Appendix

A. Formal Cryptographic Definitions

This appendix provides complete formal definitions omitted from the main paper.

1) Signature Schemes:

Definition 3 (Signature Scheme)**.** *A signature scheme* SIG *with a message space* $\mathbb{M}$ *consists of the following three PPT algorithms.*

- $\mathsf{KG}(1^\lambda) \to (\mathsf{vk}, \mathsf{sk})$*: The key generation algorithm, given a security parameter* 1^λ*, outputs a verification key* vk *and a signing key* sk*.*
- $\mathsf{Sign}(\mathsf{sk}, m) \to \sigma$*: The signing algorithm, given a signing key* sk *and a message* m*, outputs a signature* σ*.*
- $\mathsf{Ver}(\mathsf{vk}, m, \sigma) \to 1/0$*: The verification algorithm, given a verification key* vk*, a message* m*, and a signature* σ*, outputs either* 1 *(accept) or* 0 *(reject).*

Required Properties:

- *Correctness. For all* $\lambda \in \mathbb{N}$ *and* $m \in \mathbb{M}$*, if we set* $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KG}(1^\lambda)$*, then we always have* $\mathsf{Ver}(\mathsf{vk}, m, \mathsf{Sign}(\mathsf{sk}, m)) = 1$*.*
- *EUF-CMA Security. For any PPT adversary* $\mathcal{A}$*, the advantage*

$$\mathsf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{SIG},\mathcal{A}}(\lambda) := \Pr\left[\begin{array}{l} L_{sig} := \emptyset, \\ (\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KG}(1^\lambda), \\ (m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}_{sign}}(\mathsf{vk}) : \\ \mathsf{Ver}(\mathsf{vk}, m^*, \sigma^*) = 1 \\ \wedge\ m^* \notin L_{sig} \end{array}\right]$$

is negligible, where the signing oracle $\mathcal{O}_{sign}$*, on query* m*, computes* $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$*, returns* σ *to* $\mathcal{A}$*, and appends* m *to* L_{sig}*.*

2) Vector Commitment Schemes:

Definition 4 (Vector Commitment Scheme)**.** *A vector commitment scheme* VC *consists of the following four algorithms.*

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$*: Takes security parameter* λ *and outputs public parameters* pp*.*
- $\mathsf{Com}(\mathsf{pp}, \mathsf{vec}) \to (\mathsf{com}, \mathsf{aux})$*: Takes public parameters* pp *and an ordered vector* $\mathsf{vec} = (m_1, \ldots, m_N)$ *of length* N*, outputs a commitment* com *and auxiliary information* aux*.*
- $\mathsf{Open}(\mathsf{pp}, \mathsf{aux}, \mathsf{pos}) \to \mathsf{open}$*: Takes public parameters* pp*, auxiliary information* aux*, and position* $\mathsf{pos} \in [N]$*, outputs an opening proof* open *for the value at position* pos*.*
- $\mathsf{Ver}(\mathsf{pp}, \mathsf{com}, \mathsf{pos}, m, \mathsf{open}) \to \{0, 1\}$*: Takes public parameters* pp*, commitment* com*, position* pos*, value* m*, and opening proof* open*, outputs 1 (accept) or 0 (reject).*

Required Properties:

- *Correctness. For all security parameters* $\lambda \in \mathbb{N}$*, vectors* $\mathsf{vec} = (m_1, \ldots, m_N)$*, and positions* $\mathsf{pos} \in [N]$*, if* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$*,* $(\mathsf{com}, \mathsf{aux}) \leftarrow \mathsf{Com}(\mathsf{pp}, \mathsf{vec})$*, and* $\mathsf{open} \leftarrow \mathsf{Open}(\mathsf{pp}, \mathsf{aux}, \mathsf{pos})$*, then* $\mathsf{Ver}(\mathsf{pp}, \mathsf{com}, \mathsf{pos}, m_{\mathsf{pos}}, \mathsf{open}) = 1$*.*
- *Position Binding. For any PPT adversary* $\mathcal{A}$*,*

$$\mathsf{Adv}^{\mathsf{bind}}_{\mathsf{VC},\mathcal{A}}(\lambda) := \Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), \\ (\mathsf{com}, \mathsf{pos}, m, m', \mathsf{open}, \mathsf{open}') \leftarrow \mathcal{A}(\mathsf{pp}) : \\ m \neq m' \wedge \mathsf{Ver}(\mathsf{pp}, \mathsf{com}, \mathsf{pos}, m, \mathsf{open}) = 1 \\ \wedge \mathsf{Ver}(\mathsf{pp}, \mathsf{com}, \mathsf{pos}, m', \mathsf{open}') = 1 \end{array}\right]$$

is negligible.

Instantiation via Merkle Trees. We instantiate vector commitments using Merkle trees [21]. For blockchain applications, elements are tagged with block height h and position i. The commitment com is the Merkle root, and openings open are authentication paths of size $O(\log N)$. Position binding follows from collision resistance of hash function H.

Collision Resistance. For any PPT adversary $\mathcal{A}$, the collision-finding advantage

$$\mathsf{Adv}^{\mathsf{coll}}_{H,\mathcal{A}}(\lambda) := \Pr\left[(x, x') \leftarrow \mathcal{A}(1^\lambda) : x \neq x' \wedge H(x) = H(x')\right]$$

is negligible. For standard hash functions (e.g., SHA-256), this holds under the assumption that H is modeled as a random oracle or satisfies a concrete hardness assumption.

3) Two-Phase Commit Protocol: Two-phase commit (2PC) ensures atomicity across distributed participants. SATP uses 2PC to coordinate final state updates on both ledgers.

Definition 5 (Two-Phase Commit)**.** *A two-phase commit protocol between coordinator* C *and participants* $P_1, \ldots, P_n$ *consists of the following two phases.*

Prepare Phase*: Coordinator* C *sends* $\mathsf{Prepare}(\mathsf{tx})$ *to all participants. Each participant* P_i *responds* $\mathsf{Vote}(\mathsf{yes})$ *if it can commit* tx*, or* $\mathsf{Vote}(\mathsf{no})$ *otherwise. Participants voting yes enter prepared state and lock resources.*

Commit Phase*:*

- *If all participants respond* yes*, coordinator sends* $\mathsf{Commit}(\mathsf{tx})$ *to all. Each participant commits* tx *and releases locks (**global commit**).*
- *If any participant responds* no *or times out, coordinator sends* $\mathsf{Abort}(\mathsf{tx})$ *to all. Each participant aborts* tx *and releases locks (**global abort**).*

Required Property:

- *Atomicity. 2PC ensures either all participants commit or all abort. In SATP, gateways* G_A *and* G_B *coordinate to ensure final state is either (locked in* $\mathcal{L}_A$*, credited in* $\mathcal{L}_B$*) or unchanged.*

2PC Failure Model and Recovery. Standard 2PC has a well-known blocking problem: if the coordinator crashes after sending $\mathsf{Prepare}$ but before making a commit decision, participants that voted yes remain blocked indefinitely, unable to commit or abort unilaterally. Enhanced

SATP avoids this failure mode by not relying on an off-chain 2PC coordinator for completion: it replaces coordinator decisions with ledger-enforced validation rules over time-bounded escrows and publicly verifiable evidence.

1) **Time-bounded escrow**: Lock transactions include an expiry time t_{commit}.
2) **Ledger-enforced refund**: After t_{commit}, the source ledger permits refund if the escrow is not finalized.
3) **Ledger-enforced finalization**: Before expiry, the source ledger permits escrow consumption only upon an accepting destination-credit proof.

This design ensures that funds are never permanently blocked due to off-chain coordinator failure: either the destination credit occurs and the escrow can be finalized, or the timeout enables a refund. The timeout must satisfy $t_{\mathsf{commit}} > t_{\mathsf{proof}} + T$ to allow sufficient time for proof verification, relay, and confirmation.

4) Permissioned Distributed Ledgers: Following Pass et al. [24], we model each ledger as a protocol $(\Pi, \mathcal{L})$ run by a committee of n parties. Our analysis treats the consensus protocol as a black box and relies only on the abstract properties defined below (Δ-consistency and T-liveness), rather than on a particular fault model or consensus family.

Let $\mathsf{view} \leftarrow \mathsf{EXEC}^{(\Pi,\mathcal{L})}(\mathcal{A}, \mathcal{Z}, \lambda)$ be the combined view of all parties, $|\mathsf{view}|$ the number of rounds, $\mathsf{view}^{(r)}$ the view up to round r, and $\mathsf{st}_i(\mathsf{view})$ the state of committee i. Define $\mathcal{L}_i(\mathsf{view}) := \mathcal{L}(\lambda, \mathsf{st}_i(\mathsf{view}))$ and $\mathcal{L}_i^{(r)}(\mathsf{view}) := \mathcal{L}(\lambda, \mathsf{st}_i(\mathsf{view}^{(r)}))$.

Definition 6 (Permissioned Distributed Ledger)**.** *A permissioned distributed ledger $(\Pi, \mathcal{L})$ consists of a protocol Π and a state extraction function $\mathcal{L}$.*

Required Properties:

- *Δ-Consistency. For any PPT adversary $\mathcal{A}$:*

$$\mathsf{Adv}^{\mathsf{consist}}_{(\Pi,\mathcal{L}),\mathcal{A}} := \Pr\left[\begin{array}{c}\mathsf{view} \leftarrow \mathsf{EXEC}^{(\Pi,\mathcal{L})}(\mathcal{A}, \mathcal{Z}, \lambda) : \\ \mathsf{consist}(\mathsf{view}, \Delta(\lambda)) = 0\end{array}\right]$$

  is negligible, where $\mathsf{consist}(\mathsf{view}, \Delta) = 1$ if for all $r \le |\mathsf{view}| - \Delta$, honest parties i in round r, and positions $\mathsf{pos} \in [|\mathcal{L}(\mathsf{view}_i^{(r)})|]$, if message m is at position pos in $\mathcal{L}_i^{(r)}(\mathsf{view})$, then for all rounds $r' \ge r + \Delta$ and honest parties j, m is at position pos in $\mathcal{L}_j^{(r')}(\mathsf{view})$.

- *T-Liveness. For any PPT adversary $\mathcal{A}$:*

$$\mathsf{Adv}^{\mathsf{live}}_{(\Pi,\mathcal{L}),\mathcal{A}} := \Pr\left[\begin{array}{c}\mathsf{view} \leftarrow \mathsf{EXEC}^{(\Pi,\mathcal{L})}(\mathcal{A}, \mathcal{Z}, \lambda) : \\ \mathsf{live}(\mathsf{view}, T) = 0\end{array}\right]$$

  is negligible, where $\mathsf{live}(\mathsf{view}, T) = 1$ if for consecutive rounds $r, \ldots, r + T$ in view, if honest party i receives message m in round $r' \in [r, r + T]$, then all honest committee members j have m in $\mathcal{L}_j^{(r+T)}(\mathsf{view})$.

B. Complete Security Proofs

This section provides complete proofs for security theorems.

1) Proof of Enhanced SATP Correctness:

Complete proof of Theorem 2. Assume (A1), (A5), and (A6), and assume SIG and VC are correct. Consider an execution with an honest sender u_A and protocol-following gateways.

By (A1), the lock transaction $\mathsf{tx}_A := \mathsf{Lock}(u_A, \mathsf{asset}, v, \mathsf{id}, \mathcal{L}_B, u_B, t_{\mathsf{commit}})$ submitted to $\mathcal{L}_A$ is included and becomes final within bounded time. By (A5) and correctness of SIG and VC, any honest committee member that observes tx_A in a k-deep block can generate an attestation π_ℓ that passes $\mathsf{ProofVer}$; hence an honest gateway can collect $\ge f{+}1$ attestations and deliver an accepting lock evidence bundle to the destination.

If the lock evidence is accepted before t_{commit}, then by (A6) the destination credit transaction is submitted, and by (A1) it is included on $\mathcal{L}_B$. Once the destination credit becomes final, the destination committee can similarly generate a $\ge f{+}1$ credit evidence bundle, and by (A6) a corresponding $\mathsf{Finalize}$ is submitted to $\mathcal{L}_A$ and included. If the lock evidence is not accepted before t_{commit}, then by (A6) a refund transaction is submitted after timeout and included by (A1). In all cases, the protocol reaches a terminal outcome as prescribed, proving correctness. □

2) Proof of Asset-Conserving Atomicity:

Complete proof of Theorem 3. We prove asset-conserving atomicity as stated: for each id, the execution reaches exactly one terminal outcome (destination credit and source escrow consumption) or (source refund and no destination credit), except with negligible probability.

We structure the proof as follows: (1)–(2) establish that credits and consumptions require real counterpart transactions (soundness); (3) establishes mutual exclusion on the source ledger; (4)–(5) rule out the two "bad" terminal combinations (credit+refund and refund+credit). Since any execution must end in one of the four logical combinations—{credit, no-credit}×{consume/finalize, refund}—and (4)–(5) eliminate two, exactly one of the two "good" outcomes remains.

(1) No destination credit without a real source escrow lock. By on-chain enforcement, any destination credit requires an accepting lock proof under $\mathsf{ProofVer}$. Any accepting proof set contains $\ge f{+}1$ attestations, hence at least one honest committee member attests. By Theorems 4 and 6, except with probability $\mathsf{negl}(\lambda)$ this implies existence of a finalized source escrow/lock transaction whose contents match id and the agreed parameters. Furthermore, by Lemma 1, each id can be credited at most once due to the nullifier set maintained by the destination ledger.

(2) No source escrow consumption without a real destination credit. Similarly, consuming the escrow on $\mathcal{L}_A$ requires an accepting destination credit proof under $\mathsf{ProofVer}$, which (by the same evidence soundness guaran-

tees) implies that the destination credit transaction exists in a finalized state.

(3) Mutual exclusion on the source ledger. By the source ledger rules, refund is permitted only after t_{commit} and only if the escrow is not yet consumed; consumption makes refund impossible. Thus, for each id, at most one of {consume, refund} can occur on the source ledger.

(4) No "credit+refund" under liveness and submission. Assume a destination credit occurs before t_{commit}. By T-liveness, honest committee members can observe the finalized credit and G_A can submit a valid consumption transaction to the source ledger, which is included within the liveness bound. With t_{commit} chosen larger than the worst-case confirmation and relay delay, consumption occurs before refund becomes available, ruling out the terminal state where both destination credit and source refund occur.

(5) No "refund+credit". If refund occurs, then by (3) the escrow was not consumed by t_{commit}. If a destination credit had occurred, then by (4) consumption would have occurred before refund, contradiction.

Conclusion. The four logical terminal combinations are: (A) credit+consume, (B) credit+refund, (C) no-credit+consume, (D) no-credit+refund. By (1), combination (C) is impossible (consume requires credit proof). By (4), combination (B) is impossible. By (5), if refund occurs then no credit occurred, so (B) is again ruled out from the refund side. Hence, except with probability $\mathsf{negl}(\lambda)$, exactly one of (A) or (D) holds for each id. □

3) Proof of Enhanced SATP Unforgeability:

Complete proof of Theorem 4. We prove unforgeability by comparison with Theorem 1. The original SATP proves unforgeability under the assumption that gateways are honest (users may be malicious), showing that malicious users cannot forge gateway signatures. The enhanced protocol achieves a strictly stronger property: unforgeability holds even when gateways and up to f committee members are malicious. We analyze the differential security mechanism.

This proof uses the advantage notations defined in Sections A1–A2: $\mathsf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{SIG},\mathcal{A}}(\lambda)$ for EUF-CMA security and $\mathsf{Adv}^{\mathsf{coll}}_{H,\mathcal{A}}(\lambda)$ for collision resistance. We additionally define $\mathsf{Adv}^{\mathsf{unf}}_{\mathcal{A}}(\lambda)$ as the probability that adversary $\mathcal{A}$ produces a valid proof for a transaction tx^* that does not exist in a finalized ledger state.

Threat model comparison. In the original SATP (Theorem 1), gateways are trusted; malicious users cannot forge gateway signature σ_A on unauthorized transaction tx^*_A due to EUF-CMA security, with advantage $\leq \mathsf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{SIG},\mathcal{A}}(\lambda)$. In Enhanced SATP, gateways and up to f committee members may be malicious. The adversary $\mathcal{A}$ can corrupt both G_A, G_B and f committee members, attempting to forge proofs for false ledger states. The key differential is that the original SATP's single gateway signature fails when gateways are untrusted, whereas the enhanced protocol requires $\geq f+1$ committee signatures with Merkle proofs, ensuring binding to actual ledger state.

Honest majority. With $n \geq 3f+1$ and $|\mathcal{C}_{\mathsf{corrupt}}| \leq f$, at least $2f+1$ committee members are honest. Any valid proof set must include at least one proof from an honest member $C_j \in \mathcal{H}$.

Forgery analysis. Let $\mathcal{E}_{\mathrm{forge}}$ denote the event that $\mathcal{A}$ produces valid proofs for false tx^* not in $\mathcal{L}_A$. We bound $\Pr[\mathcal{E}_{\mathrm{forge}}]$ by case analysis:

1) **Case 1 (Signature forgery).** If $\mathcal{A}$ forges an honest member's signature σ^*_j on $(\mathsf{root}^*_h, h^*, i^*)$ under pk_j, we build a reduction $\mathcal{B}$ to EUF-CMA: $\mathcal{B}$ receives vk^*, assigns $\mathsf{pk}_j := \mathsf{vk}^*$, answers signing queries for all other keys, and outputs $\big((\mathsf{root}^*_h, h^*, i^*), \sigma^*_j\big)$ as a forgery. Thus $\Pr[\text{sig forge}] \leq \mathsf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{SIG},\mathcal{B}}(\lambda)$.
2) **Case 2 (Merkle equivocation).** If $\mathcal{A}$ provides a valid opening open^* for a false $\mathsf{tx}^* \neq \mathsf{tx}$ at position (h^*, i^*) under root root^*_h, then $\mathcal{A}$ found a collision in H, violating position binding: $\Pr \leq \mathsf{Adv}^{\mathsf{coll}}_{H,\mathcal{A}}(\lambda)$.
3) **Case 3 (Replay).** Transactions include a unique transfer identifier id (binding destination/recipient/expiry), so replaying an old proof for a different transfer changes the signed/Merkle-verified payload and is rejected by $\mathsf{ProofVer}$; replay does not yield a successful forgery.

Union bound and reduction tightness. The reduction embeds the EUF-CMA challenge key as one of the $n-f$ honest committee members' keys, guessing which member $\mathcal{A}$ will target. This guess succeeds with probability $1/(n-f)$, introducing a multiplicative loss factor. Combining all cases:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{unf}}_{\mathcal{A}}(\lambda) &= \Pr[\mathcal{E}_{\mathrm{forge}}] \\ &\leq (n-f) \cdot \mathsf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{SIG},\mathcal{B}}(\lambda) + \mathsf{Adv}^{\mathsf{coll}}_{H,\mathcal{A}}(\lambda) \\ &= \mathsf{negl}(\lambda).\end{aligned}$$

The $(n-f)$ factor is a standard loss in threshold signature security proofs and remains polynomial for practical committee sizes (e.g., $n \leq 50$). □

4) Proof of Independent Verifiability:

Complete proof of Theorem 5. We prove that Enhanced SATP satisfies independent verifiability (Theorem 5) under the assumption of authenticated key distribution (Definition 1).

(1) Deterministic verification without trust. Let $\pi = (\mathsf{tx}, \{(\mathsf{pk}_\ell, \pi_\ell)\}_{\ell \in S}, \sigma_{AB}, t_{\mathsf{proof}})$ denote an evidence bundle. The verification algorithm $\mathsf{ProofVer}(\mathsf{pp}, \pi)$ performs the following checks:

1) Threshold check: $|S| \geq f+1$
2) For each $\pi_\ell = (\mathsf{tx}, \mathsf{root}_h, h, i, \mathsf{open}, \sigma_\ell)$ and corresponding pk_ℓ:
   - Signature verification: $\mathsf{Ver}(\mathsf{pk}_\ell, (\mathsf{root}_h, h, i), \sigma_\ell) = 1$

- Merkle verification: $\mathsf{MerkleVer}(\mathsf{root}_h, \mathsf{tx}, i, \mathsf{open}) = 1$

3) Gateway agreement: $\mathsf{Ver}(\mathsf{pk}_{G_A}, (t_{\mathsf{proof}}, t_{\mathsf{commit}}, \mathsf{id}), \sigma_A) = 1$ and $\mathsf{Ver}(\mathsf{pk}_{G_B}, (t_{\mathsf{proof}}, t_{\mathsf{commit}}, \mathsf{id}), \sigma_B) = 1$

All checks are deterministic functions of the inputs (pp, π). No ledger access is required: Merkle roots are included in proofs, not fetched from ledgers. No gateway trust is required: verification uses committee signatures, not gateway assertions. Thus, any two verifiers V_1, V_2 with identical inputs compute identical outputs.

(2) Soundness. Suppose $\mathsf{ProofVer}(\mathsf{pp}, \pi) = 1$ for an evidence bundle π that contains transaction tx. We show tx exists in a final state except with negligible probability. By Theorem 4, forging a valid proof set requires either (a) forging an honest committee member's signature, or (b) finding a Merkle collision. Both occur with probability $\leq (n-f) \cdot \mathsf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{SIG},\mathcal{B}}(\lambda) + \mathsf{Adv}^{\mathsf{coll}}_{H,\mathcal{A}}(\lambda) = \mathsf{negl}(\lambda)$.

Since the proof set is not forged, at least one proof π_j came from an honest committee member C_j who followed Definition 7. By this definition, C_j only signs when tx appears at position (h, i) in C_j's local view and block h is k-deep (final). By Δ-consistency, a k-deep transaction is final across all honest views. Therefore, with probability $\geq 1 - \mathsf{negl}(\lambda)$, tx exists in the ledger in a final state.

Key distribution dependency. Property (2) relies on verifiers having authentic public keys. If an adversary substitutes $\mathsf{pk}'_j \neq \mathsf{pk}_j$ (violating Definition 1), it could forge signatures under pk'_j. Thus, authenticated key distribution is a necessary assumption. □

Definition 7 (Honest Committee Member Behavior)**.** *An honest committee member C_ℓ executes $\mathsf{ProofGen}(\mathsf{tx}, h, i, \mathsf{sk}_\ell)$ only if all of the following conditions hold:*

1) *Transaction tx appears at position (h, i) in C_ℓ's local view $\mathcal{L}_\ell$*
2) *Block h satisfies $h^{(\ell)}_{\mathsf{current}} - h \geq k$ (finality rule)*

An honest committee member never signs a state it has not observed or that is not yet final in its local view.

5) Proof of Finality Binding:

Complete proof of Theorem 6. We prove that no PPT adversary can produce an accepting proof for a non-final transaction. We use $\mathsf{Adv}^{\mathsf{consist}}_{(\Pi,\mathcal{L}),\mathcal{A}}$ to denote the probability that $\mathcal{A}$ causes a k-deep transaction to revert, violating Δ-consistency.

Setup. Let $\mathcal{A}$ be a PPT adversary controlling both gateways and up to f committee members. Let $\mathcal{H} = \{C_1, \ldots, C_{n-f}\}$ denote honest committee members (where $|\mathcal{H}| \geq f + 1$). Suppose $\mathcal{A}$ produces proof set $\{\pi_\ell\}_{\ell \in S}$ for transaction tx^* such that $\mathsf{ProofVer}$ accepts.

Honest member inclusion. Since $|S| \geq f + 1$ and $|\mathcal{C}_{\mathsf{corrupt}}| \leq f$, by pigeonhole principle, S contains at least one honest member $C_j \in \mathcal{H}$.

Honest behavior constraint. By Definition 7, C_j executes $\mathsf{ProofGen}(\mathsf{tx}^*, h^*, i^*, \mathsf{sk}_j)$ only if: (i) tx^* appears at position (h^*, i^*) in C_j's local view $\mathcal{L}_j$ and (ii) block h^* satisfies $h^{(j)}_{\mathsf{current}} - h^* \geq k$ (finality rule).

Finality from consistency. By the Δ-consistency property of the underlying ledger (Definition in Appendix A4): once a transaction is k-deep in any honest party's view, it remains at the same position in all future honest views, except with probability $\mathsf{Adv}^{\mathsf{consist}}_{(\Pi,\mathcal{L}),\mathcal{A}} = \mathsf{negl}(\lambda)$. Since C_j is honest and signed only when tx^* was k-deep, tx^* is final.

Adversary's options. To produce an accepting proof for non-final tx^*, $\mathcal{A}$ must either: (a) forge C_j's signature (probability $\leq \mathsf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{SIG},\mathcal{B}}(\lambda)$), (b) convince C_j to sign a non-final transaction (impossible if C_j follows Definition 7), or (c) revert a k-deep transaction (probability $\leq \mathsf{Adv}^{\mathsf{consist}}_{(\Pi,\mathcal{L}),\mathcal{A}}$).

By union bound:

$$\begin{aligned}&\Pr[\text{accept non-final } \mathsf{tx}^*]\\ &\leq (n-f) \cdot \mathsf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{SIG},\mathcal{B}}(\lambda) + \mathsf{Adv}^{\mathsf{consist}}_{(\Pi,\mathcal{L}),\mathcal{A}}\\ &= \mathsf{negl}(\lambda).\end{aligned}$$

□

6) Replay Prevention (Supporting Lemma): To prevent double-crediting under gateway corruption, the destination ledger maintains a nullifier set.

Definition 8 (Nullifier Set)**.** *A nullifier set $\mathcal{N}$ is a persistent, append-only set maintained by ledger consensus with the following interface:*

- $\mathsf{Contains}(\mathcal{N}, \mathsf{id}) \to \{0, 1\}$*: Returns 1 if $\mathsf{id} \in \mathcal{N}$, 0 otherwise.*
- $\mathsf{Add}(\mathcal{N}, \mathsf{id}) \to \mathcal{N}'$*: Returns $\mathcal{N}' = \mathcal{N} \cup \{\mathsf{id}\}$.*

The set is maintained by ledger consensus: once id is added, it cannot be removed, and all honest nodes agree on set membership.

Definition 9 (Nullifier-Protected Credit)**.** *A destination credit operation $\mathsf{Credit}(u_B, \mathsf{asset}', v, \mathsf{id}, \mathcal{L}_A)$ on ledger $\mathcal{L}_B$ with nullifier set $\mathcal{N}_B$ succeeds only if:*

1) $\mathsf{Contains}(\mathcal{N}_B, \mathsf{id}) = 0$ *(transfer ID not yet used)*
2) $\mathsf{ProofVer}$ *accepts the corresponding lock proof*
3) *All other validity conditions are satisfied*

Upon success, the ledger executes $\mathcal{N}_B \leftarrow \mathsf{Add}(\mathcal{N}_B, \mathsf{id})$.

Lemma 1 (Replay Prevention)**.** *If the destination ledger enforces nullifier checks as in Definition 9, then no adversary can cause double-crediting for the same transfer ID, even when controlling both gateways.*

Proof of Lemma 1. Suppose adversary $\mathcal{A}$ controls both gateways and attempts to credit twice with the same transfer ID id.

First credit: $\mathcal{A}$ submits $\mathsf{Credit}(\ldots, \mathsf{id}, \ldots)$ to $\mathcal{L}_B$. Since $\mathsf{id} \notin \mathcal{N}_B$ initially, the precondition is satisfied. Upon success, $\mathcal{N}_B \leftarrow \mathcal{N}_B \cup \{\mathsf{id}\}$.

Second credit attempt: $\mathcal{A}$ submits another $\mathsf{Credit}(\ldots, \mathsf{id}, \ldots)$. Now $\mathsf{Contains}(\mathcal{N}_B, \mathsf{id}) = 1$, so the precondition fails. By Definition 9, the credit is rejected.

Since $\mathcal{N}_B$ is maintained by $\mathcal{L}_B$'s consensus (not by gateways), $\mathcal{A}$ cannot bypass this check even with full gateway control. The nullifier set's append-only property (guaranteed by Δ-consistency) ensures id remains in $\mathcal{N}_B$ permanently. □

C. UC Ideal Functionality

This section provides an explicit ideal functionality that captures the intended security goal of Enhanced SATP under a static corruption model with untrusted gateways and up to f corrupted committee members per ledger. To address common modeling concerns for cross-ledger protocols, we make three choices explicit. **(i) Key material:** parties run KG to generate signature keys; upon (static) corruption, the adversary learns the corresponding secret keys, while honest keys remain hidden. Public keys are assumed to be distributed via an authenticated key distribution mechanism (Definition 1), which can be instantiated by a PKI, certificate transparency, or a consortium registry. **(ii) Ledger access:** we view each permissioned ledger as a trusted append-only data store (Section A4) that supports consistent reads of finalized transactions; committee-backed proofs serve as certified read receipts for this data store that can be checked without direct ledger access. **(iii) Communication and scheduling:** we do not assume a reliable off-chain broadcast channel (e.g., an explicit bulletin board functionality). As in UC, message delivery and scheduling are adversarially controlled (delay, reordering, and dropping). Instead, the ledgers' finalized transaction logs act as the globally readable medium: gateways and committee members can rebroadcast the publicly verifiable lock/credit evidence, and all safety-critical transitions are gated by certified reads and ledger-side enforcement. Intuitively, the simulator may control corrupted parties and off-chain scheduling, but it cannot cause a destination credit without a source escrow lock that is backed by a certified read of a finalized ledger state. Moreover, refund is time-gated (only after t_{commit}), and escrow consumption is authorized only after a certified read of a finalized destination credit; progress additionally relies on liveness/relaying assumptions (Theorem 3).

The $\mathsf{ValidLockProof}$ and $\mathsf{ValidCreditProof}$ predicates. The predicate $\mathsf{ValidLockProof}(\mathsf{proof}_A, \mathcal{L}_A, \mathcal{T}[\mathsf{id}], f)$ abstracts a certified read of a finalized lock on $\mathcal{L}_A$, while $\mathsf{ValidCreditProof}(\mathsf{proof}_B, \mathcal{L}_B, \mathcal{T}[\mathsf{id}], f)$ abstracts a certified read of a finalized destination credit on $\mathcal{L}_B$. In the real protocol, both are implemented by $\mathsf{ProofVer}$ using (i) authenticated public keys, (ii) $\geq f+1$ committee signatures, and (iii) Merkle openings to a committed transaction log. Formally, $\mathsf{ValidLockProof} = 1$ holds iff proof_A contains $\geq f+1$ distinct committee member attestations that verify under authenticated public keys, certify a finalized block of $\mathcal{L}_A$, and bind (via the Merkle opening) to a lock transaction whose contents match the stored transfer parameters in $\mathcal{T}[\mathsf{id}]$ (including id, destination ledger, recipient, and expiry). Similarly, $\mathsf{ValidCreditProof} = 1$ holds iff proof_B binds to a destination credit transaction whose contents match the stored parameters (including id and recipient u_B). Equivalently, in a "trusted data store" view, these predicates mean the corresponding transaction can be read from the ledger in a finalized state and the attestations certify that read.

Lemma 2 (SATP does not UC-realize $\mathcal{F}_{\mathsf{SATP}}^{f\text{-}\mathsf{corrupt}}$ under untrusted gateways)**.** *In the untrusted-infrastructure model where a gateway can be corrupted, the original SATP protocol Π_{SATP} does not UC-realize $\mathcal{F}_{\mathsf{SATP}}^{f\text{-}\mathsf{corrupt}}$ (Figure 4).*

Proof sketch. Let $\mathcal{Z}$ corrupt the source gateway G_A and instruct it to send a syntactically valid Phase 3 message $(\mathsf{tx}_A^*, \sigma_A^*)$ without executing Lock on $\mathcal{L}_A$ (the forged proof attack in Section IV-C1). In the real world, an honest G_B accepts σ_A^* and can proceed to credit, yielding a one-sided outcome observable to $\mathcal{Z}$. In the ideal world, the simulator would need to trigger $(\mathsf{LOCK}, \mathsf{id}, \mathsf{proof}_A)$, but $\mathsf{ValidLockProof}$ requires $\geq f+1$ valid committee proofs of an actual finalized lock, which does not exist by construction. Therefore, $\mathcal{Z}$ distinguishes the real and ideal executions. □

Theorem 7 (Enhanced SATP realizes the lock-soundness condition of $\mathcal{F}_{\mathsf{SATP}}^{f\text{-}\mathsf{corrupt}}$)**.** *Assuming authenticated key distribution for committee public keys (Definition 1), that corruption leaks the corresponding secret keys as in Figure 4, and the assumptions of Theorems 4 and 6, no PPT adversary can cause $\mathcal{F}_{\mathsf{SATP}}^{f\text{-}\mathsf{corrupt}}$ to transition a transfer from $\mathsf{pending}$ to locked without an actual finalized lock transaction on $\mathcal{L}_A$, except with probability $\mathsf{negl}(\lambda)$.*

Proof sketch. Any such transition requires $\mathsf{ValidLockProof} = 1$. By its definition, $\mathsf{ValidLockProof}$ entails $\geq f+1$ verifying committee attestations and Merkle openings for a lock transaction. By Theorem 4, forging such an accepting proof set without an honest attestation is negligible; by Theorem 6, any honest attestation must correspond to a finalized ledger state. Thus, a successful LOCK implies existence of a finalized lock transaction except with negligible probability. □

Theorem 8 (Enhanced SATP UC-realizes $\mathcal{F}_{\mathsf{SATP}}^{f\text{-}\mathsf{corrupt}}$)**.** *Assume authenticated key distribution for committee public keys (Definition 1) and static key leakage on corruption as in Figure 4. Assume the standing assumptions (A1)–(A6) and that the ledgers can be treated as trusted append-only data stores for finalized transactions (Section A4). Assume further that the ledgers enforce the authorization rules (R1)–(R3). Then the Enhanced SATP protocol $\Pi_{\mathsf{Enhanced\text{-}SATP}}$ UC-realizes $\mathcal{F}_{\mathsf{SATP}}^{f\text{-}\mathsf{corrupt}}$ (Figure 4).*

Proof sketch. Fix any PPT adversary $\mathcal{A}$ and environment $\mathcal{Z}$. We show that the ensembles of transcripts induced by the following hybrids are computationally indistinguishable, and then construct a simulator Sim for the ideal execution.

Hybrid H0 (Real). The execution $\mathsf{EXEC}_{\Pi_{\mathsf{Enhanced\text{-}SATP}},\mathcal{A},\mathcal{Z}}$ of the real protocol with adversary $\mathcal{A}$ and environment $\mathcal{Z}$.

Hybrid H1 (Ideal-signature hybrid). Same as H0 except that signatures under honest committee keys are produced via a signing oracle, while corrupted-party secret keys are still leaked as in Figure 4. If ProofVer accepts evidence that contains a signature under an honest key on a message never queried to the oracle, the execution aborts with $\perp$. By EUF-CMA security of SIG, H0 $\approx$ H1.

Hybrid H2 (Ideal-commitment hybrid). Same as H1 except that Merkle openings are treated as ideal position-binding openings: if ProofVer accepts an opening for an element that is not at the claimed position under the claimed root, the execution aborts with $\perp$. Equivalently, in H2 any accepting evidence behaves as a certified read of a finalized transaction in the trusted data store view. By position binding of VC, H1 $\approx$ H2.

Hybrid H3 (Ideal). The execution $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{SATP}}^{f\text{-corrupt}},\mathsf{Sim},\mathcal{Z}}$ where the simulator Sim runs $\mathcal{A}$ internally and interacts with $\mathcal{F}_{\mathsf{SATP}}^{f\text{-corrupt}}$ as follows: (i) it forwards initiations to the functionality; (ii) it issues $(\mathsf{LOCK}, \mathsf{id}, \mathsf{proof}_A)$ exactly when a proof set would pass ProofVer; (iii) it issues $(\mathsf{CREDIT}, \mathsf{id})$ exactly when the destination ledger-side rules (R1) would accept the corresponding credit; (iv) it issues $(\mathsf{FINALIZE}, \mathsf{id}, \mathsf{proof}_B)$ exactly when the source ledger-side rules (R2) would accept escrow consumption based on an accepting destination credit proof; and (v) it issues $(\mathsf{ABORT}, \mathsf{id})$ exactly when the refund rule (R3) becomes enabled after timeout.

To argue H2 $\approx$ H3, note that in H2 any accepting proof behaves like a certified read of a finalized transaction in the trusted data store view. By construction and Theorem 7, any successful LOCK in the ideal world corresponds to an actual finalized lock in the real world except with negligible probability. Moreover, ledger-side enforcement ensures that the only observable state transitions are those permitted by (R1)–(R3), and replay attempts are rejected due to the nullifier set. Therefore, the environment's observable transcript in H2 and H3 differs only if (a) a forged proof is accepted or (b) a non-final transaction is treated as final, which is negligible by the evidence soundness guarantees.

By transitivity, H0 $\approx$ H3, which establishes UC realization. $\square$

Ideal Functionality $\mathcal{F}_{\mathsf{SATP}}^{f\text{-corrupt}}$

Initialization:
Upon init with 1^λ, threshold f, committees $n_A, n_B \geq 3f+1$:
Init global time $\tau := 0$
Init signing keys: for each $X \in \{A, B\}$, gateway G_X and committee member $C_i^{(X)}$:
sample $(\mathsf{pk}_P, \mathsf{sk}_P) \leftarrow \mathsf{KG}(1^\lambda)$ for $P \in \{G_X, C_i^{(X)}\}$; publish pk_P
Init $\mathcal{T} := \emptyset$; used IDs $\mathcal{N}_B := \emptyset$
Corruption sets $\mathbf{C}_A, \mathbf{C}_B := \emptyset$; corrupted gateways $\mathcal{G} := \emptyset$

Clock:
Upon (TICK) from $\mathcal{Z}$: set $\tau \leftarrow \tau + 1$

Corruption (static):
Upon $(\mathsf{CORRUPT}, P)$ from the adversary $\mathcal{A}$
(i.e., Sim in the ideal world) before protocol start:
Leak sk_P to $\mathcal{A}$
if $P \in \{G_A, G_B\}$: set $\mathcal{G} \leftarrow \mathcal{G} \cup \{P\}$
if $P = C_i^{(X)}$ and $|\mathbf{C}_X| < f$: set $\mathbf{C}_X \leftarrow \mathbf{C}_X \cup \{P\}$
(Reject if $|\mathbf{C}_X| \geq f$ for threshold violation)

Transfer Initiation:
Upon $(\mathsf{INIT}, \mathsf{id}, u_A, u_B, \mathsf{asset}, v, \mathcal{L}_B, t_{\mathsf{commit}})$ from u_A:
if $\mathsf{id} \in \mathcal{N}_B$ **then** return $(\mathsf{ABORT}, \mathsf{id})$
Store $\mathcal{T}[\mathsf{id}] := (u_A, u_B, \mathsf{asset}, v, \mathcal{L}_B, t_{\mathsf{commit}}, \mathsf{pending})$; notify Sim

Lock (via certified ledger read):
Upon $(\mathsf{LOCK}, \mathsf{id}, \mathsf{proof}_A)$ from $\mathcal{A}$ (i.e., Sim in the ideal world):
if $\mathcal{T}[\mathsf{id}].\mathsf{status} = \mathsf{pending}$
and $\mathsf{ValidLockProof}(\mathsf{proof}_A, \mathcal{L}_A, \mathcal{T}[\mathsf{id}], f)$:
Update status to locked; notify u_A
(Sim cannot fake lock without a certified read of a finalized lock)

Credit (destination authorization):
Upon $(\mathsf{CREDIT}, \mathsf{id})$ from $\mathcal{A}$ (i.e., Sim in the ideal world):
if $\mathcal{T}[\mathsf{id}].\mathsf{status} = \mathsf{locked}$ **and** $\tau < \mathcal{T}[\mathsf{id}].t_{\mathsf{commit}}$ **and** $\mathsf{id} \notin \mathcal{N}_B$:
Update status to credited; set $\mathcal{N}_B \leftarrow \mathcal{N}_B \cup \{\mathsf{id}\}$; send $(\mathsf{CREDITED}, \mathsf{id})$ to u_B
(Credit only after verified lock and before timeout)

Finalize (consume source escrow):
Upon $(\mathsf{FINALIZE}, \mathsf{id}, \mathsf{proof}_B)$ from $\mathcal{A}$ (i.e., Sim in the ideal world):
if $\mathcal{T}[\mathsf{id}].\mathsf{status} = \mathsf{credited}$
and $\mathsf{ValidCreditProof}(\mathsf{proof}_B, \mathcal{L}_B, \mathcal{T}[\mathsf{id}], f)$:
Update status to completed; send $(\mathsf{FINALIZED}, \mathsf{id})$ to u_A
(Escrow consumption only after a certified read of destination credit)

Abort/Refund (after timeout):
Upon $(\mathsf{ABORT}, \mathsf{id})$ from $\mathcal{A}$ (i.e., Sim in the ideal world):
if $\mathcal{T}[\mathsf{id}].\mathsf{status} \in \{\mathsf{pending}, \mathsf{locked}\}$ **and** $\tau \geq \mathcal{T}[\mathsf{id}].t_{\mathsf{commit}}$:
Set aborted; send $(\mathsf{REFUNDED}, \mathsf{id})$ to u_A

Fig. 4. Ideal functionality $\mathcal{F}_{\mathsf{SATP}}^{f\text{-corrupt}}$ with an explicit static corruption model. The adversary learns secret keys of corrupted parties and controls their scheduling, but cannot cause destination credit without ValidLockProof, cannot consume escrow without ValidCreditProof, and cannot refund before t_{commit}.