant payments. White paper, 2016.

[TMSS20] Sri Aravinda Krishnan Thyagarajan, Giulio Malavolta, Fritz Schmidt, and Dominique Schröder. PayMo: Payment channels for Monero. Cryptology ePrint Archive, Paper 2020/1441, 2020.

[Tuc26] Adam Tucker. Quantum unlinkability for Zcash swaps. Zakura engineering blog, `https://zakura.com/engineering/quantum-unlinkability/`, 2026.

[VCL26] Violet's constrained locks, or: How i learned to stop worrying and love scriptless chains. Companion paper, 2026.

[Vis26] Abhinav Vishnu. Skyhook: Trustless cross-chain value transfer with a fully scriptless pool. Cryptology ePrint Archive, Paper 2026/2210, 2026.

[ZIP] ZIP 32: Shielded hierarchical deterministic wallets. `https://zips.z.cash/zip-0032`.

A Proof of balance security (Theorem E1)

Throughout, "except with negligible probability" is implicit wherever an assumption is used. P is the honest party, Q the adversary's.

Lemma A.1 (What can leave the channel)**.** *Every transaction in the ledger that spends a channel note, a note in a lock L_P^m or L_Q^m, or a state output u_0, u_P^m, u_Q^m, is one of the following:*

- *a transaction of Table 1 for some state;*
- HE*;*
- *a cooperative close or renewal that P signed.*

Moreover, a mined WR_P^m requires that P released $t_{P,m}$.

Proof. All these notes are spent under the channel key. By the Action statement, a spend needs a spend-authorization signature under $\mathsf{rk} = \mathsf{ak} + [\alpha]\mathcal{G}$ for some α. P contributes its share only to the digests of the listed transactions, after the VCL check. By property U of FROST and plain hashFROST, Q cannot produce a valid signature on any other digest. This is exclusivity, V6. The state outputs need P's ECDSA signature, which P gives only for the same transactions.

For WR_P^m, P closed only a pre-signing session, for the adaptor point $T_{P,m}$, which P sampled. Property U says that Q cannot output a valid signature on that digest unless $t_{P,m}$ is known. This uses the extraction rule: P never signs that digest in another session. □

Lemma A.2 (The newest own commitment is never revoked)**.** *If n is the newest state for which P holds a fully signed commitment and CleanRedeem, P has not released $t_{P,n}$.*

Proof. By R1, P releases $t_{P,m}$ only after holding a fully signed C_P^{m+1} and CR_P^{m+1}. Then the newest state would be at least $m + 1 > m$. □

Lemma A.3 (Which states each party can publish)**.** *Let f be the newest state final for P.*

1. *P's newest fully signed own commitment is for a state $n \in \{f, f+1\}$, and $n = f$ unless an update to $f + 1$ is in progress.*
2. *Every unrevoked commitment that Q holds fully signed is for a state in $\{f, f+1\}$, and only f unless an update to $f + 1$ is in progress.*

Proof. By R3, P has signed transactions only of states up to $f + 1$, and of $f + 1$ only while that update is in progress. So Q's fully signed commitments are for states at most $f + 1$. Since f is final, P has received $t_{Q,f-1}$. Each earlier update was final for P before the next began, by R3 again, so P also holds $t_{Q,m}$ for every $m \le f - 1$. Those commitments are therefore revoked. The bound on P's own commitments is the same argument. □

Lemma A.4 (Resolution by D)**.** *By height D, u_0 has been spent, by a commitment, a cooperative close or a renewal.*

Proof. Suppose u_0 is unspent at $D - \Delta$. By L2, P broadcasts a commitment at some height $\eta \leq D - \Delta$. The commitment is valid at every height up to D (`expiry_height` D, V8). By synchrony it is mined by $\eta + \Delta \leq D$, unless another transaction spending u_0 is mined first. By Lemma A.1, such a transaction is a commitment, a close, a renewal or HE. HE is valid only at heights $> D$ (V8), so it is none of the first spends before D. □

Lemma A.5 (Timing)**.** *Let $K = W - \Delta - 1$ with $\Delta < W - 1$, and $h^{\text{clean}} = D + W$.*

1. *If a transaction is mined at a height $b \leq D$, P sees it by $b + K$. A transaction P broadcasts then is mined by $b + K + \Delta = b + W - 1 \leq h^{\text{clean}} - 1$.*
2. *A transaction that P broadcasts within K heights after h^{clean} is mined by $h^{\text{clean}} + K + \Delta = h^{\text{clean}} + W - 1 < U$.*

Proof. Arithmetic; these are `penalty_in_time` and `clean_in_time` in the Lean development of the VCL paper. □

Proof of Theorem 4.1. Let f be as in the guarantee, and let n be P's newest fully signed own commitment. By Lemma A.3, $n \in \{f, f + 1\}$, so $V_P^n \geq g_P$. Consider the first transaction spending u_0. By Lemma A.4 it is mined by D; by Lemma A.1 it falls in one of four cases.

Case 1: a cooperative close or a renewal. P signed it, and by its policy it pays P at least g_P. By payment binding (V3), the mined transaction pays exactly the outputs P checked. For a renewal, the new channel's first state was signed before the renewal, so the theorem applies to the new channel with the same guarantee.

Case 2: P's own commitment C_P^m. P broadcasts only its newest commitment, so $m = n$. By V3 it pays V_Q^n to Q and V_P^n to the lock L_P^n, together with the state output u_P^n. By Lemma A.1, the note in L_P^n can leave only through CR_P^n, or through WR_P^n if $t_{P,n}$ was released. By Lemma A.2 it was not. CR_P^n is valid from h^{clean} on (V8), and WR_P^n is invalid from then on, so nothing conflicts with it. By L1, P broadcasts CR_P^n within K heights after h^{clean}, and by Lemma A.5 it is mined by $h^{\text{clean}} + W - 1 < U$. P receives $V_P^n \geq g_P$.

Case 3: an unrevoked commitment C_Q^m of Q. By Lemma A.3, $m \in \{f, f + 1\}$, and $m = f$ unless an update to $f + 1$ is in progress; in either case $V_P^m \geq g_P$. By V3, C_Q^m pays V_P^m directly to P's payout address, and P checked that it can decrypt and spend that output. So P receives $V_P^m \geq g_P$, mined with the commitment by D.

Case 4: a revoked commitment C_Q^m of Q. P holds $t_{Q,m}$, the pre-signature for WR_Q^m and its completion data, which P built. P also holds Q's ECDSA signature for u_Q^m, and the opening of the note in L_Q^m, which it checked by trial decryption.

- The commitment is mined at some $b \leq D$, and by Lemma A.5 P sees it by $b + K$.
- P completes WR_Q^m with $t_{Q,m}$ and broadcasts it. It is valid until $h^{\text{clean}} - 1$, and by Lemma A.5 it is mined by $b + W - 1 \leq h^{\text{clean}} - 1$.
- It can be preempted only by CR_Q^m, which is invalid before h^{clean}, or by a transaction P signed (Lemma A.1).

P receives V_Q^m from WR_Q^m and V_P^m directly from the commitment: the whole channel, which is at least g_P.

In every case P's payouts are mined by $h^{\text{clean}} + W - 1$. P can spend them, because it built or checked each one, recoverability included. □

B Claim status

The status labels are those of the VCL paper [VCL26].

Claim	Statement	Status
E1	balance security under synchrony	proved (Appendix A), assuming the VCL paper's P2 (a proof plan only there) and the VCL results below
E1 timing	the dispute and CleanRedeem deadlines, resolution by D, disjoint windows	machine-checked (`penalty_in_time`, `clean_in_time`, `commitment_by_deadline`, `wr_cr_disjoint`, `commit_he_disjoint`)
E2	independence of per-state viewing keys	proved, assuming $\mathsf{PRF}^{\mathsf{expand}}$ is a PRF
W1	watchtower privacy	proved in the random-oracle model, assuming IND$-CPA encryption
—	on-chain leakage	qualitative analysis (Section 6)
—	fees, rounds, storage	estimates from ZIP 317 as we read it; proving times not measured (Section 7)
V3, V5, V6, V8	payment binding, mutual exclusion, exclusivity, timelock windows	from the VCL paper [VCL26]; V5 and V8 are machine-checked there
P1, H4	hashFROST adaptation and extraction, the 2-bit trim	from the VCL paper [VCL26]; machine-checked there
P2	unforgeability of preFROST and plain hashFROST; ELSA uses the hashFROST part	proof plan only in the VCL paper [VCL26]. There is no black-box reduction to FROST, so a full proof must redo FROST's unforgeability proof with the adaptor point