Thunderbolt: Fast Asynchronous Off-Chain Bitcoin Transfers

Hongbo Wen
hongbowen@ucsb.edu
University of California, Santa Barbara
Santa Barbara, CA, USA

Hanzhi Liu
hanzhi@ucsb.edu
University of California, Santa Barbara
Santa Barbara, CA, USA

Yanju Chen
yanju@ucsd.edu
University of California, San Diego
San Diego, CA, USA

Jingyu Ke
windocotber@sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China

Dahlia Malkhi
dahliamalkhi@ucsb.edu
University of California, Santa Barbara
Santa Barbara, CA, USA

Yu Feng
yufeng@cs.ucsb.edu
University of California, Santa Barbara
Santa Barbara, CA, USA

Abstract

We present Thunderbolt, an off-chain protocol that transfers Bitcoin UTXO ownership with seconds-scale latency, requires no channel graph, no routing, and no liquidity rebalancing, and lets the recipient be offline at the time of transfer. A single UTXO is locked once on-chain under a fixed public key jointly held by the current holder and a threshold committee; ownership then passes through an unbounded sequence of holders; each transfer is a purely off-chain, asynchronous operation whose on-chain cost is zero. The chain sees exactly two transactions regardless of how many transfers occur.

The core invariant is an algebraic cancellation: at each transfer the recipient's fresh secret is added to the holder's share and subtracted from the committee's share, so both shares rotate while the on-chain key stays fixed. To enforce this, the recipient publishes an invoice to a shared append-only ledger (the *Thunderbolt Ledger*): a public commitment, an encrypted copy for himself, and an encrypted copy for the committee, together with a zero-knowledge proof that all three encode the same fresh secret. The sender fetches the invoice, verifies the proof, homomorphically folds her secret into the recipient's ciphertext to produce a new ownership credential, and publishes the result with a second zero-knowledge proof. Both proofs use a single Sigma-protocol response to force the same witness across elliptic-curve and Paillier verification equations, requiring no trusted setup. The committee operates under a standard (t, n)-threshold honest-majority assumption: at most $t-1$ of n members may be corrupted. The recipient decrypts at any later time; the committee subtracts the fresh secret from its share. By binding each transfer to a distinct fresh secret and context identifier, multiple UTXOs can be transferred independently in parallel.

On our benchmark machine, a complete off-chain transfer finishes in 1022 ms with a combined proof size of 3.8 KB. General-purpose SNARK (Succinct Non-interactive Argument of Knowledge) backends are orders of magnitude slower on the same relations: even a reduced-parameter instantiation already exceeds our native proving time by two orders of magnitude, and a faithful realization at the deployed 3072-bit Paillier modulus exceeds the memory budget of consumer hardware.

CCS Concepts

• **Security and privacy → Cryptography**; **Distributed systems security**.

Keywords

Bitcoin, Off-Chain Transfers, Verifiable Encryption, Paillier Cryptosystem, Zero-Knowledge Proofs, Payment Channels

ACM Reference Format:
Hongbo Wen, Hanzhi Liu, Yanju Chen, Jingyu Ke, Dahlia Malkhi, and Yu Feng. 2026. Thunderbolt: Fast Asynchronous Off-Chain Bitcoin Transfers. In *Proceedings of Proceedings of the 2026 ACM Conference on Computer and Communications Security (CCS '26).* ACM, New York, NY, USA, 19 pages. https://doi.org/10.1145/XXXXXXX.XXXXXXX

1 Introduction

Bitcoin [46] intentionally constrains throughput and expressiveness at the base layer: blocks are small, scripts are limited, and every state change pays the cost of global verification. This conservatism underpins Bitcoin's credibility as a settlement layer, but makes it unsuitable as a fast, low-cost medium for everyday value transfer. The dominant response has been the *payment channel*: two (or more) parties lock funds in an on-chain UTXO (unspent transaction output), exchange signed off-chain updates, and fall back to the chain only to open or close.

The Lightning Network. The Lightning Network [51] scales payment channels to a routed graph of pre-funded, bidirectionally-online channels and is today the most widely deployed Bitcoin layer-2. However, Lightning optimizes for multi-hop routing between fixed endpoints, and some scenarios call for *asynchronous* transfers (sender and receiver need not be online simultaneously), *route-free* operation (no capacity-sufficient path is needed), and *sequential ownership migration* (a fixed-denomination UTXO changes

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CCS '26, The Hague, The Netherlands
© 2026 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-XXXX-X/2026/11
https://doi.org/10.1145/XXXXXXX.XXXXXXX

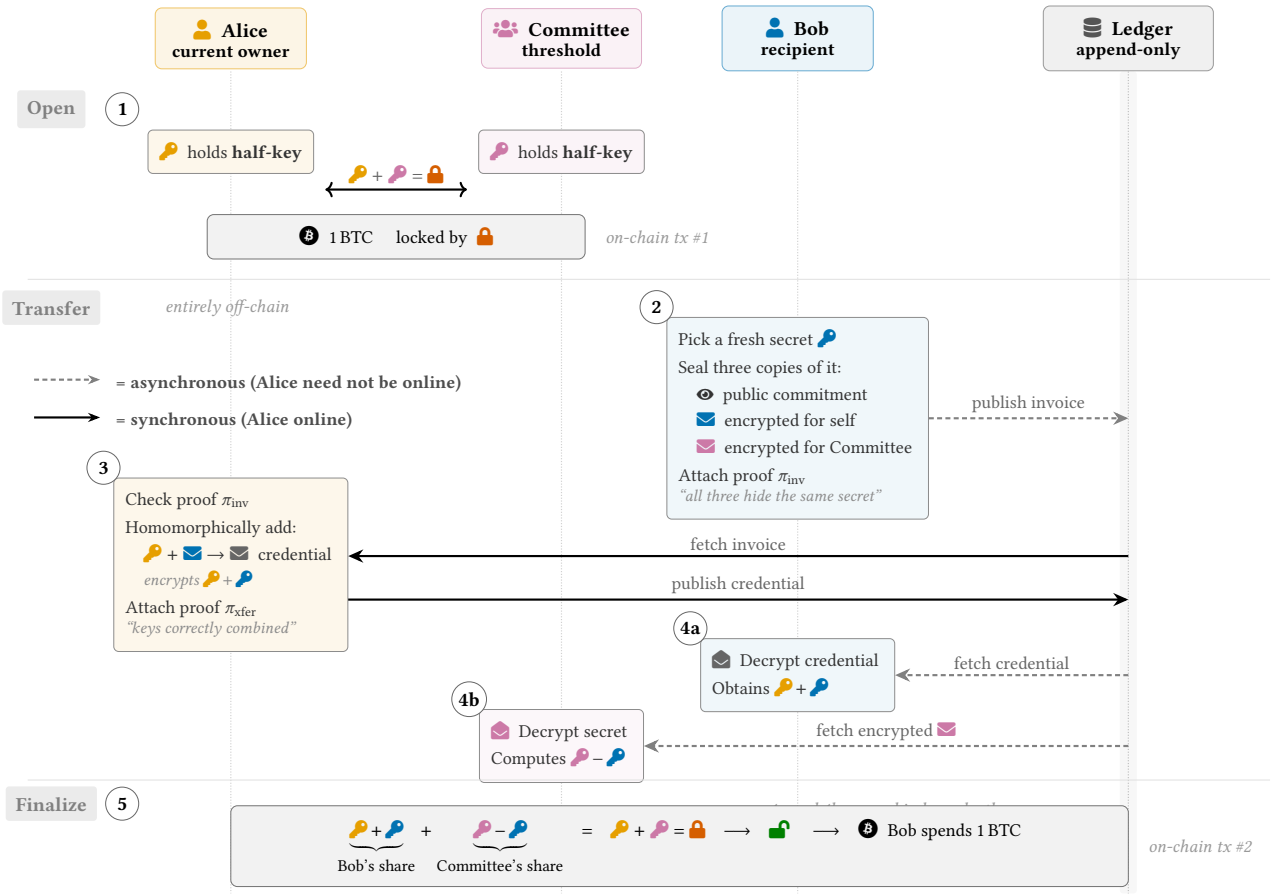

Figure 1: End-to-end overview of Thunderbolt: transferring 1 BTC from Alice to Bob via the Thunderbolt Ledger. ① Alice and the Committee lock a UTXO under their combined key. ② Bob creates an invoice: three sealed copies of a fresh secret plus proof π_{inv}. ③ Alice verifies the invoice, folds her secret into Bob's ciphertext, and publishes a credential with proof π_{xfer}. ④a Bob decrypts the credential to obtain the new holder secret. ④b The Committee decrypts and subtracts the fresh secret from its share. ⑤ Bob and the Committee co-sign to spend the UTXO on-chain. The complete protocol specification is given in Section 4.

hands through an unbounded sequence of holders), as in payroll batches, cold-wallet receipts, custody chains, and intermittently-online devices; Section 5.4 revisits these workloads once off-chain splitting lifts the fixed-denomination restriction.

Statechains and L2 Protocols. Somsen's statechain protocol [57] takes a different approach: instead of routing payments across channels, it transfers entire UTXOs off-chain via a co-signing entity that shares the lock key with the current owner. At each transfer, the entity must provably delete its old co-signing key and generate a new one with the next owner. However, key deletion is fundamentally unverifiable: no cryptographic mechanism can prove that a party has forgotten a secret. Mercury Layer [19] extends statechains with blinded signing but retains the single-entity trust model and the key-deletion assumption. Spark [40] extends the same statechain lineage with threshold operators and leaf splitting, whereas Ark [1, 37] uses an architecturally distinct, operator-coordinated virtual UTXO (VTXO) design; Section 7 compares both with Thunderbolt in detail.

The role of recipient randomness. Across these off-chain transfer designs, a decisive question is whether and how the recipient introduces a fresh random value during the transfer. In Lightning, the recipient chooses a payment preimage and publishes its hash in an invoice; the sender's payment is conditional on learning the preimage, but the preimage serves only as a release condition and does not enter the lock key itself. Statechain recipients also contribute fresh key material, but security against a past holder ultimately relies on the co-signing entity deleting its obsolete share.

Fixed-key share rotation. Thunderbolt builds on the additive Schnorr key-sharing structure used by statechains [54], but replaces key deletion with a publicly verifiable rotation under a threshold assumption. Suppose the recipient chooses a fresh secret r and the protocol arranges for r to be *added* to one share and *subtracted* from the other. Both shares rotate, but their sum—and therefore the public key—stays fixed. The on-chain UTXO never needs to move, yet ownership has changed hands. The remaining challenge is binding r consistently across all parties without revealing it: if a zero-knowledge proof can enforce this consistency across an

elliptic-curve commitment and Paillier ciphertexts, the transfer is both verifiable and private, requiring no key deletion, no routing, and no simultaneous online requirement.

Thunderbolt. This paper presents Thunderbolt, a protocol that realizes this idea. A Bitcoin UTXO is locked once under a fixed public key jointly held by a holder and a threshold committee of n members, of whom at most $t-1$ may be corrupted (a standard (t, n)-threshold honest-majority assumption). At each transfer the recipient picks a fresh secret r_s, publishes an *invoice* (a public commitment, two Paillier ciphertexts, and a zero-knowledge proof of their consistency) to a shared append-only ledger (the *Thunderbolt Ledger*). The sender fetches the invoice, homomorphically folds her secret into the recipient's ciphertext, and publishes a certified credential back to the ledger. The recipient decrypts at any later time; the threshold committee subtracts r_s from its share. The on-chain key never changes, and the transfer is *recursive*: the new holder can immediately serve as the next sender. Figure 1 gives an end-to-end overview of the protocol; the full specification is given in Section 4.

Cryptographic approach. The core challenge is ensuring that the same r_s is consistently bound across three different algebraic structures, an elliptic-curve point and two Paillier ciphertexts, without revealing r_s to anyone other than its creator. The Paillier cryptosystem [48] is essential here: its additive homomorphism operates over integers rather than group elements, so the same integer witness can appear in both an elliptic-curve discrete-log relation and a Paillier ciphertext equation. We solve the cross-domain binding with two Sigma-protocol NIZK proofs that share a common technique: a single Sigma response z forces the same integer to appear in both an EC verification equation and a Paillier verification equation. The first proof (the Invoice Consistency Proof) binds r_s across all three invoice objects; the second (the Transfer Consistency Proof) binds the sender's secret to the anchor and the new credential. Both instantiate Camenisch–Shoup verifiable encryption of discrete logarithms [13]. The threshold committee is realized via Simple VESS [56], distributing trust across n members under the same (t, n)-threshold honest-majority assumption. We compare this approach against general-purpose SNARK backends in Section 6.2.

Contributions. The main contributions of this paper are:

(1) We introduce Thunderbolt, an off-chain protocol that enables asynchronous, recursive transfer of Bitcoin UTXO ownership under a fixed on-chain lock key, mediated by a shared append-only ledger with no simultaneous online requirements.
(2) We construct two compact Sigma-protocol NIZKs (the Invoice and Transfer Consistency Proofs) that bind a single integer witness across elliptic-curve and Paillier verification equations via Camenisch–Shoup verifiable encryption, and show that the threshold committee can be realized via Simple VESS.
(3) We formalize the security of Thunderbolt in the Universal Composability (UC) framework, defining the Thunderbolt ideal functionality $\mathcal{F}_{\mathsf{TB}}$ as a state machine over Thunderbolt Ledger entries that captures ownership security, fixed-key integrity, and transfer correctness by construction, and proving that the transfer-and-finalization logic UC-realizes it via a straight-line simulator.
(4) We evaluate Thunderbolt, achieving seconds-scale transfer latency with 3.8 KB proof size, and show that even a reduced-parameter 256-bit SP1 instantiation is over two orders of magnitude slower to prove.

Roadmap. Section 2 recalls background primitives (notation is summarized in Appendix D). Section 3 defines the threat model. Section 4 presents the protocol, including the design rationale, the Invoice and Transfer Consistency Proofs, and the channel lifecycle. Section 5 formalizes security via the UC ideal functionality $\mathcal{F}_{\mathsf{TB}}$ and analyzes attack vectors. Section 6 addresses three research questions covering transfer latency and scalability (RQ1), comparison with SNARK backends (RQ2), and threshold-committee performance (RQ3). Section 7 surveys related work and Section 8 concludes.

2 Background

This section reviews the cryptographic building blocks used by Thunderbolt. The on-chain lock key and all off-chain anchors live in an *elliptic-curve group* (Section 2.1), and the lock key is additively split between a holder and a threshold committee via the algebraic structure of *Schnorr signatures* (Section 2.2). Each transfer rotates both shares homomorphically through the *Paillier cryptosystem* (Section 2.3), and correctness of each rotation is certified by *Sigma-protocol* NIZKs compiled via Fiat–Shamir (Section 2.4). The core proof technique that binds an elliptic-curve commitment to a Paillier ciphertext under a single witness is *verifiable encryption of discrete logarithms* (VEDL, Section 2.5). Finally, the threshold committee is realized via *verifiable encrypted secret sharing* (VESS, Section 2.6).

2.1 Elliptic-Curve Groups

Let $\mathbb{G}$ be an elliptic-curve group of prime order q. We fix two independent generators of $\mathbb{G}$: the standard secp256k1 base point G, used for the on-chain lock key $P = (x_s + y_s)\,G$, and a second nothing-up-my-sleeve (NUMS) generator H (a point derived by a deterministic, publicly auditable process so that no one knows its discrete log with respect to G), used for all holder anchors $X_s = x_s H$ and invoice deltas $D_s = r_s H$. Independence of G and H (i.e., the discrete log of H with respect to G is unknown) is required so that the public anchor $X_s = x_s H$ does not reveal $x_s G$, which would allow anyone to compute $y_s G = P - x_s G$ and reduce the security of the committee share to a bare discrete-log instance. A *discrete-log pair* is a pair (x, X) with $X = x H$ for some scalar $x \in \mathbb{Z}_q$. Given only X and H, no efficient algorithm can recover x; this is the *elliptic-curve discrete-logarithm problem* (ECDLP), on which the security of Bitcoin and of our proofs rests.

2.2 Schnorr Signatures and Additive Key Splitting

A Schnorr signature [54] over a message m under public key $P = x\,G$ is a pair (R, σ) where $R = k\,G$ for a random nonce k, and $\sigma = k + e\,x$ with challenge $e = \mathsf{Hash}(R, P, m)$. Verification checks $\sigma\,G \stackrel{?}{=} R + e\,P$. Bitcoin's Taproot upgrade [62] adopts Schnorr signatures as the native signature scheme.

A key property of Schnorr signatures is that the signing key can be *additively split*: two parties holding shares x and y with $P = (x+y)\,G$ can jointly produce a valid signature under P without either party learning the other's share. Each party samples a nonce (k_1, k_2), publishes its nonce point ($R_1 = k_1\,G$, $R_2 = k_2\,G$), computes the shared challenge $e = \mathsf{Hash}(R_1 + R_2,\ P,\ m)$, and responds with a partial signature ($\sigma_1 = k_1 + e\,x$, $\sigma_2 = k_2 + e\,y$). The combined signature $(R_1 + R_2,\ \sigma_1 + \sigma_2)$ is valid under P [39]. This additive structure is the foundation of Thunderbolt's key-sharing model: the lock key P is split between a holder and a threshold committee, and ownership transfer rotates both shares without changing P.

2.3 The Paillier Cryptosystem

The Paillier cryptosystem [48] is an asymmetric encryption scheme whose plaintext space is $\mathbb{Z}_N$ for an RSA modulus $N = p_1 p_2$ (two distinct primes). The public key is (N, g) with $g \in \mathbb{Z}^*_{N^2}$ of order divisible by N. Encryption of a message $m \in \mathbb{Z}_N$ with randomness $\rho \in \mathbb{Z}^*_N$ is

$$\mathrm{Enc}_{pk}(m;\rho) = g^m \rho^N \bmod N^2.$$

Decryption uses the factorization of N. The scheme is semantically secure under the *Decisional Composite Residuosity* (DCR) assumption.

Additive homomorphism. Paillier is *additively* homomorphic: given ciphertexts $c_1 = \mathrm{Enc}(m_1)$ and $c_2 = \mathrm{Enc}(m_2)$, the product

$$c_1 \cdot c_2 \bmod N^2 = \mathrm{Enc}(m_1 + m_2)$$

is a valid encryption of $m_1 + m_2 \bmod N$. We write this operation as $c_1 \oplus c_2$, and the corresponding "ciphertext subtraction" as $c_1 \ominus c_2 := c_1 \cdot c_2^{-1} \bmod N^2$. This additive homomorphism is the reason Thunderbolt can compose a fresh holder ciphertext without decrypting the invoice ciphertext.

2.4 Sigma Protocols and Fiat–Shamir

A *Sigma protocol* [20, 21] is a three-move honest-verifier zero-knowledge proof: the prover sends commitment A, receives challenge c, responds with z; the verifier checks a public equation on (A, c, z). The *Fiat–Shamir heuristic* [30] compiles it into a non-interactive zero-knowledge (NIZK) proof in the random-oracle model by setting $c = \mathsf{Hash}(\text{statement}, A)$; soundness follows from the forking lemma [50].

2.5 Verifiable Encryption of Discrete Logarithms

Camenisch and Shoup [13] give an efficient protocol for *verifiable encryption of discrete logarithms*: given a discrete-log pair $(m, M = mH)$ and a Paillier ciphertext $ct = \mathrm{Enc}_{pk}(m)$, the prover produces a Sigma-protocol proof that ct encrypts the same integer whose discrete log is M. The proof relation is

$$\mathcal{R}_{\mathrm{VE}} = \{(M, ct;\ m, \rho) : M = mH \ \wedge\ ct = g^m \rho^N \bmod N^2\}.$$

The key technique is a *shared response*: the prover commits to a single random nonce α and, after the Fiat–Shamir challenge c is derived from *all* statement components, responds with $z = \alpha + c \cdot m$. This single z must simultaneously satisfy the elliptic-curve equation ($zH \stackrel{?}{=} A_{\mathrm{EC}} + cM$) and the Paillier equation ($g^z \tau^N \stackrel{?}{=} A_{\mathrm{Pai}} \cdot ct^c$). If the integer committed in the ciphertext differed from the discrete log of M, the two equations would be mutually inconsistent for the same z. This "shared-z" construction is the technical core of our proofs.

2.6 Verifiable Encrypted Secret Sharing (VESS)

Shoup's *Simple VESS* [56] provides a mechanism for verifiably encrypting a Shamir secret sharing [55] to a committee of n shareholders, building on the line of work initiated by Feldman [29], Pedersen [49], and the distributed key-generation protocols of [32]. The sharing party (the "dealer") publishes encrypted shares $(\hat{s}_1, \ldots, \hat{s}_n)$ together with a proof that there exists a unique polynomial of degree at most $t-1$ whose evaluations are the plaintexts of these ciphertexts. Any observer can verify the consistency of the encrypted shares without decrypting them. If at least t of the n shareholders are honest, they can collectively reconstruct the shared secret.

In Thunderbolt, VESS is used when the Committee is instantiated as a threshold committee rather than a single actor. The committee-side fresh secret r_s is shared via a VESS protocol so that the compensating update $y_{s+1} = y_s - r_s$ is performed collectively, without any single committee member learning y_s or r_s in the clear. This modular substitution replaces the single-committee trust assumption with a standard (t, n)-threshold honest-majority assumption, leaving the per-step NIZK proofs unchanged.

3 Threat Model

A Thunderbolt channel involves three roles (a sequence of holders, a threshold committee, and the Bitcoin Chain as a passive settlement layer) communicating through a shared append-only ledger (the *Thunderbolt Ledger*), all operating under an additive key-sharing invariant. We specify the participants and trust model in Section 3.1, and the Thunderbolt Ledger and security properties in Section 3.2.

3.1 Parties and Trust

A Thunderbolt channel has three participant roles:

- The **Holder** (initially H_0) is the party that currently owns the UTXO off-chain. Over the lifetime of the channel, the holder role is handed from one user to the next: $\mathsf{H}_0 \to \mathsf{H}_1 \to \cdots$, where each H_s is an independent user.
- The **Threshold committee** $\{\mathsf{C}_1, \ldots, \mathsf{C}_n\}$ maintains the complementary share of the lock key under a standard (t, n)-threshold honest-majority assumption: at most $t-1$ of the n members may be corrupted. The committee never owns the UTXO and never learns the holder secret; its sole role is to preserve the fixed-key invariant via the compensating update.
- The **Bitcoin Chain** plays the role of a passive verifier: it sees exactly two transactions over the channel's lifetime (Setup and Finalize), regardless of how many holder transitions occur in between.

We consider a static-corruption adversary $\mathcal{A}$ that may corrupt any subset of holders (past or future) and at most $t-1$ of the n committee members before the channel opens. The adversary is probabilistic polynomial-time (PPT) and interacts with the protocol in the random-oracle model. This trust model is identical to that of standard threshold-signing deployments (FROST [39], ROAST [53]).

3.2 The Thunderbolt Ledger

All off-chain communication is mediated by the **Thunderbolt Ledger**, a shared append-only bulletin board visible to all participants. Each UTXO channel is independently bound to the ledger via a distinct context identifier, so multiple UTXOs can be transferred in parallel without interference. A message affects the channel state only after it has been committed by the ledger. For each context identifier ctx_id, the ledger provides three guarantees. **(G1) Append-only persistence:** once an entry is committed, it cannot be removed or modified. **(G2) Per-channel ordering:** all participants observe the same total order of committed entries for that channel. **(G3) Data availability and inclusion:** every valid entry submitted by an honest participant is eventually committed, and every committed entry is eventually retrievable by every participant. No ordering is required across different channels. Ledger entries are authenticated by signatures and NIZKs, and the ledger holds no secret key.

A practical instantiation is a permissioned blockchain whose consensus nodes are the committee members, provided that its consensus protocol guarantees G1–G3 under the assumed corruption bound. More generally, a smart-contract log, federated bulletin board, or append-only service may instantiate the Thunderbolt Ledger if it provides these guarantees.

Under these guarantees, the Thunderbolt protocol aims to provide four security properties, the first three concerning safety and the fourth liveness:

(1) **Ownership security:** Only the current holder, the party who knows the current holder secret x_s, can authorize a transfer or participate in channel finalization.
(2) **Fixed-key integrity:** The on-chain lock key P remains unchanged throughout the channel's lifetime, regardless of how many off-chain transfers occur.
(3) **Transfer correctness:** After a valid transfer from H_s to H_{s+1}, the new holder obtains a valid holder secret x_{s+1} and the committee's share y_{s+1} is updated consistently with the fixed-key invariant.
(4) **Liveness:** The protocol guarantees termination (transfers and finalization complete) provided at least t committee members are honest and responsive. When individual members become unresponsive, committee rotation via proactive resharing [56] preserves this condition without on-chain interaction.

4 The Thunderbolt Protocol

This section presents the Thunderbolt protocol in full. We begin with the design rationale (Section 4.1), explaining the role of the Thunderbolt Ledger, the choice of VEDL for cross-domain binding, and the necessity of the VEDL+VESS composition. We then fix notation and describe the complete protocol flow (Section 4.2), followed by the two core zero-knowledge proofs that enforce per-step correctness: the Invoice Consistency Proof (Section 4.3) and the Transfer Consistency Proof (Section 4.4). Their joint consequence (Section 4.5) establishes the fixed-key invariant. Finally, we describe how the threshold committee is realized via VESS (Section 4.6).

4.1 Design Rationale

The introduction motivates the key-splitting insight and the two-phase transfer flow. This subsection explains the three design decisions that make the construction concrete: why the protocol needs a persistent ledger with per-channel binding, why VEDL is the right cross-domain proof technique, and why neither VEDL nor VESS alone suffices.

The Thunderbolt Ledger and context identifiers. Because neither the sender nor the recipient need be online at the same time, the protocol requires a persistent medium where invoices and credentials can be deposited and later retrieved. The *Thunderbolt Ledger* (Section 3.2) serves this role. Each UTXO channel is bound to the ledger via a distinct *context identifier* ctx_id, concretely the UTXO outpoint (txid:vout) fixed at channel open. Within a channel, each invoice carries a fresh *invoice identifier* inv_id chosen by the recipient; the ledger enforces uniqueness by rejecting duplicate inv_id values under the same ctx_id. Both identifiers enter every Fiat–Shamir challenge hash ($c = \mathsf{Hash}(\ldots, \mathsf{ctx_id}, \mathsf{inv_id}, \ldots)$), binding each proof to a specific channel and a specific transfer step: ctx_id prevents cross-channel replay, and inv_id prevents within-channel replay.

VEDL for cross-domain binding. The core challenge is binding the recipient's fresh secret r_s consistently across three distinct algebraic structures: an elliptic-curve point and two Paillier ciphertexts. A generic Sigma protocol could prove each relation independently, but independent proofs would not prevent the prover from using different values in different domains. Verifiable encryption of discrete logarithms [13] (Section 2.5) solves this with a single shared Sigma response z that must simultaneously satisfy both an EC and a Paillier verification equation, forcing the same integer into both. We instantiate VEDL twice: the Invoice Consistency Proof extends it to a multi-receiver setting (one EC commitment and two Paillier ciphertexts bound by a single z); the Transfer Consistency Proof uses a standard single-receiver instance.

VESS for threshold trust. VEDL alone concentrates the committee role in a single actor. VESS [56] (Section 2.6) removes this single point of failure: r_s is verifiably encrypted as a (t, n)-Shamir sharing, so that any t honest members can perform $y_{s+1} = y_s - r_s$ collectively, while no coalition of fewer than t learns anything. This substitution is modular: VESS replaces the single ciphertext ct_C with a set of verifiably encrypted shares; the Transfer Consistency Proof is unchanged, and π_{inv} narrows to the two-receiver relation (D_s, ct_B) while a separate VESS consistency proof covers the committee shares.

Necessity of the composition. Neither primitive alone suffices. VEDL without VESS forces a single trusted third party. VESS without VEDL cannot bind the shared secret to the on-chain elliptic-curve anchor. Together, VEDL enforces per-step correctness and VESS distributes trust across a threshold group.

4.2 Protocol Description

Before formalizing the consistency proofs, we describe the complete protocol flow. A Thunderbolt channel passes through three phases: **Open**, **Transfer** (zero or more), and **Finalize**. At every

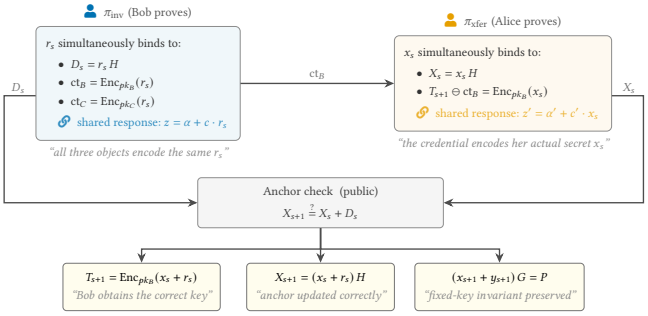

Figure 2: Proof composition for a single transfer step: a shared Sigma response binds each witness, ct_B bridges the two proofs, and the public anchor check yields the three joint consequences.

step s, the protocol maintains a *holder secret* x_s, a *committee secret* y_s, and a *public anchor* $X_s = x_s H$, subject to the fixed-key invariant

$$P = (x_s + y_s) G. \tag{1}$$

Notation. Throughout, we use the notation of Section 2.1: $\mathbb{G}$ is secp256k1 with prime order q, standard base point G (used for the lock key P), and NUMS generator H (used for all anchors and deltas). Let (N_B, g_B) and (N_C, g_C) be Paillier public keys for Bob and the Committee, respectively. All Paillier arithmetic is modulo N_B^2 or N_C^2 as appropriate. All elliptic-curve scalar equalities are taken modulo q. Let κ denote the statistical security parameter (concretely, $\kappa = 128$); we set $\tilde{q} = q^2 \cdot 2^\kappa$ as the masking range for Sigma-protocol commitment scalars, which ensures statistical zero-knowledge with distance at most $2^{-\kappa}$.

Running example. To ground the notation: suppose at step s=2 Alice holds secret x_2 with public anchor $X_2 = x_2 H$, and the committee holds y_2 such that $(x_2 + y_2) G = P$. Bob picks a fresh r_2, publishes three invoice objects ($D_2 = r_2 H$, $\text{ct}_B = \text{Enc}_{pk_B}(r_2)$, $\text{ct}_C = \text{Enc}_{pk_C}(r_2)$) plus proof π_{inv}. Alice folds: $T_3 = \text{Enc}_{pk_B}(x_2) \oplus \text{ct}_B = \text{Enc}_{pk_B}(x_2 + r_2)$, updates $X_3 = X_2 + D_2$, and publishes π_{xfer}. Bob decrypts T_3 to obtain $x_3 = x_2 + r_2$; the committee computes $y_3 = y_2 - r_2$. The lock key P never changes.

Open (on-chain). The initial holder H_0 and the Committee jointly derive $P = (x_0 + y_0) G$ and fund a Taproot UTXO [62] locked to P; in the threshold setting, at least t committee members must acknowledge the opening. The initial anchor is $X_0 = x_0 H$ and the initial holder ciphertext is $T_0 = \text{Enc}_{pk_{\mathsf{H}_0}}(x_0)$.

Transfer (off-chain). Figure 2 illustrates the proof composition that enforces the fixed-key invariant across the two phases. Each ownership transition is a purely off-chain operation mediated by the Thunderbolt Ledger. Denote the current holder by Alice and the next recipient by Bob. A single transfer from Alice to Bob proceeds in two phases:

(1) **Invoice phase (Bob).** Bob chooses a fresh secret $r_s \in \mathbb{Z}_q$ and publishes the following to the Thunderbolt Ledger:
- a public delta point $D_s = r_s H$,
- a ciphertext to himself, $\text{ct}_B = \text{Enc}_{pk_B}(r_s;\ \rho_B)$,
- a ciphertext to the Committee, $\text{ct}_C = \text{Enc}_{pk_C}(r_s;\ \rho_C)$,
- a proof π_{inv} (the Invoice Consistency Proof) that all three objects encode the same r_s.

(2) **Transfer phase (Alice).** Alice fetches the invoice from the Thunderbolt Ledger, verifies π_{inv}, forms the next holder ciphertext

$$T_{s+1} = \text{Enc}_{pk_B}(x_s;\ \eta_s) \oplus \text{ct}_B,$$

updates the public anchor to $X_{s+1} = X_s + D_s$, and publishes $(T_{s+1}, X_{s+1}, \pi_{\text{xfer}})$ to the Thunderbolt Ledger.

After the transfer, Bob fetches T_{s+1} from the ledger, decrypts it to obtain $x_s + r_s$, and sets $x_{s+1} \equiv x_s + r_s \pmod q$. The Committee fetches and verifies both π_{inv} and π_{xfer} against the current anchor X_s, then recovers r_s from ct_C and updates $y_{s+1} = y_s - r_s$. No step requires Alice and Bob to be online simultaneously or touches the chain. The fixed-key invariant (1) is preserved:

$$(x_{s+1} + y_{s+1}) G = (x_s + r_s + y_s - r_s) G = (x_s + y_s) G = P.$$

Finalize (on-chain). To close the channel, the holder proves knowledge of x_s matching the current anchor X_s. This is expressed by the ownership relation $\mathcal{R}_{\text{own}} := \{(X; x) : X = xH\}$, where X is the public statement and x is the secret witness. The holder posts the closing request to the Thunderbolt Ledger, where at least t committee members acknowledge it; the holder and the Committee then jointly sign the closing transaction under P [39]. The chain sees exactly two transactions regardless of how many transfers occurred.

By Theorem 4.5 and induction on s, the protocol preserves three invariants at every step: $(x_s + y_s) G = P$ (fixed key), $X_s = x_s H$ (anchor correctness), and $T_s = \text{Enc}_{pk_{\mathsf{H}_s}}(x_s)$ (ciphertext correctness).

Security assumptions. Both proofs rely on ECDLP hardness in $\mathbb{G}$, the DCR assumption [48] for Paillier, and the random-oracle model [7] for Fiat–Shamir. Under these, both are complete, specially sound, and HVZK; soundness follows from the forking lemma [50].

4.3 Invoice Consistency Proof

The invoice phase requires Bob to publish three objects encoding the same fresh secret r_s: an elliptic-curve commitment D_s, a Paillier ciphertext ct_B for himself, and a Paillier ciphertext ct_C for the Committee. The Invoice Consistency Proof π_{inv} is a NIZK that forces all three to encode the same integer, preventing Bob from breaking the fixed-key invariant by committing inconsistent values.

Definition 4.1 (Invoice Consistency Relation). *The relation $\mathcal{R}_{\text{inv}}$ is defined as*

$$\mathcal{R}_{\text{inv}} = \left\{ (D_s, \text{ct}_B, \text{ct}_C;\ r_s, \rho_B, \rho_C) \ : \begin{array}{l} D_s = r_s H, \\ \text{ct}_B = g_B^{r_s} \rho_B^{N_B} \bmod N_B^2, \\ \text{ct}_C = g_C^{r_s} \rho_C^{N_C} \bmod N_C^2 \end{array} \right\}.$$

Sigma protocol. The prover (Bob) samples a commitment scalar $\alpha \leftarrow \mathbb{Z}_{\tilde{q}}$ and two random commitment values $\beta_B \leftarrow \mathbb{Z}_{N_B}^*$, $\beta_C \leftarrow \mathbb{Z}_{N_C}^*$, and computes:

$$A_D = \alpha H,$$
$$A_B = g_B^\alpha \beta_B^{N_B} \bmod N_B^2,$$
$$A_C = g_C^\alpha \beta_C^{N_C} \bmod N_C^2.$$

Fiat–Shamir challenge. The challenge is computed as

$$c = \mathsf{Hash}\big(\texttt{tb/invoice},\ \texttt{ctx_id},\ \texttt{inv_id},\ H,\ pk_B,\ pk_C,\ D_s,\ \mathrm{ct}_B,\ \mathrm{ct}_C,\ A_D,\ A_B,\ A_C\big) \bmod q.$$

Response. The prover computes a single shared response and two Paillier randomness responses:

$$z = \alpha + c \cdot r_s,$$
$$\tau_B = \beta_B \cdot \rho_B^c \bmod N_B,$$
$$\tau_C = \beta_C \cdot \rho_C^c \bmod N_C.$$

The proof is $\pi_{\text{inv}} = (A_D, A_B, A_C, z, \tau_B, \tau_C)$.

Verification. The verifier accepts iff all three of the following hold (same z in all three):

(1) **EC check:** $z\,H \stackrel{?}{=} A_D + c \cdot D_s$.
(2) **Bob–Paillier check:** $g_B^z\,\tau_B^{N_B} \stackrel{?}{=} A_B \cdot \mathrm{ct}_B^c \pmod{N_B^2}$.
(3) **Committee–Paillier check:** $g_C^z\,\tau_C^{N_C} \stackrel{?}{=} A_C \cdot \mathrm{ct}_C^c \pmod{N_C^2}$.

Theorem 4.2 (Soundness of the Invoice Consistency Proof). *The Invoice Consistency Proof is a NIZK for $\mathcal{R}_{\text{inv}}$ in the random-oracle model. In particular, the shared response z in all three verification equations forces a single integer r_s to be the discrete log of D_s and the Paillier plaintext of both ct_B and ct_C.*

The complete proof (completeness, special soundness, and honest-verifier zero-knowledge) is given in Appendix E.1.

4.4 Transfer Consistency Proof

In the transfer phase, Alice folds her current holder secret x_s into the recipient's ciphertext ct_B to produce the next holder ciphertext T_{s+1}. The Transfer Consistency Proof π_{xfer} is a NIZK that forces the folded value to match the public anchor X_s, preventing Alice from injecting an arbitrary secret that would leave Bob with an unusable credential. This proof assumes the Invoice Consistency Proof has already been verified, so D_s and ct_B are known to be consistent with a single r_s.

Definition 4.3 (Transfer Consistency Relation). *The relation $\mathcal{R}_{\text{xfer}}$ is defined as*

$$\mathcal{R}_{\text{xfer}} = \left\{ (X_s, \mathrm{ct}_B, T_{s+1};\ x_s, \eta_s) \ :\ \begin{array}{l} X_s = x_s H, \\ T_{s+1} \cdot \mathrm{ct}_B^{-1} = g_B^{x_s}\,\eta_s^{N_B} \bmod N_B^2 \end{array} \right\}.$$

Sigma protocol. The prover (Alice) samples a commitment scalar $\alpha \leftarrow \mathbb{Z}_{\tilde{q}}$ and a Paillier commitment randomness $\beta \leftarrow \mathbb{Z}_{N_B}^*$, and computes:

$$A_X = \alpha H,$$
$$A_T = g_B^{\alpha}\,\beta^{N_B} \bmod N_B^2.$$

Fiat–Shamir challenge.

$$c = \mathsf{Hash}\big(\texttt{tb/xfer},\ \texttt{ctx_id},\ s{+}1,\ \texttt{inv_id},\ H,\ pk_B,\ X_s,\ \mathrm{ct}_B,\ T_{s+1},\ A_X,\ A_T\big) \bmod q.$$

Response.

$$z = \alpha + c \cdot x_s,$$
$$\tau = \beta \cdot \eta_s^c \bmod N_B.$$

The proof is $\pi_{\text{xfer}} = (A_X, A_T, z, \tau)$.

Verification. The verifier accepts iff both of the following hold (same z):

(1) **EC check:** $z\,H \stackrel{?}{=} A_X + c \cdot X_s$.
(2) **Paillier check:** $g_B^z\,\tau^{N_B} \stackrel{?}{=} A_T \cdot (T_{s+1} \ominus \mathrm{ct}_B)^c \pmod{N_B^2}$.

Additionally, the verifier checks the public anchor transition:

$$X_{s+1} \stackrel{?}{=} X_s + D_s.$$

This check is outside the zero-knowledge relation itself but is necessary for protocol correctness.

Theorem 4.4 (Soundness of the Transfer Consistency Proof). *The Transfer Consistency Proof is a NIZK for $\mathcal{R}_{\text{xfer}}$ in the random-oracle model. The shared response z forces a single integer x_s to be both the discrete log of X_s and the Paillier plaintext of $T_{s+1} \ominus \mathrm{ct}_B$.*

The complete proof follows the same structure as Theorem 4.2 and is given in Appendix E.2.

4.5 Joint Consequence: Correct Transfer Step

The two proofs above each bind a single witness to one algebraic domain; together, they guarantee that every transfer step preserves the fixed-key invariant. The following theorem makes this precise.

Theorem 4.5 (Transfer Correctness). *If both π_{inv} and π_{xfer} verify, and the anchor transition $X_{s+1} = X_s + D_s$ holds, then there exist integers x_s and r_s such that*

$$X_s = x_s H, \tag{2}$$
$$D_s = r_s H, \tag{3}$$
$$T_{s+1} = \mathrm{Enc}_{pk_B}(x_s + r_s), \tag{4}$$
$$X_{s+1} = (x_s + r_s)\,H. \tag{5}$$

Furthermore, the committee-side update $y_{s+1} = y_s - r_s$ preserves the invariant (1)*:*

$$(x_{s+1} + y_{s+1})\,G = (x_s + y_s)\,G = P.$$

Proof. From the Invoice Consistency Proof (Theorem 4.2), extract r_s satisfying $D_s = r_s H$ and $\mathrm{ct}_B = \mathrm{Enc}_{pk_B}(r_s)$. From the Transfer Consistency Proof (Theorem 4.4), extract x_s satisfying $X_s = x_s H$ and $T_{s+1} \ominus \mathrm{ct}_B = \mathrm{Enc}_{pk_B}(x_s)$. By Paillier's additive homomorphism, $T_{s+1} = \mathrm{Enc}_{pk_B}(x_s) \oplus \mathrm{Enc}_{pk_B}(r_s) = \mathrm{Enc}_{pk_B}(x_s + r_s)$, giving (4). The anchor transition gives $X_{s+1} = X_s + D_s = x_s H + r_s H = (x_s + r_s)\,H$, giving (5). Setting $x_{s+1} \equiv x_s + r_s \pmod q$ and $y_{s+1} \equiv y_s - r_s \pmod q$, we obtain $(x_{s+1} + y_{s+1})\,G = (x_s + y_s)\,G = P$. □

4.6 Threshold Committee via Simple VESS

The construction so far assumes a single third party C. A compromised C can collude with any holder to recover $x_s + y_s$ and steal the UTXO. We replace C with a committee of n members via Simple VESS [56]: at channel open, y_0 is distributed as a (t, n)-Shamir sharing with Feldman commitments [29], all such commitments being taken in base H so that a free-term commitment lives in the same domain as the anchors and deltas; at each transfer, Bob replaces ct_C with a VESS distribution of r_s and attaches a consistency proof. This is a *modular substitution*: the Transfer Consistency Proof remains unchanged, and π_{inv} narrows to the two-receiver relation (D_s, ct_B) while the committee-side binding is handled by the VESS consistency proof.

Proof bridging. In the threshold setting, π_{inv} proves the restricted relation $\mathcal{R}^{\text{thr}}_{\text{inv}}$, whose public statement is (D_s, ct_B) and whose secret witness is (r_s, ρ_B). It is obtained from Definition 4.1 by omitting ct_C and ρ_C. The committee-side binding comes from the VESS consistency proof, which guarantees that the encrypted shares encode a unique polynomial whose free-term commitment is F_0. The bridge is the public identity $F_0 = D_s$: π_{inv} certifies that D_s commits to r_s, and VESS certifies that the shares are consistent evaluations of the same r_s. Every verifier explicitly checks $F_0 = D_s$ and rejects the invoice if this public equality does not hold.

Committee update and finalization. Each member C_i decrypts its share $r_s^{(i)}$ and updates locally: $y_{s+1}^{(i)} = y_s^{(i)} - r_s^{(i)}$. Since both the committee polynomial and the sharing polynomial for r_s have degree $t-1$, their difference is also degree $t-1$ with free term $y_{s+1} = y_s - r_s$, preserving the Shamir structure and the fixed-key invariant. At finalization, the members that acknowledge the holder's closing request sign as one side of a two-of-two, mirroring the additive split $P = (x_s + y_s)\,G$ of Section 2.2: the holder contributes x_s, and the committee contributes its side t-of-n in FROST style [39, 53], with member C_i adding $e\,\lambda_i y_s^{(i)}$, where λ_i is the Lagrange coefficient of member i in a participating set S of at least t members, so that these contributions aggregate to $e\,y_s$ without reconstructing y_s. The shared challenge e is computed on the combined nonce, P, and the closing transaction, as in Section 2.2.

Security. As long as fewer than t members are corrupted, the adversary learns nothing about y_s or r_s (by information-theoretic security of Shamir sharing), and VESS verification prevents a malicious Bob from distributing inconsistent shares. The resulting trust model is the standard (t, n)-threshold honest-majority assumption, identical to FROST [39] and ROAST [53].

Theorem 4.6 (VESS Modularity). *The threshold-committee instantiation preserves every property of the single-third-party construction (fixed-key invariance, anchor correctness, ciphertext binding, and proof soundness) under the (t, n)-threshold honesty assumption and the security of Simple VESS [56].* (Proof in Appendix E.3.)

5 Security Analysis

We analyze the security of Thunderbolt within the Universal Composability (UC) framework of Canetti [14]. Section 5.1 formalizes the claim: it states the UC realization goal, specifies the hybrid model and the target ideal functionality, and concludes with the main theorem. Section 5.2 proves it: the goal splits into a trace-simulation lemma and a forgery bound, each committed ledger entry is discharged in one forward simulator step, and UC composition lifts the result to concrete component realizations. Section 5.3 then analyzes concrete attack vectors, and Section 5.4 covers two optional overlays that buy unilateral exit and off-chain splitting.

5.1 UC Realization in the Hybrid Model

Let λ be the computational security parameter and write π_{TF} for Thunderbolt's transfer-and-finalization logic. Our UC security statement asserts that, for every adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}_{\text{sl}}$ that renders the hybrid execution of π_{TF} indistinguishable from the ideal execution of $\mathcal{F}_{\mathsf{TB}}$ to every environment $\mathcal{Z}$:

$$\forall \mathcal{A} \in \mathsf{PPT}\ \exists \mathcal{S}_{\text{sl}} \in \mathsf{PPT}\ \forall \mathcal{Z} \in \mathsf{PPT}:$$
$$\text{EXEC}^{\mathcal{H}_{\text{std}}}_{\pi_{\mathsf{TF}},\mathcal{A},\mathcal{Z}}(\lambda) \approx_c \text{EXEC}_{\mathcal{F}_{\mathsf{TB}},\mathcal{S}_{\text{sl}},\mathcal{Z}}(\lambda).$$

Here $\text{EXEC}^{\mathcal{H}}_{\mathcal{M},\mathcal{B},\mathcal{Z}}(\lambda)$ is $\mathcal{Z}$'s output distribution in the $\mathcal{H}$-hybrid execution of $\mathcal{M}$ with $\mathcal{B}$ controlling its adversarial interface (the superscript is dropped for an ideal execution), and $\approx_c$ is computational indistinguishability. The quantifier order requires a single PPT simulator to work for every PPT environment.

Hybrid model. In the left execution, π_{TF} runs in the $\mathcal{H}_{\text{std}}$-hybrid model, whose five components are

$$\mathcal{H}_{\text{std}} := (\mathcal{F}_{\text{Ledger}}, \mathcal{F}_{\text{PKI}}, \mathcal{F}_{\text{NIZK}}, \mathcal{F}_{\text{VESS}}, \mathcal{F}^{\text{Auth}}_{\text{TS}}),$$

with the following interfaces.

- **Ledger ($\mathcal{F}_{\text{Ledger}}$).** $\mathcal{F}_{\text{Ledger}}$ reveals each proposed entry on the adversarial interface and commits entries in the adversarially scheduled per-channel order, enforcing G1–G2 of Section 3.2; G3 enters only the liveness discussion (Section 5.3), not the realization claim.
- **Key registration ($\mathcal{F}_{\text{PKI}}$).** $\mathcal{F}_{\text{PKI}}$ registers and authenticates the public encryption keys of holders and committee members, each registered by the party it belongs to, so a registered key binds that identity.
- **Zero-knowledge relations ($\mathcal{F}_{\text{NIZK}}$).** $\mathcal{F}_{\text{NIZK}}$ provides the standard prove/verify interface for $\mathcal{R}_{\text{own}}$ (Section 4.2), $\mathcal{R}_{\text{xfer}}$ (Section 4.4), and $\mathcal{R}^{\text{thr}}_{\text{inv}}$ (Section 4.6). For an honest prover it returns a proof without revealing the witness; a corrupted prover must supply a witness, which it exposes on its adversarial interface for the security proof to read. The simulator itself needs only the verify verdicts.
- **Encrypted sharing ($\mathcal{F}_{\text{VESS}}$).** $\mathcal{F}_{\text{VESS}}$ idealizes Shoup's Simple VESS construction [56]: the dealer submits a degree-$(t-1)$ sharing of its secret, and the functionality publishes the base-H Feldman commitments and delivers each share privately. A corrupted dealer must submit its sharing as input, from which the simulator reads the dealt secret; corrupted members learn only their own shares.
- **Authorized threshold signing ($\mathcal{F}^{\text{Auth}}_{\text{TS}}$).** At the policy level, standard (t, n) UC threshold-signature functionalities [9, 15] authorize a signer set U when

$$\mathsf{Thresh}_t(U) \iff |U \cap \{\mathsf{C}_1, \ldots, \mathsf{C}_n\}| \geq t.$$

Thunderbolt's $\mathcal{F}^{\text{Auth}}_{\text{TS}}$ strengthens this with an ownership condition stated over the public signing context carried by every request, $\mathsf{ctx} = (\mathsf{ctx_id}, s, P, X_s, \mathsf{tx})$, and signs under P only when

$$\mathsf{Auth}(\mathsf{ctx}, U) \iff \mathsf{Own}(\mathsf{ctx}, U) \wedge \mathsf{Thresh}_t(U),$$

where $\mathsf{Own}(\mathsf{ctx}, U)$ holds iff some party in U possesses an $\mathcal{F}_{\text{NIZK}}$-accepted proof for $\mathcal{R}_{\text{own}}$ on the anchor X_s in ctx, bound to ctx. The conjunction is the two-of-two structure of finalization (Section 4.6). Stating the guard over the accepted statement rather than the holder's identity keeps it evaluable from the functionality's own inputs. The knowledge-based guard and the identity-based one coincide except on a forgery event that we later bound to negligible probability.

Functionality $\mathcal{F}_{\mathsf{TB}}$

Parameters	$(\mathsf{ctx_id}, P, t, n)$
Initial state	$(0, \mathsf{H}_0, \bot, \emptyset, \emptyset)$
Mutable state	$(s, \mathsf{holder}, \mathsf{fin}, \mathcal{I}, \mathsf{rcpt})$

Committed entry	*Semantics (precondition → effect)*
$(\mathsf{invoice}, \mathsf{inv_id})$ from a recipient	$\mathsf{rcpt}[\mathsf{inv_id}] = \bot \rightarrow$ $\mathcal{I} \leftarrow \mathcal{I} \cup \{\mathsf{inv_id}\}$ $\mathsf{rcpt}[\mathsf{inv_id}] \leftarrow$ that recipient
$(\mathsf{transfer}, \mathsf{inv_id})$ from holder	$\mathsf{fin} = \bot \ \wedge \ \mathsf{inv_id} \in \mathcal{I} \rightarrow$ $\mathcal{I} \leftarrow \mathcal{I} \setminus \{\mathsf{inv_id}\}$ $s \leftarrow s + 1$ $\mathsf{holder} \leftarrow \mathsf{rcpt}[\mathsf{inv_id}]$
(finalize) from holder and committee members	$\mathsf{fin} = \bot \rightarrow$ $\mathsf{fin} \leftarrow \top$ release the spending authority under P

Conventions. Every committed entry and the state it induces is public; entries take effect atomically in the adversarially selected order, and an entry failing its precondition is ignored; there is no other transition.

Figure 3: The Thunderbolt ideal functionality $\mathcal{F}_{\mathsf{TB}}$.

Ideal functionality. The interfaces above fix the left execution. For the right one we need a machine that grants Thunderbolt's safety properties by inspection, so that realizing it transfers those properties to the protocol. $\mathcal{F}_{\mathsf{TB}}$, given in Figure 3, is that machine, obtained by reducing the channel lifecycle of Section 4.2 to its ownership content. Its mutable state comprises the step counter s, the current holder, the finalization flag fin, the pending-invoice set $\mathcal{I}$, and the recipient map rcpt.

- **Deriving $\mathcal{F}_{\mathsf{TB}}$ from the protocol.** The Open phase fixes the parameters $(\mathsf{ctx_id}, P, t, n)$ once and for all, and produces (X_0, T_0), the public anchor and encrypted credential that carry holdership in the protocol; $\mathcal{F}_{\mathsf{TB}}$ compresses this pair into the initial state $\mathsf{holder} = \mathsf{H}_0$ at $s = 0$. The Finalize phase keeps only its ledger half, since once the spending authority is released, broadcasting the resulting signature adds nothing. What remains of a channel is what it places on the Thunderbolt Ledger: an invoice and a transfer entry per transfer, and one finalization entry. Each entry is stripped to its type, the invoice identifier when it has one, and the parties submitting it; the cryptographic payload plays no role, so rules fire on bookkeeping alone. The ledger guarantees of Section 3.2 become the conventions: G1–G2 make a committed entry permanent and per-channel ordered, so entries take effect in an adversarially selected order and an uncommitted entry changes nothing; G3 is excluded, as it governs whether entries appear, not what a committed entry does.
- **Guarantees of $\mathcal{F}_{\mathsf{TB}}$.** $\mathcal{F}_{\mathsf{TB}}$ enforces the three safety properties of Section 3.2 by construction: *ownership security*, as only holder transfers or finalizes and only it obtains the channel's spending authority; *fixed-key integrity*, as P is an immutable parameter; and *transfer correctness* at the ownership level, as each identifier is consumed at most once, the share-level statement being Theorem 4.5. Liveness is of a different kind and is not established below: scheduling is adversarial in both executions, so termination remains conditional on G3 and on at least t honest, responsive committee members.

Theorem 5.1 (Thunderbolt UC-realizes $\mathcal{F}_{\mathsf{TB}}$). *Assume ECDLP is hard in $\mathbb{G}$ and Paillier encryption is semantically secure. Then, for any channel state produced by the Open phase and satisfying $P = (x_0 + y_0)G$, $X_0 = x_0 H$, and $T_0 = \mathrm{Enc}_{pk_{\mathsf{H}_0}}(x_0)$, Thunderbolt's transfer-and-finalization logic π_{TF} UC-realizes $\mathcal{F}_{\mathsf{TB}}$ in the $\mathcal{H}_{\mathrm{std}}$-hybrid model, against a static adversary corrupting any subset of holders and at most $t - 1$ committee members. The simulator $\mathcal{S}_{\mathrm{sl}}$ is straight-line: it processes each committed ledger entry once and never rewinds or forks the environment or adversary.*

5.2 Security Proof

We structure the proof as a chain of dependencies: discharging each committed ledger entry in one forward simulator step proves one edge of the paired hybrid–ideal trace (Figure 4), induction over all edges yields Lemma 5.2, the hardness assumptions yield Lemma 5.3, the two chain into Theorem 5.1, and UC composition lifts it to concrete component realizations (Corollary 5.4).

Following prior UC analyses of threshold signatures [15], we decompose the claim of Theorem 5.1 before constructing its simulator. Write HYB and IDEAL for the two execution ensembles in Theorem 5.1, and let

$$\Delta_{\mathcal{Z}}(\lambda) := |\Pr[\mathsf{HYB} = 1] - \Pr[\mathsf{IDEAL} = 1]|$$

be the distinguishing gap. The hybrid protocol authorizes transfer and finalization by knowledge, that is, by accepted $\mathcal{F}_{\mathrm{NIZK}}$ proofs, whereas $\mathcal{F}_{\mathsf{TB}}$ authorizes by identity; closing this mismatch is the task of the proof, and the mismatch is captured by a single forgery event. Let Forge be the event, over the $\mathcal{H}_{\mathrm{std}}$-hybrid execution with $\mathcal{A}$ and $\mathcal{Z}$, that $\mathcal{A}$ submits to $\mathcal{F}_{\mathrm{NIZK}}$ a valid witness for $\mathcal{R}_{\mathrm{own}}$ or $\mathcal{R}_{\mathrm{xfer}}$ against the current anchor of an honest holder. In this hybrid, the witness is an ordinary input from the corrupted prover to the ideal interface; observing it in the reduction is straight-line and is not extraction from a proof transcript. The claim then splits into two parts.

Lemma 5.2 (Straight-line trace simulation). *For every PPT adversary $\mathcal{A}$ satisfying the corruption bound, there exists a PPT straight-line simulator $\mathcal{S}_{\mathrm{sl}}$ such that, for every PPT environment $\mathcal{Z}$,*

$$\Delta_{\mathcal{Z}}(\lambda) \le \Pr[\mathsf{Forge}].$$

The simulator processes each committed ledger entry once; conditioned on ¬Forge, it preserves the same state transitions and public delivery schedule.

Lemma 5.3 (Forgery bound). *Under ECDLP in $\mathbb{G}$ and Paillier semantic security,*

$$\Pr[\mathsf{Forge}] \le \mathrm{negl}(\lambda).$$

Chaining the two bounds gives $\Delta_{\mathcal{Z}}(\lambda) \le \Pr[\mathsf{Forge}] \le \mathrm{negl}(\lambda)$, so the two lemmas together imply Theorem 5.1; Appendix F proves them in full.

We decompose trace simulation over the committed ledger entries. Fix a trace $e_1, \ldots, e_m$, each entry of one of the three types of Figure 3. Let $h^{(k)}$ be the hybrid state after its first k entries and $\Sigma^{(k)}$ the joint state of $(\mathcal{F}_{\mathsf{TB}}, \mathcal{S}_{\mathrm{sl}})$ after the same prefix. The relation $\mathcal{R}_k := \mathcal{R}(h^{(k)}, \Sigma^{(k)})$ requires agreement on the committed

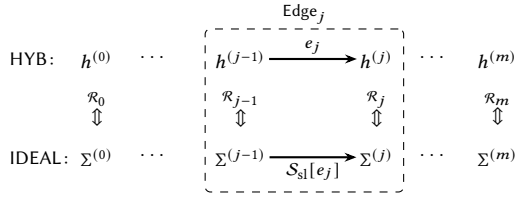

Figure 4: Each committed ledger entry e_j advances both executions by one step and preserves the trace invariant.

prefix, current holder, pending-invoice set, recipient map, and finalization status, as well as consistency of simulated honest-holder credentials and honest committee shares with the current public anchor and fixed key. The correctly opened channel establishes $\mathcal{R}_0$. Write $\mathcal{S}_{\mathrm{sl}}[e_j]$ for its single forward step on e_j, and let Edge_j denote preservation of $\mathcal{R}_{j-1}$ to $\mathcal{R}_j$ across the corresponding hybrid and ideal transitions. Thus the paired hybrid and ideal state sequences reduce Lemma 5.2 to proving every Edge_j under ¬Forge.

Proof sketch. Fix an arbitrary PPT adversary $\mathcal{A}$ satisfying the corruption bound. We construct $\mathcal{S}_{\mathrm{sl}}$, which internally runs $\mathcal{A}$, generates the encryption keys of simulated honest holders and committee members as they join, emulates the public view of the hybrid functionalities, and preserves the adversarial ordering and delivery schedule, processing each committed entry exactly once.

We prove Edge_j for an arbitrary committed entry e_j in three steps. First, the interfaces of Section 5.1 selected by the entry's type (proof, sharing, authorization, and ledger commitment) validate it, and reject malformed entries identically in both executions. Second, if the ideal transition needs a hidden value, $\mathcal{S}_{\mathrm{sl}}$ obtains it in one forward computation, as in the three cases below. Finally, π_{TF} applies its deterministic state checks (each invoice consumed once, $F_0 = D_s$, $X_{s+1} = X_s + D_s$, at most one finalization), and $\mathcal{S}_{\mathrm{sl}}$ maps the accepted entry to the corresponding ideal transition.

Corrupted invoice creator. A corrupted dealer must submit its degree-$(t-1)$ sharing polynomial f to $\mathcal{F}_{\mathrm{VESS}}$; emulating that interface, $\mathcal{S}_{\mathrm{sl}}$ reads f as ordinary input and sets $r_s = f(0)$ in one forward step. With a concrete VESS scheme it instead recovers r_s by decrypting any t honest members' shares under keys it generated and interpolating, which is available whenever at least t members are honest, a weaker condition than liveness, which additionally requires those members to be responsive.

Corrupted sender. If recipient B is honest, $\mathcal{S}_{\mathrm{sl}}$ decrypts T_{s+1} under sk_B to recover x_{s+1} and checks $x_{s+1}H = X_s + D_s$. If both endpoints are corrupted, no hidden plaintext is needed: the accepted interface statements and ordered ledger state determine the ideal transition.

Finalization and concurrency. $\mathcal{F}_{\mathrm{TS}}^{\mathsf{Auth}}$ releases a signature only upon an accepted $\mathcal{R}_{\mathrm{own}}$ proof for the current anchor together with at least t committee members. Interleaved sessions remain independent because every functionality call carries its full context. Thus each committed entry yields its edge Edge_j.

Conditioned on ¬Forge, the per-entry argument and induction over $e_1, \ldots, e_m$ give

$$\mathcal{R}_0 \wedge \bigwedge_{j=1}^{m} \mathsf{Edge}_j \Longrightarrow \mathcal{R}_m.$$

Both worlds then expose the same public outputs and delivery schedule, so $\Delta_{\mathcal{Z}}(\lambda) \le \Pr[\mathsf{Forge}]$, proving Lemma 5.2. For Lemma 5.3, Appendix F.2 embeds the honest anchor in an ECDLP challenge and replaces the associated honest Paillier ciphertexts using Paillier semantic security; the VESS dealing requires no computational step, since $\mathcal{F}_{\mathrm{VESS}}$ reveals only the corrupted members' shares, which are information-theoretically independent of the free term. It obtains $\Pr[\mathsf{Forge}] \le \mathsf{negl}(\lambda)$. □

Theorem 5.1 is stated in the $\mathcal{H}_{\mathrm{std}}$-hybrid model: it secures the protocol logic with the five components idealized. A deployment replaces each $\mathcal{F}_i$ with a concrete protocol ρ_i. The UC composition theorem makes this replacement sound whenever each ρ_i UC-realizes its interface; the following corollary records this step together with the per-component obligations it induces.

Corollary 5.4 (Concrete composition). *For each $\mathcal{F}_i \in \mathcal{H}_{\mathrm{std}}$, let ρ_i be a protocol that UC-realizes $\mathcal{F}_i$, and write $\pi_{\mathsf{TF}}[\rho_i/\mathcal{F}_i]$ for their substitution into the hybrid protocol. Under the assumptions of Theorem 5.1, $\pi_{\mathsf{TF}}[\rho_i/\mathcal{F}_i]$ UC-realizes $\mathcal{F}_{\mathsf{TB}}$ by the UC composition theorem [14].*

Each channel invokes its own instance of every component functionality, so π_{TF} is subroutine respecting and the composition theorem applies. The corollary then transfers each interface of Section 5.1 into a proof obligation on the chosen ρ_i: ρ_{NIZK} must be straight-line extractable for the three relations (e.g., the compiler of [16], whose setup and random-oracle requirements then join the assumptions of the composed statement); ρ_{VESS} must realize the ideal sharing interface with authenticated, valid committee keys [56], whose simulator recovers a corrupted dealer's value by decrypting t honest shares, so the concrete instantiation tolerates $f \le \min(t-1,\ n-t)$ corrupted members, the standard deployment condition of ROAST-style systems [53], satisfied by the committee shapes evaluated in Section 6; ρ_{TS} must enforce the Auth guard, e.g., FROST-style signing [39] released only upon an $\mathcal{F}_{\mathrm{NIZK}}$-accepted $\mathcal{R}_{\mathrm{own}}$ proof; and the ledger instantiation must provide G1–G2 of Section 3.2, with G3 required only for liveness. The standalone Sigma protocols of Section 4.3 and Section 4.4 instantiate the relations, but compiled with Fiat–Shamir alone they do not realize $\mathcal{F}_{\mathrm{NIZK}}$; Remark E.1 discusses this gap.

5.3 Attack Vector Analysis

We discuss five concrete attack patterns and identify the mechanisms that address them.

Committee–holder collusion. If t committee members collude with the current holder H_s, they can reconstruct y_s and unilaterally spend the UTXO. This is an inherent limitation of the (t,n)-threshold trust model, identical to FROST [39] and ROAST [53]. A *past* holder $\mathsf{H}_{s'}$ cannot spend, even if committee members retain historical refresh values, because: (1) finalization requires a proof of knowledge of x_s matching the *current* anchor X_s, which a past holder cannot produce; (2) fewer than t corrupted members cannot reconstruct any past $y_{s'}$ from their Shamir shares. Past-holder security follows entirely from the (t,n)-threshold honest-majority assumption, without any secure-deletion assumption. Committee selection should follow established threshold-custody principles:

diversity across organizational domains, stake or reputation with slashing, and periodic rotation via proactive resharing.

Double-spending and equivocation. A malicious holder H_s might attempt to (a) consume the same invoice twice, (b) issue valid transfer proofs to two different recipients, or (c) finalize on-chain while a transfer is in flight. Attack (a) is prevented by the *monotonic anchor*: each transfer advances X_s to $X_{s+1} = X_s + D_s$; the committee accepts a proof only against the current anchor, so a replay against the same invoice fails. Attack (b) is prevented identically: the committee processes only the first valid transfer for step s and advances its state, rejecting any subsequent proof for the same step. Attack (c) is mitigated by the cooperative nature of finalization: the holder needs t committee co-signatures and the committee can refuse to sign while a transfer is pending. All three cases correspond to $\mathcal{F}_{\mathsf{TB}}$ ignoring duplicate or out-of-order inputs via the single-consumption rule on invoice identifiers.

Self-transfer loops. A holder may issue an invoice to itself (e.g., $A \to B \to B \to C$). The protocol treats each step uniformly: the committee verifies proofs against the current anchor, advances its state, and updates its share. The self-transfer reduces to a holder-side key rotation and introduces no new attack surface, since exploiting any past secret still requires collusion with t committee members.

Committee unavailability. The liveness condition of Section 3.2 requires at least t honest, responsive committee members. Operational deployments maintain this condition through redundancy ($n \gg t$), heartbeat monitoring, and committee rotation via proactive resharing [56]: the remaining t honest members reshare y_s to a refreshed committee without reconstructing the secret or touching the chain. This is the standard (t, n)-threshold liveness model, identical to FROST [39] and ROAST [53]. A pre-signed unilateral-exit overlay can provide recovery when this condition fails, at the cost discussed next.

Ledger failures. The guarantees in Section 3.2 separate failures that affect liveness from those that may affect safety. Finite delays postpone transfer or finalization without changing the committed state. Permanent censorship violates G3 and may prevent an operation from completing, but an uncommitted entry cannot create a conflicting ownership state. Deleting a committed entry violates G1, while presenting different committed histories violates G2; either failure may cause participants to act on a stale or inconsistent state and thereby compromise safety. Subject to the cryptographic and threshold-honesty assumptions, safety therefore relies on G1 and G2, whereas liveness additionally relies on G3 and at least t honest, responsive committee members.

5.4 Exit Liveness and Off-Chain Splitting

The failure modes above motivate two optional overlays, a unilateral exit and off-chain splitting, which share one pattern: the current holder and committee co-sign a Bitcoin transaction but keep it off-chain unless settlement or recovery requires broadcast. Both are strictly additive; the core lifecycle and the realization proved in Theorem 5.1 remain unchanged.

Unilateral-exit overlay. An exit overlay gives the current holder a unilateral Bitcoin L1 fallback when the committee becomes unavailable. Following Spark [40], each transfer additionally creates a pre-signed exit transaction with a decrementing relative delay, so that the current holder's exit matures before a stale holder's competing exit, with no absolute-expiry deadline but a bounded response window. The costs are the ones Spark also pays: the holder or a watchtower must publish the latest exit within the resulting response window, every transfer gains one holder–committee co-signing round, and a finite range of delays bounds the transfers per exit epoch, after which an on-chain refresh is required. The core protocol takes the opposite point on this axis: committee unavailability can pause transfer or finalization, but a committed recipient never has to refresh, monitor Bitcoin, or come online within a deadline to preserve ownership.

Off-chain splitting overlay. The same pattern supports subdivision without first placing the split on chain. The holder and committee pre-sign an unbroadcast transaction that spends the channel UTXO into k child outputs, child j locked under a fresh joint key $P^{(j)} = (x_0^{(j)} + y_0^{(j)})G$; the Ledger then atomically consumes the parent context and opens one context per child. Each child runs the original lifecycle independently under its own fixed-key invariant. Settling the first child broadcasts the split transaction and materializes all child outputs; each child then closes with its own transaction, still requiring committee co-signing, and the exit overlay above can be composed per child.

Subdivision widens the application range. The core protocol serves workloads that move a fixed denomination end to end, such as payroll batches, cold-wallet receipts, custody chains, and intermittently-online devices, all of which fix the amount when the channel is funded. Splitting lifts that restriction at the cost of one pre-signed transaction: a payer can split a channel into a change child and a payment child for exact payments, a payroll operator can fan one funded UTXO out into per-recipient child channels in a single ledger transition, and a custodian can subdivide a holding into per-client channels. Each child channel retains whole-UTXO transfer internally, so unbounded asynchronous receipt carries over unchanged.

6 Evaluation

We address three research questions:

- **RQ1** (Section 6.1): How fast is a single off-chain transfer, and does per-transfer cost remain constant as the transfer count grows?
- **RQ2** (Section 6.2): How does Thunderbolt's Sigma-protocol construction compare with general-purpose SNARK backends?
- **RQ3** (Section 6.3): How does the threshold-committee path scale with committee size and fault tolerance?

Implementation. We implement Thunderbolt in approximately 5 700 lines of Rust (excluding tests and benchmarks), organized into two crates: `thunderbolt-core` (the protocol logic, proofs, and

Table 1: Per-stage latency and category share for a single off-chain transfer (100 iterations, 3072-bit Paillier, secp256k1).

Stage	Avg (ms)	Min (ms)	Max (ms)	Share
Invoice setup	119	116	153	18%
Transfer setup	65	63	82	
Invoice prove	309	302	340	46%
Transfer prove	162	157	169	
Invoice verify	196	192	218	30%
Anchor verify	< 1	< 1	< 1	
Transfer verify	111	108	115	
Bob decrypt	30	30	33	6%
Committee update	30	30	32	
Total (off-chain)	**1022**	**999**	**1073**	**100%**

threshold-committee path) and `bitcoin-utils` (transaction construction and signing helpers). The implementation uses the following open-source libraries: `libpaillier` (v0.6) for Paillier key generation, encryption, and decryption; `secp256k1` (v0.31) via the `secp` wrapper for elliptic-curve scalar and point arithmetic on secp256k1; `frost-secp256k1` (v2.2) and `musig2` (v0.2) for threshold and two-party Schnorr signing at channel finalization; `rayon` (v1.10) for parallel VESS share-proof generation; and `sha2` for the Fiat–Shamir hash. The SP1 comparison benchmark (approximately 860 lines) uses the Succinct SP1 zkVM toolchain. All code is compiled with `rustc` 1.90.0 in release mode with link-time optimization (LTO) enabled. The implementation and benchmarks are open-source.

Experimental setup. All benchmarks were run on a single machine with an Apple M3 Max processor (14-core, 10 performance + 4 efficiency), 36 GB unified memory, macOS 15.2. The Paillier modulus is 3072 bits ($|N| = 3072$, so all Paillier arithmetic operates over 6144-bit integers in $\mathbb{Z}_{N^2}$). The single-committee microbenchmarks report the average, minimum, and maximum over 100 iterations with a 3-iteration warmup discarded. The threshold-committee benchmark uses the same four committee shapes as ROAST [53] (3-of-5, 11-of-15, 34-of-50, 67-of-100) with 3 iterations, a 160 ms baseline RTT, ±20% multiplicative jitter per honest member, and a network fanout of 4 (shares are delivered in waves of 4 members, each wave separated by one baseline RTT). Non-responsive members are modeled as silent: they never reply, forcing the protocol to reach threshold from the remaining honest members alone.

6.1 RQ1: Transfer Latency and Scalability

A single off-chain transfer executes nine stages: invoice setup, invoice prove, invoice verify, transfer setup, transfer prove, anchor verify, transfer verify, Bob decrypt, and committee update. Table 1 reports measured wall-clock time for each stage.

Cost breakdown. The total off-chain transfer completes in 1022 ms on average. The right side of Table 1 groups the nine stages by functional role. Invoice Prove (309 ms) and Transfer Prove (162 ms) dominate at 46% combined; both are bottlenecked by modular exponentiations over 6144-bit Paillier integers ($\mathbb{Z}_{N^2}$), while the elliptic-curve operations are negligible by comparison. This Paillier-heavy profile explains the circuit-backend overhead (Section 6.2).

Table 2: Measured prove/verify cost per transfer.

Backend	Prove	Verify	Slowdown
Thunderbolt (Sigma)	471 ms	307 ms	1×
SP1 zkVM (256-bit Paillier)	54 600 ms	67 ms	116×

Proof size. π_{inv} totals ≈ 2.3 KB (three commitments, one shared response, two randomness responses at $|N| = 3072$); π_{xfer} totals ≈ 1.5 KB; combined ≈ 3.8 KB per step.

Scalability over many transfers. Because each transfer depends only on the current state (X_s, T_s, y_s) and no party re-verifies prior transfers, per-transfer cost is $O(1)$: running 1 000 sequential transfers on a single-committee channel confirms that every transfer takes the same ≈ 1010–1040 ms, with a constant proof size of 3.8 KB per step.

6.2 RQ2: Comparison with SNARK Backends

Sigma protocols vs. general-purpose SNARKs. An alternative realization would encode our relations as arithmetic circuits for a SNARK backend (Groth16 [34], Plonk [31], or a zkVM such as SP1 [59] or RISC Zero [52]). The Invoice and Transfer relations are dominated by Paillier modular exponentiation in $\mathbb{Z}_{N^2}$ for a 3072-bit N, so a generic SNARK backend must handle repeated non-native arithmetic over 6144-bit values, bit decomposition and modular reduction for ciphertext equations, and wiring the same witness consistently across both Paillier checks and the secp256k1 verification equation.

Why we do not implement a full Groth16 backend. A faithful Groth16 realization of Thunderbolt would therefore require a large custom circuit for non-native big-integer arithmetic, careful constraint engineering for modular exponentiation in $\mathbb{Z}_{N^2}$, and a per-circuit trusted setup. This is a substantial implementation burden before any performance optimization, and the resulting prover would still inherit the high cost of circuit-based proving on large-modulus arithmetic. We therefore do not implement a full Groth16 backend. This engineering gap is itself informative: Thunderbolt's statements are native algebraic relations, and forcing them into a generic SNARK circuit already erases one of the main advantages of our construction.

Reduced-parameter SP1 measurement. We instead measure the closest implemented general-purpose backend: an SP1 zkVM proof of the same invoice/transfer Paillier verification structure under a reduced 256-bit Paillier modulus. This is far smaller than the 3072-bit modulus used by Thunderbolt, but the proving cost is already high: averaged over five runs, SP1 takes 54 600 ms to prove and 67 ms to verify, compared with 471 ms and 307 ms for our native Sigma proofs. That is a 116× proving slowdown even at the reduced parameter. Larger Paillier moduli were not feasible on our laptop setup because local proving exceeded the available memory budget; increasing the modulus would also further increase proving time.

Our direct Sigma-protocol construction requires no circuit: each proof consists of a constant number of native group operations. Concretely, proving both proofs takes ≈ 471 ms and verifying both takes ≈ 307 ms. Table 2 summarizes the comparison.

Even after shrinking the Paillier modulus far below the deployed setting, the general-purpose zkVM backend remains over two orders of magnitude slower to prove. Together with the implementation complexity and trusted setup requirements of a full Groth16 circuit, this supports our design choice: Thunderbolt's algebraic relations are better handled natively by Sigma protocols than by generic SNARK backends.

6.3 RQ3: Threshold Committee Scalability

We benchmark the full threshold-committee transfer path of Section 4.6 under the four committee configurations and network model described in the experimental setup. Figure 5 reports the measured committee communication latency across all configurations. For each configuration, we measure the *committee communication latency*, defined as the wall-clock time from when Bob begins delivering shares until t members have responded, under three adversarial scenarios: 0, $\lceil n/6 \rceil$, and $\lceil n/3 \rceil$ non-responsive members (capped at $n-t$, the maximum tolerable), uniformly distributed across the committee.

Committee communication scales linearly. Committee communication latency grows linearly with n: from 314 ms at n=5 to 2957 ms at n=100 with no adversary (Figure 5). The slope is governed by the fanout constraint: shares are delivered in waves of 4 members per baseline RTT, and the protocol completes once t members have responded. For a 67-of-100 committee, reaching the threshold requires delivering to at least 67 members, or $\lceil 67/4 \rceil = 17$ waves; the 17th wave starts at 16×160 ms = 2560 ms, and response-time jitter accounts for the remaining gap to the measured 2957 ms.

Resilience to non-responsive members. When $\lceil n/3 \rceil$ members are non-responsive (at the cap $n-t$), the protocol must collect t responses from the remaining honest members alone. At n=100, committee communication latency rises from 2957 ms to 4497 ms, a 52% increase (Figure 5). At n=50, the overhead is 41% (1577 ms to 2219 ms). These increases reflect the fanout cost of reaching more distant honest members: with $\lceil n/3 \rceil$ members silent and uniformly distributed, the protocol must wait for honest members in later delivery batches that would otherwise be unnecessary. Crucially, all configurations complete successfully and preserve the fixed-key invariant at every adversary level, and the $\lceil n/6 \rceil$ curve shows a moderate intermediate overhead (17% at n=100), demonstrating graceful degradation.

End-to-end transfer latency. The full threshold transfer (VESS share generation, distribution verification, transfer prove/verify, Bob decrypt, and committee communication) takes 1720 ms at n=5 and 15 980 ms at n=100 with no adversary. VESS share generation dominates: Bob must compute n independent Paillier encryptions and NIZK proofs, which accounts for over 80% of the total at n=100. Committee communication, by contrast, is a bounded fraction of the end-to-end cost that degrades gracefully under adversarial conditions.

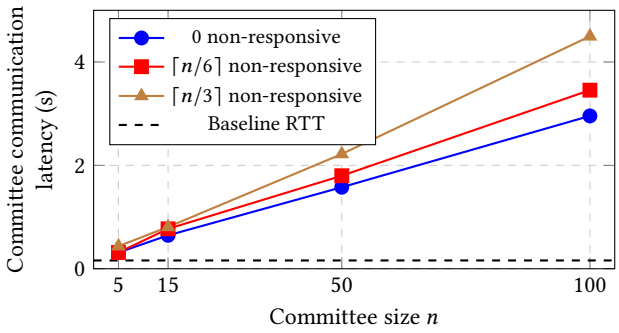

Figure 5: Committee communication latency by committee size and non-responsive level.

7 Related Work

Payment channels and layer-2 protocols. Off-chain payment channels date to Spilman [58] and were generalized by Lightning [51], duplex channels [25], and eltoo [24]. Extensions address concurrency [43], atomic updates [27], rebalancing [38], state machines [45], and miner rationality [2]; surveys appear in [17, 35]. Virtual channels [3, 5, 26] and channel factories [12] reduce on-chain cost; Sleepy Channels [6] remove the watchtower requirement. All these designs fix endpoints at open time and optimize for routing across pre-funded channels. Ark [1, 37] takes a route-free but round-based approach: normal operation is coordinated through an Ark Service Provider (ASP), and holders must refresh or exit before VTXO expiry. Although pre-signed paths permit unilateral recovery, the failure path is interaction-sensitive and depends on Bitcoin L1 fees and congestion because the holder must publish the relevant transaction path and wait through its timelocks. Beyond Bitcoin, Ethereum's layer-2 landscape adds optimistic rollups [47], zk-rollups [44], state channels [26, 45], and commit-chains [36], all leveraging expressive smart contracts unavailable in Bitcoin's scripting model, while BitVM [42] brings computation verification to Bitcoin via fraud proofs. Thunderbolt targets a complementary point: sequential ownership migration of a single UTXO on standard Taproot, with no routing, no directional capacity, and no simultaneous-online requirement.

Statechains. Somsen's statechain proposal [57] is the closest prior work: it transfers entire UTXOs off-chain via a trusted entity that co-signs transactions. Thunderbolt differs in three respects: (1) the committee never signs transfer messages; it only maintains a secret share, while validity is checked via public ZK proofs; (2) statechains require the entity to provably destroy its old co-signing key at each transfer, an unverifiable assumption; Thunderbolt eliminates this deletion assumption entirely via the (t, n)-threshold honest-majority assumption; (3) threshold instantiation distributes trust across n members, whereas statechains concentrate it in a single co-signer. Spark [40] likewise distributes the co-signer across threshold operators and supports leaf splitting, but still assumes that obsolete operator shares are forgotten and uses a per-transfer decrementing relative delay to order competing pre-signed exits. Because each delay is fixed when signed, a current holder or watchtower must respond if a stale exit is published. Section 5.4 sketches analogous exit and splitting overlays for Thunderbolt. Mercury Layer [19]

Table 3: Comparison of Thunderbolt with representative Bitcoin layer-2 (L2) systems by safety, recovery, holder response obligations, and settlement behavior.

System	Safety assumptions	Liveness / recovery	User response model	On-chain settlement behavior
Spark [40]	**Dependency:** Spark Operators **Trust:** t-of-n **Old-state safety:** fewer than t operators retain old shares	**Liveness:** t operators online **Recovery:** unilateral L1 exit	**Action:** holder publishes the latest leaf after a stale exit, possibly via a watchtower **Bound:** relative timelock	**Cooperative:** 2 transactions **Unilateral:** 2+ transactions; depth-dependent branch path plus timelocked leaf exit
Ark [1, 37]	**Dependency:** ASP **Trust:** trustless in-round; single ASP trusted out-of-round **Old-state:** no sender-ASP collusion	**Liveness:** ASP online; scheduled refresh round **Recovery:** unilateral L1 exit	**Action:** holder refreshes or exits before expiry **Bound:** VTXO expiry	**Cooperative:** 2 transactions **Unilateral:** 2+ transactions; tree depth plus one per out-of-round spend
Mercury [19]	**Dependency:** statechain operator **Trust:** single operator **Old-state safety:** superseded shares erased	**Liveness:** operator online **Recovery:** unilateral L1 exit	**Action:** holder publishes latest backup if operator fails **Bound:** decrementing absolute timelock	**Cooperative:** 2 transactions **Unilateral:** 1 transaction; latest pre-signed backup close
Lightning [51]	**Dependency:** None **Trust:** N/A **Old-state safety:** timely response to stale-state publication	**Liveness:** both channel parties online **Recovery:** unilateral L1 exit	**Action:** holder participates per payment and contests a stale close, possibly via watchtower **Bound:** dispute window	**Cooperative:** 2 transactions **Unilateral:** 1–2 transactions; commitment close plus optional stale-state contest
Thunderbolt	**Dependency:** threshold committee **Trust:** t-of-n **Old-state safety:** erasure-free threshold safety with fewer than t Byzantine members	**Liveness:** t honest committee members responsive **Recovery:** committee-dependent core; optional exit overlay	**Action:** holder retrieves and decrypts when online **Bound:** unbounded receipt	**Cooperative:** 2 transactions **Unilateral:** optional overlay extension

adds blinded signing but retains the single-entity trust model and key-deletion assumption. Table 3 makes these differences explicit across all five relevant systems.

Adaptor signatures. Adaptor signatures [4, 28] condition a pre-signature on the discrete log of a public point, used in PTLC-based routing and atomic swaps; Gerhart et al. [33] and Tairi et al. [60] formalize their security. Thunderbolt's verifiable encryption serves a related but distinct purpose: we encrypt the discrete log itself and prove consistency via Paillier homomorphism and VEDL proofs.

Verifiable encryption and secret sharing. Camenisch and Shoup [13] give a Sigma-protocol proof that a Paillier ciphertext encrypts the discrete log of a group element; Chen et al. [16] give a UC-secure NIZK compiler from Sigma protocols. Our Invoice and Transfer Consistency Proofs instantiate the Camenisch–Shoup technique, with the Invoice proof extending it to bind three objects via a single shared Sigma response. Shoup's Simple VESS [56] provides verifiable encryption of Shamir sharing [55], building on Feldman's VSS [29] and Pedersen commitments [49]; Das et al. [22] recently simplify VSS with reduced bandwidth. Thunderbolt uses VESS to distribute the committee-side refresh r_s under a (t, n)-threshold group at each transfer step.

Threshold signatures for Bitcoin. FROST [39] and ROAST [53] provide efficient threshold Schnorr [54] signing; Chu et al. [18] prove FROST secure without the algebraic group model. Das et al. [23] give practical asynchronous DKG, Thyagarajan et al. [61] address threshold-service payments, Lindell [41] gives two-party ECDSA, and Canetti et al. [15] a proactive threshold-ECDSA with UC security. These protocols are complementary to Thunderbolt and can finalize a channel when the output is a Taproot [63] key.

General-purpose zero-knowledge proofs. Groth16 [34], Plonk [31], transparent STARKs [8], and zkVM systems [52, 59] can prove any NP statement but pay a large constant for algebraic relations that Sigma protocols handle natively. Recursive composition [10] and accumulation [11] reduce verification costs but do not eliminate proving-time overhead for Paillier-heavy statements.

Privacy considerations. Thunderbolt's transfer proofs are publicly verifiable: any observer of the Thunderbolt Ledger can check π_{inv} and π_{xfer} without holding any secret key. Mercury Layer achieves operator-side unlinkability via blinded signing, but blinding is possible precisely because only the single entity verifies each transfer. Adapting blinded or anonymous credential techniques to a publicly verifiable, multi-verifier setting without sacrificing soundness remains an open direction.

8 Conclusion

We presented Thunderbolt, an off-chain Bitcoin protocol that transfers UTXO ownership asynchronously and recursively under a fixed on-chain lock key. Two Sigma-protocol NIZKs enforce the "one add, one subtract" invariant per step, while VESS distributes the committee share under a standard (t, n)-threshold honest-majority assumption. A single transfer achieves seconds-scale latency and 3.8 KB proof size; the threshold path scales linearly with committee size and degrades gracefully under adversarial conditions.

Future directions. Composed constructions (atomic cross-chain swaps, multi-channel designs) and geographically distributed deployments remain to be analyzed; Section 6.3 suggests VESS share generation will stay the bottleneck. State compaction by committee-signed checkpoints, and a routed graph of Thunderbolt channels combining Lightning's reach with asynchronous endpoints, are natural next steps.

References

[1] Ark Labs. 2024. The Ark Protocol. Technical documentation. https://assets.arklabs.xyz/ark-protocol.pdf

[2] Lukas Aumayr, Zeta Avarikioti, Matteo Maffei, and Subhra Mazumdar. 2024. Securing Lightning Channels against Rational Miners. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 393–407. doi:10.1145/3658644.3670373

[3] Lukas Aumayr, Oguzhan Ersoy, Andreas Erwig, Sebastian Faust, Kristina Hostáková, Matteo Maffei, Pedro Moreno-Sanchez, and Siavash Riahi. 2021. Bitcoin-Compatible Virtual Channels. In *2021 IEEE Symposium on Security and Privacy (S&P)*. IEEE, 901–918. doi:10.1109/SP40001.2021.00097

[4] Lukas Aumayr, Oguzhan Ersoy, Andreas Erwig, Sebastian Faust, Kristina Hostáková, Matteo Maffei, Pedro Moreno-Sanchez, and Siavash Riahi. 2021. Generalized Channels from Limited Blockchain Scripts and Adaptor Signatures. In *Advances in Cryptology — ASIACRYPT 2021 (LNCS, Vol. 13091)*. Springer, 635–664. doi:10.1007/978-3-030-92075-3_22

[5] Lukas Aumayr, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. 2023. Breaking and Fixing Virtual Channels: Domino Attack and Donner. In *30th Annual Network and Distributed System Security Symposium (NDSS)*. Internet Society. doi:10.14722/ndss.2023.24370

[6] Lukas Aumayr, Sri AravindaKrishnan Thyagarajan, Giulio Malavolta, Pedro Moreno-Sanchez, and Matteo Maffei. 2022. Sleepy Channels: Bi-directional Payment Channels without Watchtowers. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 179–192. doi:10.1145/3548606.3559370

[7] Mihir Bellare and Phillip Rogaway. 1993. Random Oracles Are Practical: A Paradigm for Designing Efficient Protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security (CCS)*. ACM, 62–73. doi:10.1145/168588.168596

[8] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. 2019. Scalable Zero Knowledge with No Trusted Setup. In *Advances in Cryptology — CRYPTO 2019 (LNCS, Vol. 11694)*. Springer, 701–732. doi:10.1007/978-3-030-26954-8_23

[9] Jan Bobolz, Elizabeth Crites, Markulf Kohlweiss, and Akira Takahashi. 2026. UC4Free! Existing Threshold Signatures are UC Secure. In *Advances in Cryptology — EUROCRYPT 2026 (LNCS)*. Springer. doi:10.1007/978-3-032-25291-3_3

[10] Sean Bowe, Jack Grigg, and Daira Hopwood. 2019. Recursive Proof Composition without a Trusted Setup. Cryptology ePrint Archive, Paper 2019/1021. https://eprint.iacr.org/2019/1021

[11] Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. 2020. Recursive Proof Composition from Accumulation Schemes. In *Theory of Cryptography — TCC 2020, Part II (LNCS, Vol. 12551)*. Springer, 1–18. doi:10.1007/978-3-030-64378-2_1

[12] Conrad Burchert, Christian Decker, and Roger Wattenhofer. 2018. Scalable Funding of Bitcoin Micropayment Channel Networks. In *Royal Society Open Science*, Vol. 5. doi:10.1098/rsos.180089

[13] Jan Camenisch and Victor Shoup. 2003. Practical Verifiable Encryption and Decryption of Discrete Logarithms. In *Advances in Cryptology — CRYPTO 2003 (LNCS, Vol. 2729)*. Springer, 126–144. doi:10.1007/978-3-540-45146-4_8

[14] Ran Canetti. 2001. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *42nd IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE, 136–145. doi:10.1109/SFCS.2001.959888

[15] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. 2020. UC Non-Interactive, Proactive, Threshold ECDSA with Identifiable Aborts. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 1769–1787. doi:10.1145/3372297.3423367

[16] Megan Chen, Pousali Dey, Chaya Ganesh, Pratyay Mukherjee, Pratik Sarkar, and Swagata Sasmal. 2025. Universally Composable Non-interactive Zero-Knowledge from Sigma Protocols via a New Straight-Line Compiler. In *Public-Key Cryptography — PKC 2025*. Springer, 381–417. doi:10.1007/978-3-031-91820-9_13

[17] Xiuyuan Chen, Junzuo Lai, Chao Lin, Xinyi Huang, and Debiao He. 2025. Cryptographic Primitives in Script-based and Scriptless Payment Channel Networks: A Survey. *Comput. Surveys* (2025), 1–34. doi:10.1145/3725846

[18] Hien Chu, Paul Gerhart, Tim Ruffing, and Dominique Schröder. 2023. Practical Schnorr Threshold Signatures Without the Algebraic Group Model. In *Advances in Cryptology — CRYPTO 2023 (LNCS, Vol. 14081)*. Springer, 743–773. doi:10.1007/978-3-031-38557-5_24

[19] CommerceBlock. 2023. Mercury Layer: Statechain Implementation with Blinded Signing. Project documentation. https://mercurylayer.com/

[20] Ronald Cramer. 1997. *Modular Design of Secure yet Practical Cryptographic Protocols*. Ph.D. Thesis. University of Amsterdam. https://ir.cwi.nl/pub/21438

[21] Ivan Damgård. 2010. On Σ-Protocols. Lecture notes, Aarhus University. https://www.cs.au.dk/~ivan/Sigma.pdf

[22] Sourav Das, Zhuolun Xiang, Alin Tomescu, Alexander Spiegelman, Benny Pinkas, and Ling Ren. 2025. Verifiable Secret Sharing Simplified. In *2025 IEEE Symposium on Security and Privacy (S&P)*. IEEE, 633–651. doi:10.1109/SP61157.2025.00046

[23] Sourav Das, Thomas Yurek, Zhuolun Xiang, Andrew Miller, Lefteris Kokoris-Kogias, and Ling Ren. 2022. Practical Asynchronous Distributed Key Generation. In *2022 IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2518–2534. doi:10.1109/SP46214.2022.9833584

[24] Christian Decker, Rusty Russell, and Olaoluwa Osuntokun. 2018. eltoo: A Simple Layer2 Protocol for Bitcoin. White paper. https://blockstream.com/eltoo.pdf

[25] Christian Decker and Roger Wattenhofer. 2015. A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels. In *Stabilization, Safety, and Security of Distributed Systems (SSS) (LNCS, Vol. 9212)*. Springer, 3–18. doi:10.1007/978-3-319-21741-3_1

[26] Stefan Dziembowski, Sebastian Faust, and Kristina Hostáková. 2018. General State Channel Networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 949–966. doi:10.1145/3243734.3243856

[27] Christoph Egger, Pedro Moreno-Sanchez, and Matteo Maffei. 2019. Atomic Multi-Channel Updates with Constant Collateral in Bitcoin-Compatible Payment-Channel Networks. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 801–815. doi:10.1145/3319535.3345666

[28] Andreas Erwig, Sebastian Faust, Kristina Hostáková, Monosij Maitra, and Siavash Riahi. 2021. Two-Party Adaptor Signatures from Identification Schemes. In *Public-Key Cryptography — PKC 2021 (LNCS, Vol. 12710)*. Springer, 451–480. doi:10.1007/978-3-030-75245-3_17

[29] Paul Feldman. 1987. A Practical Scheme for Non-interactive Verifiable Secret Sharing. In *28th Annual Symposium on Foundations of Computer Science (FOCS)*. 427–438. doi:10.1109/SFCS.1987.4

[30] Amos Fiat and Adi Shamir. 1987. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Advances in Cryptology — CRYPTO '86 (LNCS, Vol. 263)*. Springer, 186–194. doi:10.1007/3-540-47721-7_12

[31] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. 2019. PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. Cryptology ePrint Archive, Paper 2019/953. https://eprint.iacr.org/2019/953

[32] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. 2007. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. *Journal of Cryptology* 20, 1 (2007), 51–83. doi:10.1007/s00145-006-0347-3

[33] Paul Gerhart, Dominique Schröder, Pratik Soni, and Sri AravindaKrishnan Thyagarajan. 2024. Foundations of Adaptor Signatures. In *Advances in Cryptology — EUROCRYPT 2024 (LNCS, Vol. 14654)*. Springer, 161–189. doi:10.1007/978-3-031-58723-8_6

[34] Jens Groth. 2016. On the Size of Pairing-Based Non-interactive Arguments. In *Advances in Cryptology — EUROCRYPT 2016 (LNCS, Vol. 9666)*. Springer, 305–326. doi:10.1007/978-3-662-49896-5_11

[35] Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. 2020. SoK: Layer-Two Blockchain Protocols. In *Financial Cryptography and Data Security (FC) (LNCS, Vol. 12059)*. Springer, 201–226. doi:10.1007/978-3-030-51280-4_12

[36] Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S. Matthew Weinberg, and Edward W. Felten. 2018. Arbitrum: Scalable, private smart contracts. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, 1353–1370. https://www.usenix.org/conference/usenixsecurity18/presentation/kalodner

[37] Burak Keceli. 2023. Ark: An Alternative Privacy-preserving Second Layer Solution. Bitcoin-dev mailing list. https://gnusha.org/pi/bitcoindev/1300890009.1516890.1684742043892@eu1.myprofessionalmail.com/

[38] Rami Khalil and Arthur Gervais. 2017. Revive: Rebalancing Off-Blockchain Payment Networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 439–453. doi:10.1145/3133956.3134033

[39] Chelsea Komlo and Ian Goldberg. 2021. FROST: Flexible Round-Optimized Schnorr Threshold Signatures. In *Selected Areas in Cryptography (SAC) (LNCS, Vol. 12804)*. Springer, 34–65. doi:10.1007/978-3-030-81652-0_2

[40] Lightspark. 2025. Spark: A Statechain-Based Bitcoin Layer 2. Project documentation. https://docs.spark.money/

[41] Yehuda Lindell. 2017. Fast Secure Two-Party ECDSA Signing. In *Advances in Cryptology — CRYPTO 2017 (LNCS, Vol. 10402)*. Springer, 613–644. doi:10.1007/978-3-319-63715-0_21

[42] Robin Linus. 2023. BitVM: Compute Anything on Bitcoin. White paper. https://bitvm.org/bitvm.pdf

[43] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. 2017. Concurrency and Privacy with Payment-Channel Networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 455–471. doi:10.1145/3133956.3134096

[44] Matter Labs. 2026. ZKsync Docs. Project documentation. https://docs.zksync.io/

[45] Andrew Miller, Iddo Bentov, Surya Bakshi, Ranjit Kumaresan, and Patrick McCorry. 2019. Sprites and State Channels: Payment Networks that Go Faster than Lightning. In *Financial Cryptography and Data Security (FC) (LNCS, Vol. 11598)*. Springer, 508–526. doi:10.1007/978-3-030-32101-7_30

[46] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. White paper. https://bitcoin.org/bitcoin.pdf

[47] Offchain Labs. 2026. Arbitrum Docs. Project documentation. https://docs.arbitrum.io/

[48] Pascal Paillier. 1999. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Advances in Cryptology — EUROCRYPT '99 (LNCS, Vol. 1592)*. Springer, 223–238. doi:10.1007/3-540-48910-X_16

[49] Torben Pryds Pedersen. 1991. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Advances in Cryptology — CRYPTO '91 (LNCS, Vol. 576)*. Springer, 129–140. doi:10.1007/3-540-46766-1_9

[50] David Pointcheval and Jacques Stern. 1996. Security Proofs for Signature Schemes. In *Advances in Cryptology — EUROCRYPT '96 (LNCS, Vol. 1070)*. Springer, 387–398. doi:10.1007/3-540-68339-9_33

[51] Joseph Poon and Thaddeus Dryja. 2016. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. White paper. https://lightning.network/lightning-network-paper.pdf
[52] RISC Zero. 2026. RISC Zero Developer Docs. Project documentation. https://dev.risczero.com/
[53] Tim Ruffing, Viktoria Ronge, Elliott Jin, Jonas Schneider-Bensch, and Dominique Schröder. 2022. ROAST: Robust Asynchronous Schnorr Threshold Signatures. Cryptology ePrint Archive, Paper 2022/550. doi:10.1145/3548606.3560583
[54] Claus-Peter Schnorr. 1991. Efficient Signature Generation by Smart Cards. *Journal of Cryptology* 4, 3 (1991), 161–174. doi:10.1007/BF00196725
[55] Adi Shamir. 1979. How to Share a Secret. *Commun. ACM* 22, 11 (1979), 612–613. doi:10.1145/359168.359176
[56] Victor Shoup. 2025. Simple VESS. Cryptology ePrint Archive, Paper 2025/1175. https://eprint.iacr.org/2025/1175
[57] Ruben Somsen. 2018. Statechains: Off-chain Transfer of UTXO Ownership. White paper. https://github.com/RubenSomsen/rubensomsen.github.io/raw/master/img/statechains.pdf
[58] Jeremy Spilman. 2013. Anti DoS for tx replacement. Bitcoin-development mailing list. https://gnusha.org/pi/bitcoindev/CA+CODZEQX_xiaJE7WtFZC2qfVfDqZAgg-ydU5Q73O-QTkXJpPw@mail.gmail.com/
[59] Succinct Labs. 2026. SP1 Developer Docs. Project documentation. https://docs.succinct.xyz/docs/sp1/introduction
[60] Erkan Tairi, Pedro Moreno-Sanchez, and Clara Schneidewind. 2023. LedgerLocks: A Security Framework for Blockchain Protocols Based on Adaptor Signatures. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 859–873. doi:10.1145/3576915.3623149
[61] Sri AravindaKrishnan Thyagarajan, Easwar Vivek Mangipudi, Lucjan Hanzlik, Aniket Kate, and Pratyay Mukherjee. 2025. Vitārit: Paying for Threshold Services on Bitcoin and Friends. In *2025 IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2018–2036. doi:10.1109/SP61157.2025.00170
[62] Pieter Wuille, Jonas Nick, and Tim Ruffing. 2020. BIP 340: Schnorr Signatures for secp256k1. Bitcoin Improvement Proposals. https://github.com/bitcoin/bips
[63] Pieter Wuille, Jonas Nick, and Anthony Towns. 2020. BIP 341: Taproot: SegWit Version 1 Spending Rules. Bitcoin Improvement Proposals. https://github.com/bitcoin/bips

A Open Science

The artifacts needed to evaluate the core implementation and experimental claims of this submission are publicly available at https://github.com/RiemaLabs/thunderbolt/.

The repository contains the Thunderbolt implementation, the supporting benchmark code, and a reproducibility bundle under `artifacts/**`. In particular, it includes the code and scripts used for the single-committee microbenchmark, the scalability benchmark, the threshold-committee benchmark, and the reduced-parameter SP1 proof-cost experiment. The main reproduction entry point is `./artifacts/reproduce-eval.sh`, and the repository also provides the checked-in benchmark outputs used in the evaluation. No credentials are required for access; external toolchain dependencies for full reruns are documented in the repository.

B Ethical Considerations

This work presents a cryptographic protocol for off-chain Bitcoin transfers. The protocol employs only standard primitives (Paillier encryption, Schnorr-based Sigma protocols, Shamir secret sharing) that have been publicly studied for over two decades. No user data was collected or processed; all experiments use synthetic cryptographic inputs on a locally controlled machine. The protocol does not introduce new mechanisms for bypassing existing blockchain compliance or anti-money-laundering frameworks; it operates within the same trust model as existing custody and multi-signature deployments. We see no direct ethical hazards beyond those already present in any privacy-enhancing cryptocurrency technology.

C Acknowledgments

This paper was edited for grammar using Claude Opus 4.6 (Anthropic).

D Notation Summary

Table 4 collects the principal symbols used throughout the paper and in the proof details that follow (Appendix E).

E Proof Details

This section provides the full proofs of the theorems stated in the main body. We use the notation summarized in Table 4; proof-local variables (α, β, z, τ, c, etc.) are introduced where needed.

E.1 Proof of Theorem 4.2 (Invoice Consistency)

Standalone instantiation and scope. The Invoice and Transfer relations follow the verifiable-encryption-of-discrete- logarithms (VEDL) Sigma-protocol template of Camenisch and Shoup [13]. We therefore invoke their standard single-receiver argument, prove Thunderbolt's multi-receiver lift, and recall completeness and HVZK for concreteness. For this standalone instantiation, witnesses use canonical representatives $0 \le r_s, x_s < q$, the complete verifier rejects a response outside $0 \le z < \tilde{q} + q^2$, and accepted Paillier keys satisfy the key-validity conditions required by the cited VEDL construction. As in Camenisch–Shoup, the extractor is exact whenever the challenge difference divides the response difference and otherwise returns their standard relaxed witness; the elliptic-curve

Table 4: Summary of notation used throughout the paper.

Symbol	Meaning
$\mathbb{G}$	Elliptic-curve group (secp256k1), prime order q
G	Standard base point of $\mathbb{G}$ (lock key domain)
H	NUMS generator of $\mathbb{G}$ (anchor / delta domain)
$P = (x_s + y_s)\,G$	Fixed on-chain lock key
x_s	Holder secret at step s
y_s	Committee secret at step s
r_s	Refresh value chosen by the recipient at step s
$X_s = x_s\,H$	Public anchor at step s
$D_s = r_s\,H$	Public delta (anchor increment) at step s
F_0	VESS free-term commitment; public bridge $F_0 = D_s$
(N, g)	Paillier public key; $N = p_1 p_2$
$\mathrm{Enc}_{pk}(m;\rho)$	Paillier encryption of m with randomness ρ
$\oplus$, $\ominus$	Homomorphic ciphertext addition / subtraction
ct_B, ct_C	Invoice ciphertexts for Bob and Committee
T_s	Holder ciphertext at step s (encrypts x_s)
π_{inv}	Invoice Consistency Proof
π_{xfer}	Transfer Consistency Proof
κ	*Statistical* security parameter ($\kappa = 128$)
$\tilde{q} = q^2 \cdot 2^{\kappa}$	Masking range for Sigma commitment scalars
(t, n)	Threshold parameters for the committee
ctx_id	Per-channel context identifier
inv_id	Per-invoice identifier chosen by the recipient
UC security analysis (Section 5)	
λ	*Computational* security parameter (UC statements)
π_{TF}	Transfer-and-finalization logic of Thunderbolt
$\mathcal{F}_{\mathsf{TB}}$	Thunderbolt ideal functionality (Figure 3)
$\mathcal{H}_{\mathrm{std}}$	Hybrid model of five ideal components
$\mathcal{F}_{\mathrm{Ledger}}$, $\mathcal{F}_{\mathrm{PKI}}$	Ledger and key-registration components
$\mathcal{F}_{\mathrm{NIZK}}$, $\mathcal{F}_{\mathrm{VESS}}$	Proof and encrypted-sharing components
$\mathcal{F}_{\mathrm{TS}}^{\mathrm{Auth}}$	Authorized threshold-signing component
$\mathcal{R}_{\mathrm{own}}$, $\mathcal{R}_{\mathrm{xfer}}$	Ownership and transfer relations of $\mathcal{F}_{\mathrm{NIZK}}$
$\mathcal{R}_{\mathrm{inv}}^{\mathrm{thr}}$	Threshold invoice relation of $\mathcal{F}_{\mathrm{NIZK}}$
ctx	Signing context $(\mathrm{ctx_id}, s, P, X_s, \mathsf{tx})$
H_s	Holder at step s
$\mathcal{A}$, $\mathcal{Z}$	Adversary and environment
$\mathcal{S}_{\mathrm{sl}}$	Straight-line simulator (never rewinds)
HYB, IDEAL	Hybrid and ideal execution ensembles
$\Delta_{\mathcal{Z}}(\lambda)$	$\lvert\Pr[\mathsf{HYB} = 1] - \Pr[\mathsf{IDEAL} = 1]\rvert$
Forge	Forgery on an honest anchor (Lemma 5.3)

scalar is always extracted exactly, so the relaxation can only decouple a recipient-key Paillier plaintext from that scalar. Its sole effect is a mis-formed credential under the recipient's own key, which the recipient detects against the public anchor, while the committee-side binding runs through D_s (Section 4.6). These are primitive-level instantiation conditions; they are not assumptions made by the straight-line simulator of Theorem 5.1, which accesses the ideal $\mathcal{F}_{\mathrm{NIZK}}$ interface, and $\mathcal{F}_{\mathrm{NIZK}}$ admits no relaxation.

Proof. We establish the three standard properties.

Completeness. If the prover is honest, then $z = \alpha + c\,r_s$, $\tau_B = \beta_B \cdot \rho_B^c$, and $\tau_C = \beta_C \cdot \rho_C^c$. For the EC check: $z\,H = (\alpha + c\,r_s)\,H = \alpha\,H + c\,r_s\,H = A_D + c\,D_s$. For the Bob–Paillier check: $g_B^z\,\tau_B^{N_B} = g_B^{\alpha + c\,r_s}\,(\beta_B \cdot \rho_B^c)^{N_B} = (g_B^{\alpha}\,\beta_B^{N_B}) \cdot (g_B^{r_s}\,\rho_B^{N_B})^c = A_B \cdot \mathrm{ct}_B^c$. The Committee–Paillier check is analogous.

Special soundness. For either pair (D_s, ct_B) or (D_s, ct_C), two accepting transcripts with the same first message and distinct challenges give the standard single-receiver VEDL extractor of Camenisch–Shoup [13]. The Invoice proof runs these two instances in parallel with the same EC commitment, challenge, and response z. Hence both component extractors return the discrete logarithm bound to D_s; their respective Paillier randomness values complete one witness (r_s, ρ_B, ρ_C) for $\mathcal{R}_{\mathrm{inv}}$. This component-wise lifting is the only additional argument required by the second receiver.

Honest-verifier zero-knowledge. The simulator $\mathcal{S}$, given the statement $(D_s, \mathrm{ct}_B, \mathrm{ct}_C)$, samples $z \leftarrow \mathbb{Z}_{\tilde{q}}$, $\tau_B \leftarrow \mathbb{Z}_{N_B}^*$, $\tau_C \leftarrow \mathbb{Z}_{N_C}^*$, programs the random oracle to return c, and sets: $A_D = z\,H - c\,D_s$, $A_B = g_B^z\,\tau_B^{N_B} \cdot \mathrm{ct}_B^{-c} \bmod N_B^2$, $A_C = g_C^z\,\tau_C^{N_C} \cdot \mathrm{ct}_C^{-c} \bmod N_C^2$. The simulated transcript $(A_D, A_B, A_C, c, z, \tau_B, \tau_C)$ is statistically indistinguishable from an honest transcript (statistical distance at most $2^{-\kappa}$), since the masking range $\tilde{q}$ absorbs the shift $c \cdot r_s$.

Soundness in the non-interactive (Fiat–Shamir compiled) setting follows from the forking lemma [50] applied in the random-oracle model. □

E.2 Proof of Theorem 4.4 (Transfer Consistency)

Proof. Let $U = T_{s+1} \ominus \mathrm{ct}_B = T_{s+1} \cdot \mathrm{ct}_B^{-1} \bmod N_B^2$ for brevity.

Completeness. With $z = \alpha + c\,x_s$ and $\tau = \beta \cdot \eta_s^c$: $z\,H = \alpha\,H + c\,x_s\,H = A_X + c\,X_s$, and $g_B^z\,\tau^{N_B} = g_B^{\alpha}\,\beta^{N_B} \cdot (g_B^{x_s}\,\eta_s^{N_B})^c = A_T \cdot U^c$.

Special soundness. The Transfer proof is the single-receiver specialization of the same VEDL template, with (X_s, U) as the EC point and ciphertext. Its special soundness therefore follows directly from the cited extractor under the instantiation conditions above.

Honest-verifier zero-knowledge. The simulator samples $z \leftarrow \mathbb{Z}_{\tilde{q}}$, $\tau \leftarrow \mathbb{Z}_{N_B}^*$, programs c via the random oracle, and sets $A_X = z\,H - c\,X_s$ and $A_T = g_B^z\,\tau^{N_B} \cdot U^{-c} \bmod N_B^2$. The resulting transcript is statistically indistinguishable from an honest one (statistical distance at most $2^{-\kappa}$).

As with the Invoice proof, non-interactive soundness follows from the forking lemma [50]. □

Remark E.1 (Standalone versus UC proof). *The Camenisch–Shoup extractor and the standard Fiat–Shamir forking lemma [50] justify only the standalone realization of the two proof relations. They are not invoked by Theorem 5.1, which assumes the $\mathcal{F}_{\mathrm{NIZK}}$ interface. Existing straight-line UC compilers [16] motivate this modularization, but a concrete instantiation must separately check the compiler's setup, relation, and key requirements. Neither the UC simulator nor the UC hybrid argument below rewinds or forks the environment or adversary.*

E.3 Proof of Theorem 4.6 (VESS Modularity)

Proof. The Transfer Consistency Proof π_{xfer} depends only on D_s, ct_B, X_s, and T_{s+1}; none of these are affected by the VESS substitution, so π_{xfer} remains sound and zero-knowledge as before. In the threshold setting, π_{inv} is run on the two-receiver relation (D_s, ct_B), which is a strict sub-relation of $\mathcal{R}_{\mathrm{inv}}$; its soundness and ZK properties follow by the same shared-z argument of Theorem 4.2. The committee-side binding is provided instead by the VESS consistency proof: VESS verification guarantees that the encrypted shares are evaluations of a unique degree-$(t-1)$ polynomial f with $f(0) = r_s$, where $r_s\,H = F_0$ is the public free-term commitment. A malicious

Bob who distributes inconsistent shares is detected by any observer, while the protocol's explicit public check $F_0 = D_s$ rejects an invoice that is not bound to its delta point. Correctness of the committee-side update follows from the Shamir-preservation argument above. Secrecy of y_s and r_s holds information-theoretically as long as fewer than t members are corrupted. □

F Full Proof of Theorem 5.1

We prove Lemmas 5.2 and 5.3, which together imply Theorem 5.1. The proof starts from the correctly opened state specified in the theorem, so Open is an initialization condition rather than a transition simulated below. We write HYB for the $\mathcal{H}_{\text{std}}$-hybrid execution and IDEAL for the execution with $(\mathcal{F}_{\text{TB}}, \mathcal{S}_{\text{sl}})$. The interfaces are those of Section 5.1; three of their properties carry the argument below and are worth restating. The simulator generates the key pairs of the simulated honest parties, so it knows their decryption keys exactly. Every call to $\mathcal{F}_{\text{NIZK}}$, $\mathcal{F}_{\text{VESS}}$, and $\mathcal{F}_{\text{TS}}^{\text{Auth}}$ binds the complete $(\text{ctx_id}, s, \text{inv_id})$ context, so concurrent sessions are handled independently. And $\mathcal{F}_{\text{TS}}^{\text{Auth}}$ is assumed as an explicit authorization wrapper: we do not claim that the cited threshold-signature functionalities enforce an ownership condition on their own.

F.1 Proof of Lemma 5.2: Straight-Line Simulation

Throughout this subsection, $\mathcal{A}$ is an arbitrary PPT adversary within the corruption bound of Theorem 5.1. The simulator $\mathcal{S}_{\text{sl}}$ announced above runs $\mathcal{A}$ once in the forward direction. It starts from the correctly opened public state and the local states generated during Open for simulated honest parties, and maintains $(\text{ctx_id}, s, X_s)$, the pending-invoice set and recipient map, the honest holder's credential when the holder is honest, and the honest committee shares. It gives $\mathcal{A}$ the same public entries and the same scheduling choices as $\mathcal{F}_{\text{Ledger}}$.

Honest operations. For an honest invoice creator, $\mathcal{S}_{\text{sl}}$ samples r_s exactly as the protocol does, creates the Paillier and VESS objects, and invokes the ideal proof and sharing interfaces. For an honest sender, it forms T_{s+1} using the maintained x_s and the recipient's public key. Consequently, a corrupted recipient decrypts the same correctly distributed next credential as in the hybrid execution. Honest finalization uses the authorized-signing interface. Each operation is submitted to the ideal functionality only when its ledger entry is committed.

Corrupted invoice creator. Suppose a corrupted recipient submits an invoice accepted by the proof and sharing interfaces and by the public check $F_0 = D_s$. Dealing to $\mathcal{F}_{\text{VESS}}$ is an input action, not a proof: the ideal sharing interface accepts a degree-$(t-1)$ polynomial from its dealer and nothing else, so a corrupted dealer whose invoice was accepted has already handed that polynomial f over. Because $\mathcal{S}_{\text{sl}}$ emulates the interface that received it, the free term $r_s = f(0)$ lies in an input it already holds. This is one forward step; no proof transcript is extracted, and nothing is rewound. The public check $F_0 = D_s$ forces $D_s = r_s H$, and $\mathcal{S}_{\text{sl}}$ records the resulting honest committee update. If $\mathcal{F}_{\text{VESS}}$ is instantiated by a concrete VESS scheme, no polynomial is handed over; $\mathcal{S}_{\text{sl}}$ then decrypts the shares of any t honest members under the keys it generated for them and interpolates

$$r_s = \sum_{i \in S} \lambda_i r_s^{(i)},$$

with λ_i the Lagrange coefficients of the decrypted set S. This is available whenever at least t members are honest, a condition on the concrete scheme discharged at composition (Corollary 5.4).

Corrupted sender. The ledger state determines whether the sender is the current holder and whether the referenced invoice remains pending. If the recipient is honest, $\mathcal{S}_{\text{sl}}$ decrypts the received credential directly:

$$x_{s+1} = \text{Dec}_{sk_{\text{H}_{s+1}}}(T_{s+1}), \qquad x_{s+1}H \stackrel{?}{=} X_s + D_s.$$

It stores x_{s+1} as that honest recipient's next local state. If the recipient is corrupted, the simulator need not learn the plaintext: membership of the accepted transfer statement, the public anchor check, and the ordered ledger state already determine whether the ideal transfer is enabled; for a corrupted holder, $\mathcal{S}_{\text{sl}}$ submits the corresponding ideal input on that party's behalf. Stale, malformed, and replayed entries are rejected in both executions.

Finalization and concurrency. For the current context, $\mathcal{S}_{\text{sl}}$ relays the ownership-proof result and signer set to $\mathcal{F}_{\text{TS}}^{\text{Auth}}$, which releases a signature only when an accepted $\mathcal{R}_{\text{own}}$ proof for the anchor of the current step is joined by a signer set of at least t members. Proposed, committed, delayed, and censored ledger entries are exposed in the same order in both executions; an uncommitted entry changes neither state.

Every case consumes one committed ledger entry, performs only local computation, ideal functionality calls, or decryption with an honest party's key, and then advances the state. The simulator never rewinds or forks $\mathcal{A}$ or $\mathcal{Z}$. Arbitrarily interleaved sessions are handled independently because every call carries its full channel, step, and invoice context.

Edges and induction. In the $\mathcal{H}_{\text{std}}$-hybrid, malformed proofs, inconsistent sharings, unauthorized signer sets, and invalid ledger transitions are rejected by the corresponding ideal interfaces. Honest ciphertexts and other public objects are generated with the real algorithms and their real distributions. For each committed entry, the checks above therefore enable exactly the same single state transition of $\mathcal{F}_{\text{TB}}$ as in the hybrid protocol. Each case is thus one edge Edge_j: it preserves the trace invariant $\mathcal{R}_j$ of Section 5.2, and induction over the committed trace $e_1, \ldots, e_m$ yields $\mathcal{R}_m$.

The forgery event. One gap remains. The hybrid protocol identifies the party entitled to transfer or finalize at step s by *knowledge of* x_s, whereas $\mathcal{F}_{\text{TB}}$ identifies it by party identity. The two notions coincide unless $\mathcal{A}$ supplies $\mathcal{F}_{\text{NIZK}}$ with a valid witness for $\mathcal{R}_{\text{own}}$ or $\mathcal{R}_{\text{xfer}}$ against the *current* anchor X_s of an *honest* holder; this is exactly the event Forge of Section 5.2. Conditioned on ¬Forge, the two executions are identically distributed by the paragraph above, so

$$\Delta_{\mathcal{Z}}(\lambda) \;\le\; \Pr[\text{Forge}],$$

which proves Lemma 5.2.

F.2 Proof of Lemma 5.3: Bounding Pr[Forge]

Let Q be a polynomial bound on the number of steps and let $\mathcal{B}$ be given an ECDLP challenge (H, Y) in $\mathbb{G}$. $\mathcal{B}$ guesses the step $s^* \le Q$ at which Forge first occurs, losing a factor $1/Q$, and runs $\mathcal{S}_{\text{sl}}$ with two changes. If the honest holder at step s^* is the initial holder, $\mathcal{B}$ embeds Y as X_{s^*}; otherwise that holder acquired ownership at the preceding transfer by choosing the refresh value itself, and $\mathcal{B}$ embeds Y as the corresponding D_{s^*-1}, so that $X_{s^*} = X_{s^*-1} + Y$. In both cases $\mathcal{B}$ no longer knows the honest holder's credential, and it replaces the honest objects that would require it as follows. Paillier ciphertexts under honest keys (the holder ciphertext T_{s^*} and, in the second case, the invoice ciphertext ct_B carrying the refresh value) are replaced by encryptions of 0 under the same honest keys; this change is indistinguishable by the semantic security of Paillier, since the decryption keys belong to simulated honest parties and $\mathcal{A}$ never receives them. Proofs the honest holder would produce are supplied by $\mathcal{F}_{\text{NIZK}}$ without a witness, hence perfectly simulated. The VESS dealing needs no computational replacement: $\mathcal{F}_{\text{VESS}}$ reveals to $\mathcal{A}$ only the at most $t-1$ corrupted members' shares and the public commitments; the shares are evaluations of a random degree-$(t-1)$ polynomial and are information-theoretically independent of the free term, so $\mathcal{B}$ samples them uniformly and fixes the base-H Feldman commitments by interpolation in the exponent through $F_0 = Y$ and the sampled shares. Hence $\mathcal{A}$'s view is computationally unchanged, and its entire view of the honest credential reduces to the group element Y. If Forge occurs at step s^*, the witness $\mathcal{A}$ submits to $\mathcal{F}_{\text{NIZK}}$, the scalar x_{s^*} of $\mathcal{R}_{\text{own}}$ or $\mathcal{R}_{\text{xfer}}$, is handed to $\mathcal{B}$ by that interface and yields $\log_H Y$: directly in the first case, and as $x_{s^*} - x_{s^*-1}$ in the second, where $\mathcal{B}$ knows the previous credential x_{s^*-1} because it either maintains it (honest previous holder) or read it from the $\mathcal{F}_{\text{NIZK}}$ witness of the preceding transfer (corrupted previous holder). Therefore $\Pr[\text{Forge}] \le Q \cdot (\text{Adv}^{\text{ECDLP}} + \text{Adv}^{\text{IND-CPA}}_{\text{Paillier}})$, which is negligible, proving Lemma 5.3; with Lemma 5.2, Theorem 5.1 follows. The standalone forking lemmas of Appendix E.1 and Appendix E.2 are used only to establish the underlying Sigma relations; they are never called by this UC simulator. □