A FORMAL ANALYSIS OF FLEX AND FLEX2

RAMSES FERNANDEZ

Abstract. This paper formalizes the cryptographic core of the FLEX protocol and its enhanced variation FLEX2 [11]. The analysis formalizes a minimal ledger abstraction, capturing Taproot, CSV timelocks, and reorg bounds, and defines ideal functionalities implemented as transaction-DAG and state machines. Main contributions include proving on-chain enforceability, CDS secrecy, soundness, leakage-bounded privacy, and universal composability realization under standard assumptions.

Keywords. BitVM, Cross-chain bridges, Garbled circuits, Conditional Disclosure of Secrets

1. Introduction

1.1. **Motivation.** The development of efficient and secure bridges for blockchain assets, particularly those involving Bitcoin, has long been a challenge in the field of cryptocurrency interoperability. Traditional approaches often require substantial capital commitments in the form of permanent security bonds, which can impose significant financial burdens on operators and challengers alike. In this context, the FLEX protocol [11] emerges as an important advancement, designed specifically to ease capital-efficient optimistic bridges on Bitcoin through the implementation of on-demand security bonds. By activating these bonds exclusively during periods of dispute, FLEX substantially mitigates the ongoing financial overhead that plagues systems reliant on perpetual deposits. This mechanism not only enhances the economic viability for participants in permissioned challenge environments but also aligns with the broader objectives of improving scalability and accessibility in BitVM-based bridging solutions, as elaborated in the foundational discussions within the FLEX manuscript.

1.2. **High-level technique.** At its core, the FLEX protocol [11] employs an approach inspired by conditional disclosure of secrets (CDS) within garbled circuits, where each involved party embeds a confidential value that is disclosed only upon the satisfaction of predefined input conditions, such as the verification failure of a succinct non-interactive argument of knowledge (SNARK). Complementing this, Taproot-based pot scripts are used in deposit addresses to enable multiple spending pathways: one contingent on the opponent's signature coupled with the revealed secret, another on the party's own signature and personal secret, and a third allowing unilateral withdrawal through signature after a designated timelock period. The on-demand nature of security bonds is achieved through the strategic application of the SIGHASH_ANYONECANPAY flag, which permits the flexible attachment of dynamic deposit inputs to transactions that have been pre-signed. Furthermore, the FLEX2 [11] iteration introduces interlocking mechanisms that mandate the challenger's bond posting prior to the operator's input provision, effectively deterring the initiation of unfounded disputes and enhancing overall protocol integrity.

Fairgate Labs.
E-mail address: ramses.fernandez@fairgate.io.

1.3. **Contributions.** We provide a comprehensive analysis of FLEX and FLEX2, isolating the exact assumptions they rely on and proving their security. In particular:

(1) We formalize a minimal ledger abstraction $\mathcal{G}_{\mathsf{Ledger}}$ that captures Taproot script-path leaves, CSV relative timelocks, and the operation SIGHASH_ANYONECANPAY, together with a common-prefix reorg bound $k_{\max}$. Within this model we give a precise account of on-demand bonds (late-bound inputs, fixed outputs).

(2) We pin down a CDS specialization of garbled circuits that releases exactly one preimage (P_a or P_b), and we enforce bitwise input binding through Lamport OTS commitments proved correct in a succinct setup proof. This identifies the minimal cryptographic assumptions: hash one-wayness/second-preimage, Schnorr EUF-CMA, GC authenticity/privacy, and SNARK knowledge soundness.

(3) We define ideal functionalities FLEX and FLEX2 and implement them by concrete protocols π_{FLEX} and π_{FLEX2} given as transaction-DAG/state machines.

(4) We prove on-chain enforceability of pot scripts, CDS one-of-two secrecy with input binding, soundness (no wrong-winner), leakage-bounded privacy, and UC realization of FLEX/FLEX2 by $\pi_{\mathsf{FLEX}}/\pi_{\mathsf{FLEX2}}$ in the $(\mathcal{G}_{\mathsf{Ledger}}, \mathcal{G}_{\mathsf{Hash}}, \mathcal{G}_{\mathsf{Sig}}, \mathcal{G}_{\mathsf{SNARK}}, \mathcal{G}_{\mathsf{GC}})$-hybrid. For FLEX2 we formalize and prove a mutual-exclusion invariant such that EarlyRefund and any winner spend are incompatible provided $\Delta_S > k_{\max}$.

(5) We relate the protocol windows $(\Delta_A, \Delta_B, \Delta_R, \Delta_S)$ to $k_{\max}$ and derive simple margin conditions that ensure safety and liveness in the presence of reorganizations and fee/policy constraints.

(6) We prove a sequential composition theorem: disjoint, k-finalized instances of either protocol (π_{FLEX} or π_{FLEX2}) preserve the single-instance guarantees, enabling safe pipelining of disputes.

2. Related Work

2.1. **BitVM/BitVM2 and emulated-covenant designs.** The foundational BitVM paradigm represents a significant leap in enabling Turing-complete smart contracts on the Bitcoin network without necessitating alterations to its underlying consensus rules, primarily through the utilization of fraud-proof mechanisms for off-chain computations [2]. Building upon this, BitVM2 refines the dispute resolution process by limiting on-chain transactions to a mere three, while introducing support for permissionless challengers under a 1-of-n honest minority assumption [12]. BitVMX leverages a virtual RISC-V CPU to execute general-purpose programs, in particular SNARK verifiers, across multiple dispute rounds. [9]. These protocols collectively emulate covenant functionalities through meticulously pre-signed transaction directed acyclic graphs (DAGs), akin to the structures employed in Section 6 of FLEX [11], harnessing Taproot scripts and SIGHASH flags to enforce predetermined transaction flows in the absence of native covenant support.

Recent advancements in this domain have further propelled the practical deployment of these technologies. For instance, the introduction of BitVM3 in September 2025 has been highlighted as a pivotal development, enhancing the capabilities of BitVM with improved efficiency and broader applicability. In parallel, Bitlayer has achieved a milestone by launching its BitVM Bridge on mainnet in July 2025, providing a trust-minimized pathway for Bitcoin liquidity in decentralized finance. Complementing these efforts, the BOB project has integrated BitVM with its Gateway infrastructure, with mainnet activation slated for the fourth quarter of 2025, following successful testnet operations. Citrea's Clementine bridge exemplifies the application of BitVM2 principles for connecting Bitcoin to rollups, maintaining a permissioned model with batched fraud proofs. Strata,

developed by Alpen Labs, preserves permissionless challenger participation alongside permanent staking requirements. Rootstock's Union bridge transitions from federated models to BitVMX-based trust minimization, ensuring security with a 1-of-n honest participant model [1, 10]. Cardinal, Cardano's bridge, adopts BitVMX with the Transfer-of-Ownership Protocol (TOOP) [7] to enable direct user fund access without operator fronting.

A critical limitation in these emulated covenant designs, as critiqued in Section 5 of FLEX [11], pertains to the constraints on incorporating dynamic inputs, which FLEX addresses through innovative use of SIGHASH_ANYONECANPAY.

2.2. **Bonded bridges and rollup dispute frameworks.** Bonded bridging protocols mandate the posting of security bonds that are subject to slashing in cases of detected misbehavior, a concept that resonates with the penalty structures analyzed in Section 3 of FLEX [11]. Within BitVM ecosystems, such bonds are typically maintained on a permanent basis to cover both liveness violations and misappropriation attempts. Complementary frameworks like BoLD optimize dispute resolution latencies in contexts such as Arbitrum rollups. The Dave protocol provides a unified abstraction for achieving balance among decentralization, security, and liveness in fraud-proof systems [13]. Optimism's dispute mechanism employs multi-round challenges with bonded assertions to ensure protocol integrity.

Subsequent refinements include the formalization of Dave in 2025, offering comparative analyses against BoLD and Optimism. Arbitrum's BoLD upgrade, deployed in February 2025 following unanimous DAO approval, eliminates permissioned validator dependencies, enhancing decentralization. These developments underscore the ongoing trade-offs between promptness, safety, and decentralization, as discussed in Section 5 of FLEX [11], where the adoption of on-demand bonds is posited as a means to augment efficiency.

2.3. **Conditional disclosure of secrets in practice.** The conditional disclosure of secrets (CDS) primitive facilitates the revelation of confidential information contingent upon the satisfaction of a specified predicate [8]. Yao's garbled circuit protocol serves as a foundational two-party secure multi-party computation (MPC) technique enabling such disclosures [14]. Practical applications have evolved to include laconic CDS variants optimizing verifier computation [6]. Linear multiparty CDS protocols have achieved optimal communication rates [3].

Recent scholarly contributions have further enriched this area. In the context of subsection 5.1 of FLEX [11], CDS is harnessed through garbled circuits to embed secrets for hash-lock activations, thereby enabling the dynamic provisioning of bonds in BitVM environments.

3. Preliminaries

3.1. **Notation.** We write $\lambda \in \mathbb{N}$ for the security parameter and use 1^λ to denote its unary encoding. A (possibly randomized) algorithm $\mathcal{A}$ is probabilistic polynomial-time (PPT) if its running time is bounded by a polynomial in λ. Unless stated otherwise, all probabilities are taken over the internal random coins of all algorithms and, when present, random oracles and setup procedures.

For a finite set S, we write $x \xleftarrow{\$} S$ for sampling x uniformly from S. We let $\{0,1\}$ denote the set of bits, $\{0,1\}^\ell$ the set of bitstrings of length ℓ, and $\{0,1\}^*$ finite bitstrings of arbitrary length. For $n \in \mathbb{N}$, $[n] := \{1, 2, \ldots, n\}$. A function $\mu : \mathbb{N} \to \mathbb{R}_{\geq 0}$ is negligible in λ (denoted $\mu(\lambda) = \mathsf{negl}(\lambda)$) if for all constants $c > 0$ there exists N_c such that for all $\lambda \geq N_c$ we have $\mu(\lambda) < \lambda^{-c}$. An event holds with overwhelming probability if its failure probability is negligible. Two distribution ensembles $\{X_\lambda\}_\lambda$ and $\{Y_\lambda\}_\lambda$ are

computationally indistinguishable, written $X_\lambda \approx_c Y_\lambda$, if for every PPT distinguisher $\mathcal{D}$, $|\Pr[\mathcal{D}(X_\lambda)=1] - \Pr[\mathcal{D}(Y_\lambda)=1]| = \mathsf{negl}(\lambda)$.

We write $\mathsf{Adv}^{\mathsf{auth}}_{\mathsf{CDS}} := \max_{\mathcal{A}} \mathsf{Adv}^{\mathsf{auth}}_{\mathsf{GC}}(\mathcal{A})$ for the garbled circuit authenticity advantage, it measures the advantage of a PPT adversary $\mathcal{A}$ (acting as garbler) in forging a valid output label pair that decodes to an incorrect circuit evaluation, violating output integrity. We assume $\mathsf{Adv}^{\mathsf{auth}}_{\mathsf{CDS}} = \mathsf{negl}(\lambda)$

3.2. **Cryptographic primitives and assumptions.** We formalize the primitives required by the protocol and state the exact security notions we rely on in proofs.

3.2.1. *Hash functions.* Let $\mathcal{H} = \{H_\lambda : \{0,1\}^* \to \{0,1\}^{\Theta(\lambda)}\}$ be an efficiently computable hash family. For a PPT adversary $\mathcal{A}$, we use the following advantage notation for preimage, second preimage and collision resistance, respectively:

$$\mathsf{Adv}^{\mathsf{pre}}_H(\mathcal{A}) := \Pr\Big[x \leftarrow \{0,1\}^*;\ y := H_\lambda(x);\ x' \leftarrow \mathcal{A}(1^\lambda, y) :\ H_\lambda(x') = y\Big]$$

$$\mathsf{Adv}^{\mathsf{2pre}}_H(\mathcal{A}) := \Pr\Big[x \leftarrow \{0,1\}^*;\ x' \leftarrow \mathcal{A}(1^\lambda, x) :\ x' \neq x \ \wedge\ H_\lambda(x') = H_\lambda(x)\Big]$$

$$\mathsf{Adv}^{\mathsf{coll}}_H(\mathcal{A}) := \Pr\Big[(x, x') \leftarrow \mathcal{A}(1^\lambda) :\ x \neq x' \ \wedge\ H_\lambda(x) = H_\lambda(x')\Big]$$

3.2.2. *Digital signatures.* A digital signature scheme $\Sigma = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vf})$ is correct if $\mathsf{Vf}(\mathsf{pk}, m, \mathsf{Sign}(\mathsf{sk}, m)) = 1$ for $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and all m. The EUF-CMA experiment EUF-CMA$_\Sigma(\mathcal{A})$ gives pk to $\mathcal{A}$, answers adaptive signing queries, and $\mathcal{A}$ wins with a valid $(m^\star, \sigma^\star)$ for a new message. We write $\mathsf{Adv}^{\mathsf{euf\text{-}cma}}_\Sigma(\mathcal{A}) := \Pr[\mathcal{A} \text{ wins EUF-CMA}_\Sigma]$, and require it to be negligible. FLEX uses two digital signature schemes, namely Schnorr and Lamport, which are briefly introduced below:

(1) Schnorr: Let $(\mathbb{G}, q, g)$ be a prime-order group. Key generation: $x \leftarrow \mathbb{Z}_q$, $X = g^x$. Signature $\mathsf{Sign}(x, m)$: pick $k \leftarrow \mathbb{Z}_q$, set $R = g^k$, $c = H(m \| R)$, $s = k + c\,x \bmod q$, output $\sigma = (R, s)$. Verification $\mathsf{Verify}(X, m, (R, s))$: recompute $c = H(m\|R)$ and accept if, and only if, $g^s = R \cdot X^c$. In the ROM, Schnorr is EUF-CMA secure under discrete-log hardness.

(2) Lamport: Fix output length n and message length ℓ. Key generation: sample $s_{i,b} \leftarrow \{0,1\}^n$ for $i \in [\ell], b \in \{0,1\}$ and set $y_{i,b} = H(s_{i,b})$; $\mathsf{pk} = \{y_{i,b}\}$. Signature $m \in \{0,1\}^\ell$, output $\sigma = \{s_{i,m_i}\}_{i \in [\ell]}$. Verification by checking $H(s_{i,m_i}) = y_{i,m_i}$ for all i. Under preimage resistance of H, Lamport is one-time EUF-CMA secure.

3.2.3. *SNARKs.* Let us write vp for the public constants required to check a proof. It consists of a verification key and a commitment to the statement and circuit description. An (argument) system for an NP relation $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ is given by a triple of the form $\mathfrak{P} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ such that:

(1) $\mathsf{Setup}(1^\lambda, \mathcal{R}) \to vp$: returns the verifier parameters.

(2) $\mathsf{Prove}(vp, x, w) \to \Pi$: prover outputs proof Π for $(x, w) \in \mathcal{R}$.

(3) $\mathsf{Verify}(vp, x, \Pi) \in \{0,1\}$: verifier accepts valid proofs.

We assume the following properties:

(1) Succinctness: The length of the proof, $|\Pi|$, and the verifier time are $poly(\lambda, \log|x|)$.

(2) Completeness: For all $(x, w) \in \mathcal{R}$,

$$\Pr\Big[vp \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R});\ \Pi \leftarrow \mathsf{Prove}(vp, x, w) : \mathsf{Verify}(vp, x, \Pi) = 1\Big] = 1 - \mathsf{negl}(\lambda).$$

(3) Knowledge soundness: There exists a PPT extractor Ext such that for all PPT $\mathcal{A}$,

$$\mathsf{Adv}^{\mathsf{know\text{-}snd}}_{\mathfrak{P}}(\mathcal{A}) := \Pr\left[\begin{array}{l} vp \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R}); \\ (x, \Pi) \leftarrow \mathcal{A}^{\mathsf{Prove}(vp,\cdot,\cdot)}(vp); \\ \mathsf{Verify}(vp, x, \Pi) = 1 \wedge (x, w) \notin \mathcal{R} \text{ for all } w \end{array}\right] = \mathsf{negl}(\lambda),$$

and moreover $\mathsf{Ext}^{\mathcal{A}}(vp)$ outputs a witness w such that $(x, w) \in \mathcal{R}$ whenever $\mathcal{A}$ produces (x, Π) accepted by Verify.

(4) Zero-knowledge: There exists a PPT simulator Sim such that for all PPT distinguishers $\mathcal{D}$ and $(x, w) \in \mathcal{R}$,

$$\left|\Pr\left[\begin{array}{l} vp \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R}); \\ \Pi \leftarrow \mathsf{Prove}(vp, x, w) : \\ \mathcal{D}(vp, x, \Pi) = 1 \end{array}\right] - \Pr\left[\begin{array}{l} vp \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R}); \\ \Pi \leftarrow \mathsf{Sim}(vp, x) : \\ \mathcal{D}(vp, x, \Pi) = 1 \end{array}\right]\right| = \mathsf{negl}(\lambda).$$

3.2.4. *Garbled circuits and CDS specialization.* A (public-key-free) garbling scheme for Boolean circuits is a triple $\mathsf{GC} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev})$ operating over a circuit $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^\ell$ with two parties: garbler G and evaluator E.

(1) $\mathsf{Gb}(1^\lambda, C) \to (\widetilde{C}, \{K^0_i, K^1_i\}_{i\in[n+m]})$ outputs a garbled circuit $\widetilde{C}$ and wire labels (keys) for each input wire i.

(2) $\mathsf{En}(\{K^b_i\}, x, y) \to (L_x, L_y)$ maps inputs to corresponding label sets $L_x = \{K^{x_i}_i\}_{i\le n}$ and $L_y = \{K^{y_j}_{n+j}\}_{j\le m}$.

(3) $\mathsf{Ev}(\widetilde{C}, L_x, L_y) \to \widetilde{z}$ produces a garbled output $\widetilde{z}$. A public decoding function De maps $\widetilde{z}$ to $z = C(x, y)$ (either via a decoding table emitted by Gb or by having output labels be global offsets), with $\mathsf{De}(\widetilde{z}) = \perp$ when no consistent decoding exists.

We require the following properties:

(1) Correctness: For all (x, y) and C, if $(\widetilde{C}, K) \leftarrow \mathsf{Gb}(1^\lambda, C)$ and $(L_x, L_y) \leftarrow \mathsf{En}(K, x, y)$, then $\mathsf{De}(\mathsf{Ev}(\widetilde{C}, L_x, L_y)) = C(x, y)$ with probability $1 - \mathsf{negl}(\lambda)$.

(2) Privacy: There exists a PPT simulator $\mathsf{Sim}_{\mathsf{GC}}$ such that for all PPT distinguishers $\mathcal{D}$ and all (x, y),

$$\left|\Pr\left[\begin{array}{l} (\widetilde{C}, K) \leftarrow \mathsf{Gb}(1^\lambda, C); \\ (L_x, L_y) \leftarrow \mathsf{En}(K, x, y); \\ \mathcal{D}(\widetilde{C}, L_x, L_y) = 1 \end{array}\right] - \Pr\left[\begin{array}{l} (\widetilde{C}', L'_x, L'_y) \leftarrow \mathsf{Sim}_{\mathsf{GC}}(1^\lambda, C, |x|, |y|, C(x, y)); \\ \mathcal{D}(\widetilde{C}', L'_x, L'_y) = 1 \end{array}\right]\right|$$

is negligible.

(3) Authenticity: No PPT evaluator can produce labels that simultaneously make two different output decodings valid, nor can it forge labels for input wires that pass gate checks without having obtained the corresponding labels from G. Formally, for any PPT $\mathcal{A}$ interacting with G on $(\widetilde{C}, K)$ and receiving (L_x, L_y), the probability that $\mathcal{A}$ outputs (L'_x, L'_y) with $\mathsf{De}(\mathsf{Ev}(\widetilde{C}, L'_x, L'_y)) \neq \perp$ for an input tuple inconsistent with (x, y) is $\mathsf{negl}(\lambda)$.

4. System model and threat assumptions

4.1. **Functionalities $\mathcal{G}_{\mathsf{Hash}}$, $\mathcal{G}_{\mathsf{Sig}}$, $\mathcal{G}_{\mathsf{SNARK}}$, and $\mathcal{G}_{\mathsf{GC}}$.** We define ideal functionalities exposing the behavior required in proofs; cryptographic hardness is implicit in the interfaces.

Algorithm 1 $\mathcal{G}_{\mathsf{Hash}}$

function HASH(x) **return** $\mathsf{H}(x)$
end function
function CHECKPREIMAGE(y, x) **return** $[\mathsf{H}(x) = y]$
end function

Algorithm 2 $\mathcal{G}_{\mathsf{Sig}}$

function KeyGen(scheme) **return** $\mathsf{KeyGen}_{\mathsf{scheme}}$
end function
function Sign(sk, m) **return** $\mathsf{Sign}_{\mathsf{scheme}}(\mathsf{sk}, m)$
end function
function Vf(pk, m, σ) **return** $\mathsf{Verify}_{\mathsf{scheme}}(\mathsf{pk}, m, \sigma)$
end function
function SERIALIZE(tx, i, sighash):
    base = BIP341/342 Taproot sighash preimage
    if sighash has ANYONECANPAY **then**: include only input i's outpoint, sequence, annex
    else include the aggregated hashes over all inputs
    end if
    handle outputs per SIGHASH_ALL/NONE/SINGLE
    return TaggedHash("TapSighash", base || sighash)
end function
procedure SIGN_ACP(tx, sk, i)
    $m \leftarrow$ SERIALIZE($\mathsf{tx.outputs}, \mathsf{tx.inputs}[i], \mathsf{SIGHASH_ANYONECANPAY}$);
    return Sign(sk, m)
end procedure

Algorithm 3 $\mathcal{G}_{\mathsf{SNARK}}$

function SETUP($\mathcal{R}$) **return** vp
end function
function PROVE(vp, x, w) **return** Π
end function
function VERIFY(vp, x, Π) **return** $0/1$
end function

Algorithm 4 $\mathcal{G}_{\mathsf{GC}}$

function $\mathsf{Gb}(C)$ **return** $(\widetilde{C}, \mathsf{labels})$
end function
function EncInputs(labels, x) **return** L
end function
function BindOK(Y, L)
    return 1 if, and only if, every provided label in L verifies against its $(y_{i,0}, y_{i,1}) \in Y$
end function
function $\mathsf{Ev}(\widetilde{C}, L, Y)$
    if not BindOK(Y, L) **then return** $\perp$
    end if
    $o \leftarrow \mathsf{Evaluate}(\widetilde{C}, L)$
    if o contains predicate bit b and secrets (P_a, P_b) **then**
        if $b = 0$ **then return** P_a
        elsereturn P_b
        end if
    else return o
    end if
end function

4.2. **Participants and roles.** The main participants and their associated role are Alice (the prover) who asserts a predicate and, in disputes, aims to win by producing the appropriate opening; and Bob (the challenger), who disputes invalid assertions and can win to seize deposits or block reimbursement. We also consider the following auxiliary actors: a public register $\mathcal{C}_{\mathsf{perm}}$ of permissioned challengers authorized to challenge; and the watchtowers, $\mathcal{W}$, who may broadcast pre-signed closure transactions per protocol logic.

FLEX operates in a *permissioned challenge setting, where a committee of operators (such as liquidity providers or watchtowers) actively participates during peg-outs* (Section 6). Unlike BitVM2, which allows *anyone to act as a challenger* with a *1-of-n honest minority assumption*, FLEX focuses on scenarios where the set of challengers is predetermined and known. This design choice reflects the practical deployment model of bridges like Rootstock's Union, where *liquidity provider and watchtower roles are open to a wide range of participants* but still require registration with the protocol.

The permissioned model enables optimizations such as direct transfers when all operators agree (subsection 3.1), but requires maintaining a public roster of authorized challengers. This roster must be queryable to determine who may legitimately issue challenges, penalty transactions, and stall-closure transactions in the FLEX2 variant.

Algorithm 5 $\mathcal{G}_{\mathsf{Perm}}$

State: fixed public set $\mathcal{C}_{\mathsf{perm}} = \{vk_1, \ldots, vk_n\}$
function Query **return** $\mathcal{C}_{\mathsf{perm}}$
end function
function IsPermitted(vk) **return** $[vk \in \mathcal{C}_{\mathsf{perm}}]$
end function

4.3. **Communication and timing.** The FLEX protocol [11] assumes asynchronous off-chain communication between operators for setup and coordination (subsection 6.2 of [11]). During the setup phase, *Alice and Bob exchange Lamport public keys* and *each party proves, in zero-knowledge, the correctness of the circuit and the correspondence between*

input labels and Lamport public keys. This exchange requires reliable message delivery but does not assume synchronous timing.

The protocol's security relies on on-chain timing through Bitcoin's block confirmation mechanism and relative timelocks (CSV), not on off-chain message delivery bounds. As stated in subsection 6.4 of [11], FLEX defines three timelock periods $(\Delta_A, \Delta_B, \Delta_R)$ measured in blocks, within which participants must broadcast specific transactions. The network model therefore needs only eventual delivery for off-chain messages, while timing guarantees come from the blockchain layer.

Algorithm 6 $\mathcal{G}_{\mathsf{Net}}$

State: multiset mailbox $\mathcal{M}$
procedure SEND$(u \to v, m)$: $\mathcal{M} \leftarrow \mathcal{M} \cup \{(u, v, m)\}$
end procedure
function DELIVER **return** any $(u, v, m) \in \mathcal{M}$ chosen by $\mathcal{A}$ (and remove it)
end function

A transaction is confirmed when included in a block, and k-final after k more blocks without reorg. Relative timelocks (CSV) count blocks since the locked UTXO's confirmation.

4.4. **Ledger model** $\mathcal{G}_{\mathsf{Ledger}}$**.** FLEX leverages Bitcoin without consensus changes, using Taproot, CSV, hashlocks, and the SIGHASH_ANYONECANPAY flag (with SIGHASH_ALL) to fix outputs while allowing inputs to be added on demand. Subsection 6.5 describes the core construct such as *Taproot addresses,* pot_A *or* pot_B *associated with a Taproot tree containing three leaves* representing:

(1) Counterparty winning with preimage P_b (or P_a),

(2) Owner winning with own preimage, and

(3) Owner refunding after CSV using CSV.

Section 5 identifies a key limitation: with emulated covenants, one cannot pre-build a transaction DAG where a child spends a deposit output whose txid is unknown at construction time. FLEX overcomes this by fixing outputs and late-binding inputs via SIGHASH_ANYONECANPAY, enabling on-demand bonds.

Let $\mathsf{ht}[x]$ denote the confirmation height of object x (a transaction or UTXO), that is: the height of the block that first confirms it.

Algorithm 7 $\mathcal{G}_{\mathsf{Ledger}}$

State: chain height h; confirmation map $\mathsf{ht}[\cdot]$; reorg bound $k_{\max}$
function CONFIRM(tx)
    $\mathsf{ht}[\mathsf{tx}] \leftarrow h$
    for all $o \in \mathsf{tx.outputs}$ **do** $\mathsf{ht}[o] \leftarrow h$
    end for
end function
function FINALIZE_K(tx) **return** $[\,\mathsf{tx}$ confirmed $\wedge\ h - \mathsf{ht}[\mathsf{tx}] \geq k_{\max}\,]$
end function
function POLICYADMISSIBLE(tx) **return** True/False
end function
function CONFLICTS(tx, tx′) **return** $[\,\mathsf{tx}$ double-spends an input of $\mathsf{tx}'\,]$
end function
function CSV_OK(o, τ) **return** $[\,h - \mathsf{ht}[o] \geq \tau\,]$ ▷ h is the current chain height
end function
function TAPROOTSPENDOK$(\ell, w, \mathsf{tx}, o)$ ▷ o is the UTXO spent by this input
    if $\ell = \mathsf{HashLock}(\mathsf{pk}, V)$ **then** parse $w = (p, \sigma)$
        return $[\,\mathcal{G}_{\mathsf{Hash}}.\mathsf{Hash}(p) = V \wedge \mathcal{G}_{\mathsf{Sig}}.\mathsf{Vf}(\mathsf{pk}, \mathsf{tx}, \sigma) = True\,]$
    end if
    if $\ell = \mathsf{CSVRefund}(\mathsf{pk}, \tau)$ **then** parse $w = \sigma$
        return $[\,\mathrm{CSV_OK}(o, \tau) \wedge \mathcal{G}_{\mathsf{Sig}}.\mathsf{Vf}(\mathsf{pk}, \mathsf{tx}, \sigma) = True\,]$
    end if
end function
function POTCLAIMFINALIZED **return** [FINALIZE_K(AliceTakesDeposit) $= True \vee$ FINALIZE_K(BobTakesDeposit) $= True$]
end function

4.5. **Adversary capabilities.** The security of FLEX bridges depends on honest behavior from operators and the underlying blockchain security assumptions. Section 3 distinguishes between two types of failures: misappropriation attempts, where *a party deviates from honest behavior in an attempt to unlawfully extract funds from the bridge*, and liveness violations, where *a party does not respond when required.*

The protocol assumes that misappropriation attempts will be actively challenged by remaining operators, with *each operator disputing separately, as BitVM does not support shared disputes* (Section 3). The adversary model must account for malicious operators who may attempt to create invalid state transitions, forge signatures, or manipulate the dispute process.

FLEX inherits Bitcoin's security model, including the assumption of honest majority mining power and bounded blockchain reorganizations. As discussed in subsection 4.1 regarding BitVM2, the protocol operates under assumptions about chain quality and common-prefix properties. The adversary can schedule message delivery, craft conflicting transactions, and choose transaction fees, but cannot violate fundamental blockchain properties except with negligible probability. We assume a longest-chain ledger with the common-prefix property: there exists a parameter $k_{\max}$ such that $\Pr[\text{reorg depth} > k_{\max}] = \mathsf{negl}(\lambda)$, that is any transaction with at least $k_{\max}$ confirmations is final.

Algorithm 8 Adversary $\mathcal{A}$

State: corrupted set $\mathcal{P}_{\mathsf{corrupt}}$
procedure CORRUPT(P): add P and expose P's state/keys
end procedure
procedure SCHEDULE
    Call $\mathcal{G}_{\mathsf{Net}}.\mathsf{Deliver}$; choose blocks subject to common-prefix with bound $k_{\max}$
end procedure

4.6. **Proof objects and verifier parameters.** FLEX uses succinct proofs as inputs to the dispute circuits: the side-system burn proof is provided as a SNARK (or any sound succinct proof), and setup also includes a proof that binds input labels to Lamport commitments. These proofs are verified inside the garbled circuits: if the burn proof fails verification, Bob's circuit releases P_b, and symmetrically Alice learns P_a when the proof verifies. We do not assume a global CRS. Instead, during the two-party off-chain setup the parties fix whatever verifier parameters their chosen proof system requires, and hard-code those parameters into the circuits. This matches FLEX §6, which allows interactive proofs for setup and leaves the proof system choice to the instantiation.

5. Informal protocol overview

We overview the FLEX dispute protocol and its FLEX2 enhancements from the perspective of transactions and relative-timelock windows. The construction realizes a capital-light, on-demand bonded dispute between a prover (Alice) and a challenger (Bob). Its cryptographic core couples:

(1) CDS-style garbled circuits that release exactly one secret P_a or P_b,

(2) Taproot pot outputs whose script leaves are gated by hashlocks and relative timelocks,

(3) SIGHASH_ANYONECANPAY signatures so deposits can be late-bound at dispute time while outputs remain fixed.

We specify the base transaction DAG (FLEX), then the FLEX2 interlocks that: (1) enforce ordering, (2) allow third-party stall-closure, and (3) enable a two-step early refund.

5.1. **FLEX transaction DAG.** Let $V_a = \mathsf{H}(P_a)$ and $V_b = \mathsf{H}(P_b)$ denote hash commitments, while $v_A, v_B \in \mathbb{Z}_{\geq 0}$ denote the corresponding pot output amounts. The Taproot pot for Alice's deposit, pot_A, exposes three script-path leaves:

$$\ell^A_{\mathsf{BobWins}} : \mathsf{Verify}(\mathsf{pk}_B, \cdot) \ \wedge \ \big(\mathsf{H}(p) = V_b\big),$$
$$\ell^A_{\mathsf{AliceWins}} : \mathsf{Verify}(\mathsf{pk}_A, \cdot) \ \wedge \ \big(\mathsf{H}(p) = V_a\big),$$
$$\ell^A_{\mathsf{Refund}} : \mathsf{Verify}(\mathsf{pk}_A, \cdot) \ \wedge \ \mathsf{CSV}(\tau_A),$$

and symmetrically for Bob's deposit pot_B with leaves that accept P_a or P_b accordingly.

If Bob wins the dispute he learns P_b and can spend pot_A through $\ell^A_{\mathsf{BobWins}}$; if Alice wins, she learns P_a and can spend pot_B through $\ell^B_{\mathsf{AliceWins}}$. Otherwise, owners regain funds through the refund leaves after their CSV delays elapse.

By using SIGHASH_ANYONECANPAY, each deposit transaction includes a pre-signed covenant input, fixing the pot outputs while allowing additional inputs to be attached later. The funding inputs that supply the deposit are added and signed normally at dispute time. This removes any need to pre-commit deposit TXIDs in the covenant DAG.

The base FLEX DAG consists of pre-signed transactions and three relative-timelock windows:

(1) StartDispute produces the ReimburseEnabler output that time-gates the process.

(2) Window W_A opens: Alice must broadcast AliceDeposit (to pot_A) and her input commitment AliceInput (Lamport-bound labels) within W_A.

(3) Then W_B opens: Bob must broadcast BobDeposit (to pot_B) and BobInput (his labels) within W_B.

(4) Finally W_R opens: if neither party has won, *Reimburse* can be executed after W_R (subject to CSV).

Failure to meet a window enables penalties: MissingAliceDeposit, MissingBobDeposit, and ultimately AliceLost if Bob has won, which blocks reimbursement.

Off-chain, the parties evaluate two complementary CDS circuits tied to the dispute predicate f: one reveals P_a if, and only if, Alice's assertion holds (or a counter-condition fails), the other reveals P_b if, and only if, it does not. Input labels are bound to Lamport OTS commitments through a setup proof to prevent opening both labels for any bit.

Let the node set be

$$\begin{aligned}\mathcal{T} =\{&\mathsf{StartDispute}, \mathsf{ReimburseEnabler}, \mathsf{AliceDeposit}, \mathsf{BobDeposit},\\ &\mathsf{AliceInput}, \mathsf{BobInput}, \mathsf{MissingAliceDeposit}, \mathsf{MissingBobDeposit},\\ &\mathsf{AliceLost}, \mathsf{Reimburse}\}.\end{aligned}$$

We write $u \rightsquigarrow v$ if v is only valid/relay-admissible after u confirms or a window after u opens. The base partial order includes:

$$\begin{aligned}\mathsf{StartDispute} &\rightsquigarrow \mathsf{ReimburseEnabler} \rightsquigarrow \mathsf{W}_A\\ &\rightsquigarrow \{\mathsf{AliceDeposit}, \mathsf{AliceInput}\}\\ &\rightsquigarrow \mathsf{W}_B \rightsquigarrow \{\mathsf{BobDeposit}, \mathsf{BobInput}\} \rightsquigarrow \mathsf{W}_R \rightsquigarrow \mathsf{Reimburse},\end{aligned}$$

and penalty edges W_A expired $\rightsquigarrow$ MissingAliceDeposit, W_B expired $\rightsquigarrow$ MissingBobDeposit and winner-take-all closure Bob learns $P_b \rightsquigarrow$ AliceLost.

5.2. **FLEX2 enhancements.** FLEX2 introduces three input UTXOs and additional pre-signed transactions that enforce ordering, permit third-party stall-closure, and enable accelerated (two-window) recovery for an honest Alice when Bob is unresponsive.

The interlocks encode the intended order

$$\mathsf{AliceDeposit} \;\to\; \mathsf{BobDeposit} \;\to\; \mathsf{AliceInput} \;\to\; \mathsf{BobInput}$$

by making later steps spend an enabler output created by the previous step. Any deviation becomes non-viable on-chain because it lacks the required enabler.

Two transactions, DisputeIsStalled1 and DisputeIsStalled2, can be broadcast by any watchtower after specific delays to force progress and block premature early-refund by consuming ReimburseEnabler. They consume public enabler outputs and re-anchor the process so collateral/time-at-risk are bounded even if one party disappears.

FLEX2 provides a guarded early-refund path:

$$\mathsf{TryEarlyRefund} \;\to\; (\mathsf{StillOpen} \;\vee\; \mathsf{EarlyRefund}).$$

Alice can attempt an accelerated refund via TryEarlyRefund. If the dispute is still live and Bob is online, then Bob can immediately publish the conflicting spend StillOpen (Figure

3 of [11]), thereby preventing EarlyRefund. Otherwise, if Bob remains inactive, Alice can complete the accelerated refund via EarlyRefund.

(1) TryEarlyRefund can be broadcast by Alice after Bob's deadline in W_B, but only after a penalization transaction (such as MissingBobDeposit or NoBobChallenge) has confirmed.

(2) StillOpen is a counter-transaction broadcast only by Bob: upon seeing TryEarlyRefund while the dispute is still open, Bob broadcasts StillOpen to invalidate the early-refund attempt.

(3) EarlyRefund becomes spendable only after the CSV delay Δ_S counted from the confirmation height of TryEarlyRefund, and it succeeds only if StillOpen has not been confirmed (that is: no conflict on-chain).

6. Formal specification

We specify the protocol in a simulation-friendly manner. We first present state machines for Alice, Bob, a generic watchtower, and the ledger abstraction; then ideal functionalities ($\mathcal{G}_{\mathsf{Hash}}, \mathcal{G}_{\mathsf{Sig}}, \mathcal{G}_{\mathsf{SNARK}}, \mathcal{G}_{\mathsf{GC}}$), the core functionalities FLEX/FLEX2, followed by the real-world protocols π_{FLEX} and π_{FLEX2}. We close with a precise leakage function Leak and a table of safety invariants and time windows.

6.1. State machines.

We model each participant as a reactive state machine driven by events (off-chain messages, on-chain confirmations, and timers).

6.1.1. *Alice.* Alice represents the operator role in FLEX, specifically the party who fronts funds to end-users during peg-outs and later seeks reimbursement from the protocol. Subsection 6.1 describes the scenario: *Alice is assumed to be an operator of a bridge that fronts funds to an end-user, Carol. Alice will be reimbursed later by the protocol.*

The protocol begins when Carol requests a peg-out by burning bitcoins on the side-system. *If Alice and Bob have agreed to pay Carol, they would transfer the funds directly to Carol's address. However, if Bob refuses to cooperate with Alice in the direct payment to Carol, Alice fronts the funds to Carol, and the reimbursement protocol is triggered* (Section 6.3).

Alice's execution follows a windowed schedule across three phases. In the first window W_A, Alice must *broadcast AliceDeposit (to* pot_A*) and her input commitment AliceInput within* W_A (subsection 6.4). AliceDeposit includes one pre-signed enabler input using SIGHASH_ANYONECANPAY, which fixes the pot outputs while allowing additional funding inputs to be attached later. Those funding inputs are added at dispute time and signed normally. This removes any need to pre-commit the deposit TXID in the DAG.

During the second window W_B, Alice monitors Bob's actions. If both parties complete their deposits and inputs, the protocol enters the circuit evaluation phase (subsection 6.6). Alice evaluates her garbled circuit using labels obtained from Bob's Lamport-bound input (openings under Lamport commitments). *If the SNARK is invalid, Bob will obtain the secret* P_b allowing him to claim Alice's deposit and block reimbursement. Conversely, if Alice's assertion holds, the evaluation reveals P_a (subsection 6.6). The Taproot tree for pot_B has an Alice wins leaf that validates (pk_A, P_a), so Alice can spend pot_B, as described in subsection 6.5.

In FLEX2, if Bob fails to respond, Alice can pursue an accelerated reimbursement path. Subsection 6.9 describes this: *if Bob's node is down and cannot interact on-chain, the protocol waits for the full dispute period* in baseline FLEX, but FLEX2 offers *a resolution within two timelock periods when Bob is not operational but is not malicious* through the TryEarlyRefund mechanism.

Remark 1. *Our base FLEX formalization uses the two-transaction variant that separates deposit and input publication (AliceDeposit/AliceInput and BobDeposit/BobInput)* [11]. *FLEX2 additionally enforces ordering via interlocks (subsection 7.6) and adds the guarded early-refund path; accordingly, we give a separate Alice machine for FLEX2 (Alg. 10).*

Algorithm 9 Alice State Machine (FLEX)

State: $s \in \{\textsc{Init}, \mathsf{W}_A, \mathsf{W}_B, \textsc{Eval}, \textsc{Refund}, \textsc{Done}\}$; heights h_0, h_A
procedure $\textsc{Start}(h)$ $s \leftarrow \mathsf{W}_A$; $h_0 \leftarrow h$
end procedure
procedure $\textsc{Tick}(h)$
  if $s = \mathsf{W}_A$ **then**
    broadcast AliceDeposit, AliceInput; if both k-final then $h_A \leftarrow h$; $s \leftarrow \mathsf{W}_B$
    if $h \geq h_0 + \Delta_A$ and not both k-final **then** $s \leftarrow \textsc{Refund}$
    end if
  else if $s = \mathsf{W}_B$ **then**
    if $\textsc{Finalize_k}$(BobDeposit) $\wedge$ $\textsc{Finalize_k}$(BobInput) **then** $s \leftarrow \textsc{Eval}$
    end if
    if $h \geq h_A + \Delta_B$ **and** $\neg(\textsc{Finalize_k}(\mathsf{BobDeposit}) \wedge \textsc{Finalize_k}(\mathsf{BobInput}))$ **then** $s \leftarrow \textsc{Refund}$
    end if
  else if $s = \textsc{Eval}$ **then**
    run $\mathcal{G}_{\mathsf{GC}}$.Ev on inputs; if output $= P_a$ broadcast AliceTakesDeposit; $s \leftarrow \textsc{Done}$; else $s \leftarrow \textsc{Refund}$
  else if $s = \textsc{Refund}$ **then**
    if $h \geq h_0 + \Delta_A + \Delta_B + \Delta_R$ and $\neg$PotClaimFinalized() **then**
      broadcast Reimburse; $s \leftarrow \textsc{Done}$
    end if
  end if
end procedure

Algorithm 10 Alice State Machine (FLEX2)

State: $s \in \{\textsc{Init}, \mathsf{W}_A, \mathsf{W}_B, \textsc{Eval}, \textsc{Refund}, \textsc{Done}\}$; heights h_0, h_A
procedure $\textsc{Start}(h)$ $s \leftarrow \mathsf{W}_A$; $h_0 \leftarrow h$
end procedure
procedure $\textsc{Tick}(h)$
  if $s = \mathsf{W}_A$ **then**
    broadcast AliceDeposit, AliceInput; if both k-final then $h_A \leftarrow h$; $s \leftarrow \mathsf{W}_B$
    if $h \geq h_0 + \Delta_A$ and not both k-final **then** $s \leftarrow \textsc{Refund}$
    end if
  else if $s = \mathsf{W}_B$ **then**
    if $\textsc{Finalize_k}$(BobDeposit) $\wedge$ $\textsc{Finalize_k}$(BobInput) **then** $s \leftarrow \textsc{Eval}$
    end if
    if $h \geq h_A + \Delta_B$ **and** $\neg(\textsc{Finalize_k}(\mathsf{BobDeposit}) \wedge \textsc{Finalize_k}(\mathsf{BobInput}))$ **then** $s \leftarrow \textsc{Refund}$
    end if
  else if $s = \textsc{Eval}$ **then**
    run $\mathcal{G}_{\mathsf{GC}}$.Ev on inputs; if output $= P_a$ broadcast AliceTakesDeposit; $s \leftarrow \textsc{Done}$; else $s \leftarrow \textsc{Refund}$
  else if $s = \textsc{Refund}$ **then**
    if $\textsc{Finalize_k}$(MissingBobDeposit $\vee$ NoBobChallenge) **then**
      broadcast TryEarlyRefund if not sent
      if $\textsc{Finalize_k}$(TryEarlyRefund) and $h \geq \mathsf{ht}[\mathsf{TryEarlyRefund}] + \Delta_S$ and $\neg$PotClaimFinalized() **then**
        broadcast EarlyRefund; if k-final then $s \leftarrow \textsc{Done}$
      end if
    else if $h \geq h_0 + \Delta_A + \Delta_B + \Delta_R$ and $\neg$PotClaimFinalized() **then**
      broadcast Reimburse; $s \leftarrow \textsc{Done}$
    end if
  end if
end procedure

6.1.2. *Bob.* Bob serves as the challenger in FLEX disputes, with responsibility to *defend the protocol against any potential malicious actions by Alice* (subsection 6.1). In the permissioned challenge setting, Bob is one of the pre-authorized operators or watchtowers who can dispute invalid assertions.

Bob's involvement begins after Alice has initiated the dispute. Subsection 6.4 specifies: *Alice tried to take the funds [...] Bob is required to perform the same action* after Alice posts her deposit and input. The *BobDeposit transaction must include Alice's SNARK included alongside Alice's Lamport-bound input (with a setup proof that the labels match the commitments). In addition, Bob verifies Alice's Lamport binding (from AliceInput) and provides his Lamport-bound labels; co-signing is only required in the counter-proofs variant, not in the baseline.* (Subsection 6.7).

The co-signing mechanism ensures that Bob's circuit evaluation depends on the same SNARK that Alice committed to. *Bob's Lamport signatures reveal the labels necessary for Alice to evaluate Bob's circuit* (Subsection 6.7). The symmetry is essential: both parties evaluate complementary circuits, with *only one of the two secrets, P_a or P_b, will be revealed during the execution of the protocol* (Subsection 6.6).

Bob's circuit is designed with the opposite condition from Alice's. Subsection 6.6 explains: *Bob circuit is designed so that, if Alice's SNARK is correct, a secret, P_a, is revealed.*

This secret is revealed by Bob's circuit under the opposite condition of Alice's circuit. If Bob's evaluation reveals P_b, indicating Alice submitted an invalid assertion, Bob can *forfeit Alice's security deposit and prevent the reimbursement of the bridge funds* (Subsection 6.6).

Additionally, *Bob can also prevent the Reimbursement transaction from being executed by issuing the AliceLost transaction* (Subsection 6.8), which blocks Alice's reimbursement path even if Bob doesn't claim the deposits directly.

Algorithm 11 Bob State Machine

State: $s \in \{\textsc{Init}, \mathsf{W}_A, \mathsf{W}_B, \textsc{Eval}, \textsc{Done}\}$; heights h_0, h_A
procedure OnStart(h) $s \leftarrow \mathsf{W}_A$; $h_0 \leftarrow h$
end procedure
procedure Tick(h)
  if FLEX2 and Finalize_k(TryEarlyRefund) and $\neg$(Finalize_k(MissingBobDeposit) $\vee$ Finalize_k(NoBobChallenge)) **then**
    broadcast StillOpen
  end if
  if $s = \mathsf{W}_A$ **then**
    if Finalize_k(AliceDeposit)$\wedge$Finalize_k(AliceInput) **then** $h_A \leftarrow h$; $s \leftarrow \mathsf{W}_B$
    end if
    if $h \geq h_0 + \Delta_A$ **and** $\neg$(Finalize_k(AliceDeposit) $\wedge$ Finalize_k(AliceInput)) **then**
      broadcast MissingAliceDeposit; $s \leftarrow$ Done
    end if
  else if $s = \mathsf{W}_B$ **then**
    broadcast BobDeposit, BobInput; **if** both k-final **then** $s \leftarrow$ Eval
    if $h \geq h_A + \Delta_B$ **and** $\neg$(Finalize_k(BobDeposit) $\wedge$ Finalize_k(BobInput)) **then**
      broadcast MissingBobDeposit; $s \leftarrow$ Done
    end if
  else if $s =$ Eval **then** run $\mathcal{G}_{\mathsf{GC}}$.Ev
    if output $= P_b$ **then** broadcast BobTakesDeposit and AliceLost; $s \leftarrow$ Done
    else $s \leftarrow$ Done
    end if
  end if
end procedure

6.1.3. *Watchtower.* Watchtowers play a crucial monitoring and enforcement role in FLEX, particularly for liveness violations. Section 3 notes that *the risk of a misappropriation incentivizes watchtowers to employ high-availability servers*, while acknowledging that *as the number of watchtowers increases, the associated risk decreases.*

The main responsibility of watchtowers is to monitor liveness and assist with protocol progress using actions whose validity is checkable from public on-chain state.

The penalty mechanism described in Subsection 6.4 is as follows: if Alice does not issue the AliceDeposit transaction before the deadline, Bob can impose a penalty by issuing MissingAliceDeposit; conversely, if Bob fails to issue BobDeposit within the deadline, Alice can issue MissingBobDeposit.

In FLEX's design one does *not* rely on watchtowers to *issue* the penalty transactions MissingAliceDeposit and MissingBobDeposit. Instead, these penalties are issued by the

counterparty (or, more generally, by an authorized operator that participated in the pre-signing setup and therefore holds a valid instance of the corresponding pre-signed penalty transaction). This keeps watchtowers outside the covenant-signing set, avoiding the large-multiplier signature blow-up that would arise if all watchtowers had to be included in verification/signing for the full transaction family.

FLEX2 extends the watchtower role with stall-closure capabilities. In particular, both DisputeIsStalled1 and DisputeIsStalled2 (and the safety attestation StillOpen) can be broadcast by any watchtower after the specified delays to force progress.

FLEX2 extends the watchtower role with stall-closure capabilities. Section 7 introduces *two transactions, DisputeIsStalled1 and DisputeIsStalled2, can be broadcast by any watchtower after specific delays to force progress.* These allow *third-party stall-closure* that *bound or bounds total value/time-at-risk across open disputes* even when neither primary party is advancing the protocol.

The early-refund mechanism does not rely on watchtowers in our design: StillOpen is issued by Bob (Fig. 3 in [11]) to invalidate premature early-refund attempts while the dispute is still open. Watchtowers (when present) are used only for publicly verifiable liveness actions, such as stall-closure.

6.2. **Ideal functionalities.**

6.2.1. FLEX *Protocol.* The FLEX ideal functionality captures the core dispute protocol between Alice and Bob for a single peg-out. Section 6 describes the protocol flow: the dispute begins with StartDispute, proceeds through three windowed phases (W_A, W_B, W_R), and concludes either with a winner determined by circuit evaluation or with reimbursement after timeouts.

The fundamental security property is that *only one of the two secrets, P_a or P_b, will be revealed during the execution of the protocol* (subsection 6.6). This one-of-two disclosure determines the dispute outcome: if P_a is revealed, Alice wins and can claim Bob's deposit; if P_b is revealed, Bob wins, claims Alice's deposit, and blocks reimbursement.

The windowed structure enforces timely participation. Subsection 6.4 specifies: *Alice must issue the StartDispute transaction. Alice deposits her security bond and publishes the SNARK circuit input OTS-signed by her in the AliceDeposit transaction, subject to penalty if not performed within the first timelock period. Bob deposits his security bond and publishes the SNARK circuit input subject to penalty if not performed within two timelock periods. Alice can be reimbursed after three timelock periods.*

Importantly, timeouts enable penalties and phase transitions but do not determine the cryptographic winner. Subsection 6.6 makes clear that winner determination depends on circuit evaluation: *Alice uses her circuit's input wire labels as pre-images for the corresponding Lamport public keys Bob uses the labels obtained to evaluate Alice's circuit. If the SNARK is invalid, Bob will obtain the secret output, P_b.* The winner bit reflects which secret was cryptographically revealed, not which party missed a deadline.

The protocol includes penalty mechanisms described in subsection 6.4: *If Alice does not issue the AliceDeposit transaction before the deadline, Bob can impose a penalty on Alice by issuing the MissingAliceDeposit transaction. Conversely, if Bob fails to issue the BobDeposit transaction within the deadline, Alice can issue the MissingBobDeposit transaction.*

Finally, the reimbursement mechanism operates as specified in subsection 6.4: *Alice can be reimbursed by issuing the Reimbursement transaction after three timelock periods*, but subsection 6.8 adds that *Bob can prevent the Reimbursement transaction from being executed by issuing the AliceLost transaction* when Bob has won the dispute.

Algorithm 12 $\mathsf{FLEX}(\Delta_A, \Delta_B, \Delta_R, k_{\max}, \mathsf{Leak})$

State: $b \leftarrow \bot$; $h_0, h_A, h_B \leftarrow \bot$; windows $\mathsf{W}_A, \mathsf{W}_B, \mathsf{W}_R \leftarrow \emptyset$
$\mathsf{View}_{\mathcal{F}} \leftarrow \emptyset$
Const: $\Omega^{\mathsf{FLEX}}_{\mathsf{win}} = \{\mathsf{AliceTakesDeposit}, \mathsf{BobTakesDeposit}, \mathsf{AliceLost}, \mathsf{Reimburse}\}$
Decoder: $\mathsf{De}^{\mathsf{FLEX}} : \Omega^{\mathsf{FLEX}}_{\mathsf{win}} \to \{0,1\}$ defined by $\mathsf{De}^{\mathsf{FLEX}}(\mathsf{AliceTakesDeposit}) = \mathsf{De}^{\mathsf{FLEX}}(\mathsf{Reimburse}) = 1$ (Alice wins), $\mathsf{De}^{\mathsf{FLEX}}(\mathsf{BobTakesDeposit}) = \mathsf{De}^{\mathsf{FLEX}}(\mathsf{AliceLost}) = 0$ (Bob wins).
Interfaces:
(1) Receives $\mathcal{G}_{\mathsf{Ledger}}.\mathsf{final}_k(\mathsf{tx}, h)$ notifications
(2) Messages from parties mirror real protocol control (no secret values)
(3) **Simulator interface:** Sim receives $(\mathsf{Leak}(\mathsf{View}_{\mathcal{F}}), o)$ where o is the output
procedure OnFinalK(tx, h)
  append $(\mathsf{final}_k(\mathsf{tx}), h)$ to $\mathsf{View}_{\mathcal{F}}$
  send $\mathsf{Leak}(\mathsf{View}_{\mathcal{F}})$ **to** Sim ▷ Simulator receives leakage from $\mathcal{F}$
  if $\mathsf{tx} = \mathsf{StartDispute}$ **and** $h_0 = \bot$ **then** $h_0 \leftarrow h$; $\mathsf{W}_A \leftarrow [h_0,\ h_0{+}\Delta_A)$
  end if
  if $\mathsf{tx} \in \{\mathsf{AliceDeposit}, \mathsf{AliceInput}\}$ **and** both are k-final **then**
    $h_A \leftarrow h$ if unset; $\mathsf{W}_B \leftarrow [h_A,\ h_A{+}\Delta_B)$
  end if
  if $\mathsf{tx} \in \{\mathsf{BobDeposit}, \mathsf{BobInput}\}$ **and** both are k-final **then**
    $h_B \leftarrow h$ if unset; $\mathsf{W}_R \leftarrow [\max\{h_0{+}\Delta_A,\ h_B\},\ \max\{h_0{+}\Delta_A,\ h_B\}{+}\Delta_R)$
  else if $\mathsf{W}_R = \emptyset$ **and** $h \geq h_0{+}\Delta_A$ **then** $\mathsf{W}_R \leftarrow [h_0{+}\Delta_A,\ h_0{+}\Delta_A{+}\Delta_R)$
  end if
  if $\mathsf{tx} \in \Omega^{\mathsf{FLEX}}_{\mathsf{win}}$ **and** $b = \bot$ **then**
    $b \leftarrow \mathsf{De}^{\mathsf{FLEX}}(\mathsf{tx})$; append $(\mathsf{winner} = b)$ to $\mathsf{View}_{\mathcal{F}}$
    let $o := P_a$ **if** $b = 1$ **else** $o := P_b$ ▷ the released secret
    send $(\mathsf{Leak}(\mathsf{View}_{\mathcal{F}}), o)$ **to** Sim ▷ Release winner's secret to Sim
    output o to the winning party; **halt**
  end if
end procedure

6.2.2. FLEX2. FLEX2 introduces enhancements over the base protocol, focusing on faster reimbursement when one party is unresponsive and protection against griefing attacks. Section 7 motivates the need: *In the simplified protocol, if Bob's node is down and cannot interact on-chain, the protocol waits for the full dispute period. This introduces an undesirable financial cost.*

Section 7 explains: *We first isolate the FLEX block from the bridge use case and make it generic. Then we extend it with additional features* this enforces order $\mathsf{AliceDeposit} \to \mathsf{BobDeposit} \to \mathsf{AliceInput} \to \mathsf{BobInput}$ by requiring (e_1, e_2, e_3) progressively. This ordering, shown in Figure 3, prevents Bob from initiating disputes without committing resources, addressing the *Non-refundable Circuit Input Publication Problem (NCIPP)* discussed in subsection 6.10.

Subsection 6.14 describes the solution: *Method 3: Separation of Deposit and Input Publication. We separate the deposit and input publication into two separate transactions. The input is only published after both parties have made their security bond deposits.* This *eliminates even minor incentive misalignment* by ensuring Bob deposits before Alice publishes her expensive Lamport-bound SNARK input.

Section 7 describes the process: *TryEarlyRefund can be broadcast by Alice after Bob's deadline in* W_B*, but only after a penalization transaction (such as MissingBobDeposit) has confirmed.* This is followed by a safety check: *StillOpen is an attestation (by any watchtower) that no winning spend has been seen within a safety buffer, preventing race conditions with late but valid challenges.* Finally, *EarlyRefund becomes valid after a short additional delay, allowing Alice to recover in two windows.*

Mutual exclusion is important for FLEX2's safety, indeed: Section 7 states that parameters are chosen so the cumulative CSV spacing dominates plausible reorg depth, and StillOpen is invalidated by any appearance of a winner's spend. The safety buffer Δ_S must exceed the reorg bound $k_{\max}$ to prevent attacks where a late winner-spend could invalidate an already-finalized StillOpen transaction.

Lemma 1 (FLEX2 Mutual Exclusion)**.** *Assume* $\Delta_S \geq k_{\max} + \rho'$ *and* $\Delta_S \geq \tau_A$ *(parameter alignment of Section 7.6). Then, except with probability* $\mathsf{negl}(\lambda)$ *over* $\mathcal{G}_{\mathsf{Ledger}}$*'s randomness:*

(1) Unconditional ledger-level exclusion: At most one of

$$\left\{\mathsf{tx} \in \Omega^{\mathsf{FLEX2}}_{\mathsf{win}} \setminus \{\textit{EarlyRefund}\} \text{ that consumes } \mathsf{pot}_A\right\} \quad \text{and} \quad \textit{EarlyRefund}$$

becomes k-finalized. This part requires no behavioural assumption on either party.

(2) Conditional exclusion of pre-pot_A*-spend races: If Bob follows protocol with respect to StillOpen then EarlyRefund cannot k-finalise once StillOpen has k-finalised, and conversely. This part covers the case where Bob would have been entitled to a winner-spend but had not yet broadcast it on-chain.*

Proof. **Part (1).** EarlyRefund consumes pot_A via ℓ^A_{Refund}. Every $\mathsf{tx} \in \Omega^{\mathsf{FLEX2}}_{\mathsf{win}} \setminus \{\mathsf{EarlyRefund}\}$ that affects pot_A also consumes it (through some leaf). The Bitcoin consensus rule forbids two transactions from consuming the same outpoint; once one is k-finalised, reorganising the other into a competing k-final chain requires a reorganisation of depth $\geq k_{\max}$, which occurs with probability $\mathsf{negl}(\lambda)$ by the common-prefix bound on $\mathcal{G}_{\mathsf{Ledger}}$. Part (1) follows.

Part (2). EarlyRefund also consumes the synchronisation UTXO S. If Bob follows protocol, he broadcasts StillOpen within $\Delta_S - k_{\max}$ blocks of $\mathsf{ht}[\mathsf{TryEarlyRefund}]$. Since $\Delta_S \geq k_{\max} + \rho'$, StillOpen accrues $k_{\max}$ confirmations before EarlyRefund becomes valid. Because StillOpen also consumes S, at most one of $\{\mathsf{StillOpen}, \mathsf{EarlyRefund}\}$ can k-finalise. The common-prefix bound rules out a reorganisation that flips this outcome, again with probability $\mathsf{negl}(\lambda)$. □

Remark 2 (Adversarial B and the soundness budget)**.** *For a corrupted challenger B who deliberately suppresses StillOpen, Part (2) does not apply. The relevant safety guarantee in that case is Theorem 2: if* $b^\star = 0$ *(Bob is the legitimate winner) and B nevertheless allows EarlyRefund to finalise, then this constitutes a wrong-winner event whose probability is bounded by the composite bound of Theorem 2. Conversely, if* $b^\star = 1$ *(Alice is the legitimate winner), a finalised EarlyRefund is the correct outcome and there is no safety violation. In short: Part (1) gives unconditional ledger-level mutual exclusion; Part (2) gives an additional safety hedge when Bob cooperates; and Theorem 2 absorbs the remaining adversarial scenarios under the cryptographic budget.*

The protocol also provides stall-closure hooks. Section 7 explains: *For the FLEX2 component to be used as part of larger multi-party protocol, we provide a method for external parties to close the dispute if the two parties do not interact.* The transactions DisputeIsStalled1 and DisputeIsStalled2 allow *any of the other parties can issue* these transactions to force

protocol termination, bounding the age (in blocks) of any active dispute and thereby limiting value-at-risk.

Algorithm 13 $\mathsf{FLEX2}(\Delta_A, \Delta_B, \Delta_R, \Delta_S, k_{\max}, \mathsf{Leak})$

As in Alg. 12, plus state: $h_T \leftarrow \perp$; guard window $W_S \leftarrow \emptyset$
Const: $\Omega_{\mathsf{win}}^{\mathsf{FLEX2}} := \Omega_{\mathsf{win}}^{\mathsf{FLEX}} \cup \{\mathsf{EarlyRefund}\}$
Decoder: $\mathsf{De}^{\mathsf{FLEX2}}$ extends $\mathsf{De}^{\mathsf{FLEX}}$ with $\mathsf{De}^{\mathsf{FLEX2}}(\mathsf{EarlyRefund}) = 1$ (Alice wins)
Simulator interface: as in Alg. 12; Sim receives $(\mathsf{Leak}(\mathsf{View}_{\mathcal{F}}), o)$ where o is the released secret as decoded by $\mathsf{De}^{\mathsf{FLEX2}}$
procedure OnFinalK(tx, h)
    process window logic as in Alg. 12
    send $\mathsf{Leak}(\mathsf{View}_{\mathcal{F}})$ **to** Sim
    if $\mathsf{W}_B \neq \emptyset$ **and** $h = \mathsf{end}(\mathsf{W}_B)$ **and** not both $\{\mathsf{BobDeposit}, \mathsf{BobInput}\}$ are k-final **then**
        issue virtual event TryEarlyRefund at height h; $h_T \leftarrow h$; $W_S \leftarrow [h_T, h_T + \Delta_S)$
        append to $\mathsf{View}_{\mathcal{F}}$; **send** $\mathsf{Leak}(\mathsf{View}_{\mathcal{F}})$ **to** Sim
    end if
    if tx = EarlyRefund **then require** $h \geq h_T + \Delta_S$
    end if
    if $\mathsf{tx} \in \Omega_{\mathsf{win}}^{\mathsf{FLEX2}}$ **and** $b = \perp$ **then**
        $b \leftarrow \mathsf{De}^{\mathsf{FLEX2}}(\mathsf{tx})$; append $(\mathsf{winner} = b)$ to $\mathsf{View}_{\mathcal{F}}$
        let $o := P_a$ **if** $b = 1$ **else** $o := P_b$
        send $(\mathsf{Leak}(\mathsf{View}_{\mathcal{F}}), o)$ **to** Sim; **output** o; **halt**
    end if
end procedure

6.3. **Protocols π_{FLEX} and π_{FLEX2}.** We now specify the concrete protocols that run in the $(\mathcal{G}_{\mathsf{Hash}}, \mathcal{G}_{\mathsf{Sig}}, \mathcal{G}_{\mathsf{SNARK}}, \mathcal{G}_{\mathsf{GC}}, \mathcal{G}_{\mathsf{Ledger}})$-hybrid world.

Algorithm 14 π_{FLEX}

Setup: exchange Lamport commitments; run $\mathcal{G}_{\mathsf{SNARK}}$ binding proof; garble CDS; prepare pots $\mathsf{pot}_A, \mathsf{pot}_B$; pre-sign SIGHASH_ANYONECANPAY deposits
StartDispute $\rightarrow$ open W_A; Alice broadcasts AliceDeposit, AliceInput within Δ_A else MissingAliceDeposit
On k-final A: open W_B; Bob broadcasts BobDeposit, BobInput within Δ_B else MissingBobDeposit
Evaluate CDS: output P_a (Alice wins $\Rightarrow$ AliceTakesDeposit) or P_b (Bob wins $\Rightarrow$ BobTakesDeposit, AliceLost)
If no winner spend: after Δ_R output Reimburse

Algorithm 15 π_{FLEX2}

As in π_{FLEX} plus enablers that enforce order; stall-closure by any watchtower
If W_B expires with Bob incomplete: TryEarlyRefund; then either Bob counters with StillOpen, or after Δ_S Alice broadcasts EarlyRefund.

6.4. **Leakage definition.** Let View be the real execution transcript. The leakage function Leak maps View to the following tuple:

$$\mathsf{Leak}(\mathsf{View}) = (\mathsf{TX\text{-}Shape},\ \mathsf{LeafIdx},\ \mathsf{Amounts},\ \mathsf{Timings},\ \mathsf{Fees},\ \mathsf{Winners}),$$

where:

(1) TX-Shape: multiset of transaction types observed on-chain (StartDispute, AliceDeposit, BobDeposit, AliceInput, BobInput, penalties, Reimburse, TryEarlyRefund, StillOpen, EarlyRefund, winner-spends), without payloads beyond script templates.

(2) LeafIdx: which Taproot leaf indices were exercised (hashlock vs owner-after-CSV).

(3) Amounts: output amounts of pots and reimbursements.

(4) Timings: block heights for confirmations and window expiries.

(5) Fees: feerates of confirmed transactions (to the extent inferable on-chain).

(6) Winners: a one-bit outcome (Alice/Bob) if a winner-spend or AliceLost is observed.

Definition 1. *Let $\mathcal{E}$ be the alphabet of events consisting of:*

(1) off-chain: $\mathit{msg}(u \to v, m)$;

(2) on-chain: $\mathit{conf}(\mathsf{tx})$, $\mathit{final}_k(\mathsf{tx})$ (finalized at depth k), and $\mathit{spend}(o \xrightarrow{tx} _)$ for UTXO o spent by tx;

(3) timers: $\mathit{open}(W, h)$, $\mathit{expire}(W, h)$ for windows $W \in \{\mathsf{W}_A, \mathsf{W}_B, \mathsf{W}_R\}$ at height h, and FLEX2 guards $\mathit{open}(\Delta_S, h)$.

An execution trace is a finite sequence $\mathsf{Hist} = (e_1, \ldots, e_t) \in \mathcal{E}^*$ consistent with the state machines of Section 6.1 and the ledger model of Section 4. We write $\mathsf{Hist}' \preceq \mathsf{Hist}$ if Hist' is a prefix of Hist.

Definition 2 (Enabled predicates and knowledge predicates)**.** *We say a transaction tx is enabled in a trace Hist if:*

(1) All protocol preconditions for tx hold in the state induced by Hist;

(2) $\mathcal{G}_{\mathsf{Ledger}}.\textsc{PolicyAdmissible}(tx) = 1$; and

(3) for every tx' already confirmed in Hist, $\neg\, \mathcal{G}_{\mathsf{Ledger}}.\textsc{Conflicts}(tx, tx')$.

We write $\mathsf{Enabled}(\mathsf{tx}, \mathsf{Hist}) \in \{0, 1\}$. For an artifact X, we write $\mathsf{knows}(P, \mathsf{X}, \mathsf{Hist}) \in \{0, 1\}$ to denote that party P possesses X at the end of Hist.

Definition 3 (Window schedule contract)**.** *Fix the block height h_0 at which StartDispute is confirmed. Let W_A be the interval $[h_0, h_0 + \Delta_A)$ and W_B the interval that opens at the first height $h_A \in \mathsf{W}_A$ where $\{\mathit{AliceDeposit}, \mathit{AliceInput}\}$ are both confirmed and lasts Δ_B blocks; if no such h_A exists, skip to W_R, which opens at $\max\{h_0 + \Delta_A, h_B\}$ where h_B is the first height both $\{\mathit{BobDeposit}, \mathit{BobInput}\}$ are confirmed (if any), and lasts Δ_R blocks. In FLEX2, TryEarlyRefund becomes eligible at the end of W_B if $\{\mathit{BobDeposit}, \mathit{BobInput}\}$ were not both confirmed. The guard Δ_S opens at that eligibility height and closes at $+\Delta_S$.*

Choose positive margins ρ, ρ', ρ'' and set $\Delta_A + \Delta_B \geq k_{\max} + \rho$, $\Delta_S \geq k_{\max} + \rho'$, and $\Delta_R \geq k_{\max} + \rho''$. These ensure safety against $k_{\max}$-deep reorganizations and provide liveness slack.

Definition 4 (Safety invariants as prefix-closed predicates)**.** *Let $\Phi_1, \ldots, \Phi_6$ be predicates over traces. We say Φ is a safety invariant if for every trace Hist and every prefix $\mathsf{Hist}' \preceq \mathsf{Hist}$, $\Phi(\mathsf{Hist}') = 1$. The invariants for our protocol are:*

(1) I_1 *(Input binding): For each input wire* i*, the set*

$$\mathcal{O}_i(\mathsf{Hist}) = \{s : s \text{ opens some label for } i \text{ and is accepted}\} \text{ has size at most } 1.$$

(2) I_2 *(CDS one-of-two): Either* $\mathsf{knows}(\mathit{Alice}, P_a, \mathsf{Hist})$ *or* $\mathsf{knows}(\mathit{Bob}, P_b, \mathsf{Hist})$ *holds.*

(3) I_3 *(Pot enforceability): If* $\mathit{spend}(\mathsf{pot}_U \xrightarrow{tx} _)$ *appears before the refund* CSV *height, then* tx *satisfies the designated-signer and the matching hashlock preimage check.*

(4) I_4 *(Early-refund safety): If* StillOpen *finalizes at height* h_s *and was issued at height* h_t *with* $h_s - h_t \geq \Delta_S$*, then no winner-spend can subsequently finalize in any extension; conversely, if a winner-spend finalizes before* StillOpen*, then* EarlyRefund *cannot be enabled thereafter.*

Proposition 1. *Assume* $\Delta_S \geq k_{\max} + \rho'$*. If* EarlyRefund *finalizes at height* h_e*, then there exists no extension of the trace in which a winner-spend finalizes; conversely, if a winner-spend finalizes at height* h_w*, then* EarlyRefund *cannot be enabled thereafter.*

Proof. Invariant I_4 is the prefix-closed version of Lemma 1 Part (1): at most one of EarlyRefund and a winner-spend that consumes pot_A becomes k-finalised in any extension of the trace. If both became k-finalised in some extension, two transactions would share the consumed outpoint pot_A, contradicting the common-prefix bound on $\mathcal{G}_{\mathsf{Ledger}}$. □

7. Cryptographic components

We give exact, implementable constructions of each cryptographic component and state the security properties each must satisfy for the proofs in Sections 6–9. Where alternatives exist, we parameterize the construction so the security theorems depend only on the abstract properties (correctness, privacy, authenticity, knowledge-soundness, etc.).

7.1. **Lamport OTS-based input binding for bitwise labels.** For each input bit (wire) $i \in \{1, \ldots, n\}$ of Alice's private input $x \in \{0,1\}^n$ to the CDS circuit, we bind the two possible labels to public commitments so that:

(1) The evaluator can locally verify the binding.

(2) Revealing both labels is detectable and cryptographically meaningful.

(3) The setup proof can attest to correctness of the binding without revealing the labels.

Fix output length $n_\lambda = \Theta(\lambda)$.

(1) For each wire $i \in [n]$ and bit $b \in \{0,1\}$ we derive the GC input label by

$$K_{i,b} := \mathsf{HKDF}\big(\mathsf{ikm} = s_{i,b}, \mathsf{salt} = \langle i, b, \mathsf{ctx}\rangle, \mathsf{info} = \texttt{FLEX:LABEL}\big) \in \{0,1\}^{n_\ell},$$

where $s_{i,b} \xleftarrow{\$} \{0,1\}^{n_\lambda}$ is the committed seed for wire (i,b), ctx is a per-dispute domain-separation string, and n_ℓ is the garbling label length.

(2) Publish the commitments $y_{i,b} := \mathsf{H}(\texttt{FLEX:LAMPORT} \,\|\, s_{i,b})$ for all (i,b).

(3) Let $Y = \{(y_{i,0}, y_{i,1})\}_{i=1}^n$ be Alice's public binding commitments.

Given a candidate label K for wire i and a purported opening s, accept if, and only if $K = \mathsf{HKDF}(s; \langle i, b\rangle, \texttt{FLEX:LABEL})$ for some $b \in \{0,1\}$ and $\mathsf{H}(\texttt{FLEX:LAMPORT} \,\|\, s) = y_{i,b}$.

If H is preimage/second-preimage resistant and HKDF is a PRF/KDF, then any accepted label must be derived from a committed seed; presenting two accepted openings for the

same wire implies finding two preimages with distinct (i, b) tags that both map to the same $y_{i,b}$, which is infeasible. The binding also composes with GC authenticity: a label not derived from a committed seed will fail gate checks with overwhelming probability.

Definition 5. *Let* $F_s(\cdot) := \mathsf{HKDF}(\mathit{ikm} = s, \mathit{salt} = \cdot, \mathit{info} = \texttt{FLEX:LABEL})$ *for a secret key* $s \xleftarrow{\$} \{0,1\}^{n_\lambda}$*, and let* R *be a uniformly random function mapping salts to* $\{0,1\}^{n_\ell}$*. A distinguisher* $\mathcal{D}$ *is given oracle access to* $\mathcal{O}$*, which is either* F_s *or* R*, and outputs a bit* b'*. Define* $\mathsf{Adv}^{\mathit{prf}}_{\mathsf{HKDF}}(\mathcal{D}) := \Big|\Pr[b' = 1 \mid \mathcal{O} = F_s] - \Pr[b' = 1 \mid \mathcal{O} = R]\Big|$*. We say* HKDF *is a PRF if* $\mathsf{Adv}^{\mathit{prf}}_{\mathsf{HKDF}}(\mathcal{D}) = \mathsf{negl}(\lambda)$ *for all PPT distinguishers* $\mathcal{D}$*.*

7.2. **Setup proof binding labels to Lamport public keys.** Let $vp \leftarrow \mathsf{Setup}(1^\lambda)$ be per-dispute verifier parameters for a SNARK with completeness, knowledge-soundness, and zero-knowledge. Define the relation $\mathcal{R}_{\mathsf{bind}}$ with instances $x_{\mathsf{bind}} := \big(Y = \{y_{i,b}\}_{i\in[n],b\in\{0,1\}}, \ell = n, n_\ell\big)$, and witnesses $w_{\mathsf{bind}} := \{s_{i,b}\}_{i,b}$, such that the circuit C_{bind} checks for all (i, b):

$$\mathsf{H}(\texttt{FLEX:LAMPORT} \parallel s_{i,b}) = y_{i,b} \quad \text{for all } i \in [n],\ b \in \{0,1\}.$$

Alice computes $\pi_{\mathsf{bind}} \leftarrow \mathsf{Prove}(vp, x_{\mathsf{bind}}, w_{\mathsf{bind}})$ and publishes (Y, π_{bind}). The verifier checks $\mathsf{Verify}(vp, x_{\mathsf{bind}}, \pi_{\mathsf{bind}}) = 1$.

(1) Completeness: honest Alice always convinces the verifier.

(2) Knowledge-soundness: any prover that convinces the verifier knows preimages $\{s_{i,b}\}$ for Y.

(3) Zero-knowledge (optional): the proof leaks nothing beyond (Y, ℓ, n_ℓ).

Role of the setup proof. π_{bind} is a *seed-knowledge* proof, not a direct label-binding proof: knowledge-soundness of $\mathcal{G}_{\mathsf{SNARK}}$ extracts witnesses $\{s_{i,b}\}$ such that $\mathsf{H}(\texttt{FLEX:LAMPORT} \parallel s_{i,b}) = y_{i,b}$ for every (i, b). The labels $K_{i,b} = \mathsf{HKDF}(s_{i,b}; \langle i, b\rangle, \texttt{FLEX:LABEL})$ are then deterministic public functions of the seeds. Binding of labels to commitments is enforced at evaluation time by the joint check of Section 7.1, which on a candidate (K, s) for wire i checks both $K = \mathsf{HKDF}(s; \langle i, b\rangle, \texttt{FLEX:LABEL})$ and $\mathsf{H}(\texttt{FLEX:LAMPORT} \parallel s) = y_{i,b}$ for some $b \in \{0,1\}$. The setup proof is therefore what commits Alice to a specific pair of seeds per wire ahead of time and prevents her from later switching Y to a different commitment.

7.3. **CDS-mode garbled circuits.** We design a garbling that releases exactly one of two secrets (P_a, P_b) depending on a predicate $f(x, y) \in \{0,1\}$ over Alice's input x and Bob's input y, while preserving privacy and authenticity.

Given $f : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ and two secrets $(P_a, P_b) \in \{0,1\}^{n_\lambda} \times \{0,1\}^{n_\lambda}$ with $\mathsf{H}(P_a) = V_a$, $\mathsf{H}(P_b) = V_b$, define

$$C_{f,P_a,P_b}(x, y) := \begin{cases} P_a & \text{if } f(x, y) = 1, \\ P_b & \text{if } f(x, y) = 0. \end{cases}$$

Let $\mathsf{GC} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev})$ be any garbling with correctness, privacy, and authenticity. We fix:

$$(\widetilde{C}, \{K_{i,0}, K_{i,1}\}_{i\in[n+m]}, \mathsf{dec}) \leftarrow \mathsf{Gb}(1^\lambda, C_{f,P_a,P_b}); \\ L_x \leftarrow \mathsf{En}(\{K_{i,b}\}, x); L_y \leftarrow \mathsf{En}(\{K_{n+j,b}\}, y).$$

Alice replaces her n labels $\{K_{i,b}\}_{i\le n}$ by the KDF-derived labels from Section 7.1 (same bit order), and augments each input gate with a check that rejects any label failing the binding verification against Y.

For any (x, y), $\mathsf{Ev}(\widetilde{C}, L_x, L_y)$ decodes to $C_{f,P_a,P_b}(x, y) \in \{P_a, P_b\}$ with probability $1 - \mathsf{negl}(\lambda)$; by authenticity, producing both outputs on any input has probability $\mathsf{negl}(\lambda)$.

There exists a simulator Sim such that $(\widetilde{C}, L_x, L_y)$ is simulatable given $(|x|, |y|, f(x, y))$ and the resulting output; the unrevealed secret remains hidden computationally.

Definition 6 (CDS Authenticity)**.** *Fix a CDS circuit C_{f,P_a,P_b} and a garbling algorithm Gb. An adversary $\mathcal{A}$ receives a single garbling $\widetilde{C} \leftarrow \mathsf{Gb}(1^\lambda, C_{f,P_a,P_b})$and at most one input label per wire (for both parties' inputs). We say* CDS authenticity *holds if for every PPT $\mathcal{A}$, the probability that $\mathcal{A}$ outputs both secrets P_a and P_b (i.e., values p_a, p_b satisfying $\mathsf{H}(p_a) = V_a$ and $\mathsf{H}(p_b) = V_b$) is negligible. We denote the best such success probability by $\mathsf{Adv}^{\mathsf{auth}}_{\mathsf{CDS}}$.*

For the analysis of FLEX, we will require:

(1) CDS specialization: We specialize C to implement the predicate-controlled release of one among two secrets $(s_0, s_1) \in \{0,1\}^* \times \{0,1\}^*$:

$$C_{s_0,s_1}(x, y) = \begin{cases} s_0 & \text{if } f(x, y) = 0, \\ s_1 & \text{if } f(x, y) = 1, \end{cases}$$

where $f : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$. A CDS-capable garbling realizes:

(a) CDS correctness: If $f(x, y) = b$, then Ev yields s_b with probability $1 - \mathsf{negl}(\lambda)$.

(b) One-of-two secrecy: For any PPT evaluator (even maliciously choosing y adaptively), its view is computationally indistinguishable from one that contains s_b but hides s_{1-b}; that is, learning both s_0 and s_1 has negligible probability.

(2) Input binding through one-time signatures: In malicious-garbler/evaluator settings, we additionally assume *bitwise input binding*: each input bit $b \in \{0,1\}$ for wire i is tied to a public key $(y_{i,0}, y_{i,1})$ such that opening both labels for i implies breaking EUF-CMA of the underlying OTS.

7.4. **Taproot pot scripts.** Following FLEX, all enforcement is through script-path spends; the key path is not used by the protocol and plays no role in the analysis. Concretely, each pot pot_A (resp. pot_B) exposes three leaves:

$$\begin{aligned} \ell^A_{\mathsf{BobWins}} &: \quad \mathsf{pk_k}(B) \ \wedge \ \mathsf{SHA256}(V_b), \\ \ell^A_{\mathsf{AliceWins}} &: \quad \mathsf{pk_k}(A) \ \wedge \ \mathsf{SHA256}(V_a), \\ \ell^A_{\mathsf{Refund}} &: \quad \mathsf{pk_k}(A) \ \wedge \ \mathsf{older}(\tau_A), \end{aligned}$$

and symmetrically for pot_B with roles swapped and τ_B. Here $\mathsf{pk_k}$ denotes a Schnorr Taproot key check, $\mathsf{SHA256}$ checks a preimage against $V \in \{V_a, V_b\}$, and older is a CSV relative timelock. Our proofs reason only about these script-path predicates.

Under Schnorr EUF-CMA and H preimage/second-preimage resistance, the only ways to spend pot_A before τ_A are Bob presents P_b with a valid Schnorr signature under B, or Alice presents P_a under A. Otherwise the only redemption is $\texttt{older}(\tau_A)$ by Alice.

7.5. **On-demand bonds through** SIGHASH_ANYONECANPAY**.** We make deposits on-demand, that is, allow assembling any set of inputs at dispute time while ensuring the pot outputs are fixed and pre-signed.

For party $U \in \{A, B\}$:

(1) Outputs: a single $\mathsf{Taproot}$ output paying to pot_U of value v_U (plus optional change).

(2) Sighash: each input signature is made with flag SIGHASH_ANYONECANPAY, covering all outputs and only the signing input, so additional inputs can be appended later without invalidating prior signatures.

We pre-sign one input, or we distribute a PSBT with a partially signed input, to be finalized later.

SIGHASH_ANYONECANPAY enables on-demand input aggregation: any UTXOs $\{I_j\}$ with value greater that v_U + fees can be appended as inputs while preserving the pre-signed pot_U outputs and prior signature validity.

Since SIGHASH_ANYONECANPAY signatures commit to all outputs, no adversary can alter the destination (the pot) without invalidating signatures. Thus, on-demand input aggregation does not weaken output integrity.

7.6. **FLEX2 interlocks.** One encodes the intended order

$$\mathsf{AliceDeposit} \to \mathsf{BobDeposit} \to \mathsf{AliceInput} \to \mathsf{BobInput}$$

through enabler UTXOs and pre-signed transactions that consume them in sequence; add stall-closure and early-refund paths that are mutually exclusive with winner spends by construction.

One creates three 1-sat enabler UTXOs: E_1 := Alice can win, E_2 := Bob challenge enabler, and E_3 := Alice tries early refund.

Transactions must consume them in sequence: AliceDeposit spends E_1 and creates E_2; BobDeposit spends E_2 and gates acceptance of AliceInput; AliceInput creates a short-lived UTXO that permits BobInput.

The early-refund path comprises three transactions and one synchronisation UTXO:

(1) TryEarlyRefund: spends E_3 and creates a short-lived synchronisation UTXO S.

(2) EarlyRefund: a *two-input* transaction that consumes *both* S and pot_A via the CSV-refund leaf ℓ^A_{Refund}. The transaction is valid only when (i) S has aged Δ_S blocks and (ii) τ_A has elapsed on the pot_A outpoint. Parameters are chosen so that $\Delta_S \geq \tau_A$.

(3) StillOpen: a one-input transaction spending S, issued by Bob (Section 6.1.3) upon observing TryEarlyRefund while the dispute is still open. StillOpen confirms a negligible-value output and serves purely to invalidate any subsequent EarlyRefund attempt.

The construction creates two distinct conflict edges for EarlyRefund:

(1) Conflict on S: {EarlyRefund, StillOpen} both consume S; at most one can finalise.

(2) Conflict on pot_A: {EarlyRefund, BobTakesDeposit, AliceTakesDeposit, Reimburse} all consume pot_A (through different leaves); at most one can finalise.

The first edge handles the case "Bob disputes the early refund"; the second handles the case "Bob (or a third party) completes a normal winner-spend or reimbursement".

Remark 3. *The requirement $\Delta_S \geq \tau_A$ is consistent with the margin conditions of Definition 3 in Section 6.4: with $\Delta_S \geq k_{\max} + \rho'$ and τ_A set to the conventional refund delay used by Reimburse, the relation $\Delta_S \geq \tau_A$ follows from $\tau_A \leq \Delta_A + \Delta_B + \Delta_R - k_{\max}$.*

8. Security definitions

All probabilities are taken over honest parties' randomness, the stochasticity of $\mathcal{G}_{\mathsf{Ledger}}$, and the PPT adversary's coins. The adversary may schedule messages and craft conflicting transactions subject to $\mathcal{G}_{\mathsf{Ledger}}$'s rules. Privacy is leakage-bounded by Leak (subsection 6.4).

Definition 7 (Correctness)**.** *Fix inputs (x, y) to the dispute predicate $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$. Consider an execution of π_{FLEX} (or π_{FLEX2}) in which both principals are honest and $\mathcal{G}_{\mathsf{Ledger}}$ confirms any policy-admissible transaction submitted with a market-rate fee within its allotted window; formally: any honest party can ensure that a policy-admissible transaction it broadcasts confirms within $k_{\max}$ blocks, capturing the effect of fee bumping and the absence of mempool "pinning" attacks that artificially delay confirmation. The protocol is correct if, except with probability $\mathsf{negl}(\lambda)$:*

(1) If $f(x, y) = 1$, Alice learns P_a while Bob does not learn P_b, a valid winner-spend for Alice exists on-chain, and AliceLost does not finalize (hence Reimburse is not blocked).

(2) If $f(x, y) = 0$, the symmetric statements hold for Bob and P_b, and AliceLost finalizes, blocking Reimburse.

Definition 8 (Input Binding)**.** *Let the binding commitments $Y = \{y_{i,b}\}$ and setup proof π_{bind} be as in Section 7.1 and Section 7.2. Consider the following experiment with adversary $\mathcal{A}$ playing the role of the evaluator:*

(1) The challenger samples honest seeds $\{s_{i,b}\}_{i,b}$ and computes Y, π_{bind}, and the label family $\{K_{i,b}\}_{i,b}$ as in Section 7.1 and Section 7.2.

(2) $\mathcal{A}$ gets (Y, π_{bind}, vp) and, for each i, at most one label $K_{i,b_i^\sharp}$ (wire's intended bit).

(3) $\mathcal{A}$ wins if it outputs an index i^* and two distinct openings (s'_0, s'_1) both accepted by the binding check of Section 7.1 for the two bit values 0 and 1.

Define $\mathsf{Adv}^{\mathsf{bind}}_{\mathsf{Lamport/Setup}}(\mathcal{A})$ as the probability that $\mathcal{A}$ wins. The scheme satisfies input binding if $\mathsf{Adv}^{\mathsf{bind}}_{\mathsf{Lamport/Setup}}(\mathcal{A}) = \mathsf{negl}(\lambda)$ for all PPT $\mathcal{A}$.

Definition 9 (CDS One-of-Two Secrecy)**.** *In the CDS garbling of Section 7.3:*

(1) Exclusivity: For all PPT $\mathcal{A}$, the probability that $\mathcal{A}$ outputs both P_a and P_b (hashing to V_a, V_b) is $\mathsf{negl}(\lambda)$.

(2) Leakage-bounded hiding: There exists a PPT simulator Sim such that for every PPT distinguisher $\mathcal{D}$,

$$\Big|\Pr[\mathcal{D}(\mathsf{View}_{\mathsf{Real}}) = 1] - \Pr[\mathcal{D}(\mathsf{Sim}(\mathsf{Leak}(\mathsf{View}_{\mathsf{Real}}))) = 1]\Big| = \mathsf{negl}(\lambda),$$

where $\mathsf{View}_{\mathsf{Real}}$ is the evaluator's real view and Leak is as in Section 6.4.

Definition 10 (On-chain Enforceability)**.** *Let $\mathsf{Adv}^{\mathsf{enf}}_{\mathsf{pot}}(\mathcal{A})$ be the probability that a PPT $\mathcal{A}$ causes $\mathcal{G}_{\mathsf{Ledger}}$ to finalize a pre-CSV spend of a pot that is not a designated-signer and matching-hashlock spend. The protocol has on-chain enforceability if $\mathsf{Adv}^{\mathsf{enf}}_{\mathsf{pot}}(\mathcal{A}) = \mathsf{negl}(\lambda)$ for all PPT $\mathcal{A}$.*

Definition 11 (Soundness)**.** *Let f be the ground-truth predicate. Define a wrong-winner event as one where $\mathcal{G}_{\mathsf{Ledger}}$ finalizes a winner outcome contradicting $f(x, y)$ (detected by a winner-spend or AliceLost). The protocol is sound if for all PPT $\mathcal{A}$ the probability of a wrong-winner event is $\mathsf{negl}(\lambda)$.*

Definition 12 (Fairness)**.** *The protocol provides weak on-chain fairness if, for all PPT $\mathcal{A}$:*

(1) Winner realizability conditioned on correctness, the honest winner can execute the corresponding winner-spend with probability $1 - \mathsf{negl}(\lambda)$.

(2) No theft the losing party cannot extract value from the opponent's pot before CSV *without the correct preimage and designated signature, except with probability $\mathsf{negl}(\lambda)$.*

Definition 13 (Leakage-bounded Privacy)**.** *Consider the real-world execution of π_{FLEX} (or π_{FLEX2}). Let $\mathsf{View}_{\mathsf{Real}}$ denote the adversary's view, and let $o \in \{P_a, P_b\}$ denote the unique secret released by the protocol. The protocol satisfies* standalone *leakage-bounded privacy if there exists a PPT simulator $\mathsf{Sim}_{\mathsf{st}}$ such that for every PPT distinguisher $\mathcal{D}$,*

$$\Big|\Pr[\mathcal{D}(\mathsf{View}_{\mathsf{Real}}) = 1] - \Pr[\mathcal{D}(\mathsf{Sim}_{\mathsf{st}}(\mathsf{Leak}(\mathsf{View}_{\mathsf{Real}}), o)) = 1]\Big| = \mathsf{negl}(\lambda),$$

where Leak *is as defined in subsection 6.4.*

Remark 4. *Definition 13 is a* standalone *simulation-based notion where the simulator is explicitly given $\mathsf{Leak}(\mathsf{View}_{\mathsf{Real}})$. For the UC formulation, the simulator instead receives leakage through its interface with the ideal functionality $\mathcal{F}$. We bridge these notions via Lemma 2 below, which shows the two leakage sources are identically distributed.*

Lemma 2 (Leakage Equivalence)**.** *Let $\mathsf{View}_{\mathsf{Real}}$ be the real-world transcript of π_{FLEX} (or π_{FLEX2}) and let $\mathsf{View}_{\mathcal{F}}$ be the internal view maintained by the ideal functionality* FLEX *(or* FLEX2*). For the leakage function* Leak *defined in subsection 6.4:*

$$\mathsf{Leak}(\mathsf{View}_{\mathsf{Real}}) \equiv \mathsf{Leak}(\mathsf{View}_{\mathcal{F}}),$$

where $\equiv$ denotes identical distributions over executions with the same inputs and randomness.

Proof. The leakage function Leak outputs the tuple

$$(\mathsf{TX\text{-}Shape}, \mathsf{LeafIdx}, \mathsf{Amounts}, \mathsf{Timings}, \mathsf{Fees}, \mathsf{Winners}).$$

We show each component is identically determined in both worlds.

Both the real protocol π and the ideal functionality $\mathcal{F}$ receive transaction confirmation notifications from the shared global functionality $\mathcal{G}_{\mathsf{Ledger}}$. Specifically:

(1) $\mathcal{G}_{\mathsf{Ledger}}$ issues $\mathsf{final}_k(\mathsf{tx}, h)$ notifications to all registered parties.

(2) In the real world, honest parties and the adversary observe these notifications.

(3) In the ideal world, $\mathcal{F}$ receives the same notifications (Algorithm 12, line 1) and appends them to $\mathsf{View}_{\mathcal{F}}$.

Since $\mathcal{G}_{\mathsf{Ledger}}$ is a global functionality shared between the real and ideal executions in the hybrid model, the sequence of $(\mathsf{final}_k(\mathsf{tx}, h))$ events is identical in both worlds for any fixed adversary strategy and environment inputs.

Each component of Leak is a deterministic function of confirmed transactions:

(1) TX-Shape: Determined by which transactions from $\mathcal{T}$ are confirmed.

(2) LeafIdx: Determined by the Taproot control blocks in confirmed spends.

(3) Amounts: Determined by the output values in confirmed transactions.

(4) Timings: Determined by the confirmation heights h in notifications.

(5) Fees: Determined by input/output value differences in confirmed transactions.

(6) Winners: Determined by which $\mathsf{tx} \in \Omega_{\mathsf{win}}^{\mathsf{FLEX2}}$ is confirmed.

Since both $\mathsf{View}_{\mathsf{Real}}$ and $\mathsf{View}_{\mathcal{F}}$ are constructed from the same $\mathcal{G}_{\mathsf{Ledger}}$ notifications, and Leak is a deterministic function of these notifications, we have $\mathsf{Leak}(\mathsf{View}_{\mathsf{Real}}) = \mathsf{Leak}(\mathsf{View}_{\mathcal{F}})$ with probability 1. □

Definition 14 (UC Realization)**.** *In the $(\mathcal{G}_{\mathsf{Ledger}}, \mathcal{G}_{\mathsf{Hash}}, \mathcal{G}_{\mathsf{Sig}}, \mathcal{G}_{\mathsf{SNARK}}, \mathcal{G}_{\mathsf{GC}})$-hybrid, a protocol π UC-realizes a functionality $\mathcal{F}$ if for every PPT adversary $\mathcal{A}$ there exists a PPT simulator Sim such that for every PPT environment $\mathcal{Z}$, the execution with $(\pi, \mathcal{A})$ is indistinguishable from the execution with $(\mathcal{F}, \mathsf{Sim})$.*

9. Proofs of security

We prove the security properties of π_{FLEX} and π_{FLEX2} in the $(\mathcal{G}_{\mathsf{Ledger}}, \mathcal{G}_{\mathsf{Hash}}, \mathcal{G}_{\mathsf{Sig}}, \mathcal{G}_{\mathsf{SNARK}}, \mathcal{G}_{\mathsf{GC}})$-hybrid model. Reductions explicitly track which primitive is broken in case of an adversary's success.

Remark 5. *Our security analysis (Section 9) uses the UC framework of Canetti* [4]*, with $\mathcal{G}_{\mathsf{Ledger}}$ and $\mathcal{G}_{\mathsf{Hash}}$ modelled as global functionalities and the remaining ideal functionalities ($\mathcal{G}_{\mathsf{Sig}}, \mathcal{G}_{\mathsf{SNARK}}, \mathcal{G}_{\mathsf{GC}}, \mathsf{FLEX}, \mathsf{FLEX2}$) in the hybrid model. The recently updated UC framework of Chevalier and Sageloli* [5] *addresses several subtleties in the original formulation; its composition theorem applies whenever the protocol is* sub-party respecting *and* sid-exposing*. Our protocols π_{FLEX} and π_{FLEX2} satisfy both hypotheses by inspection: every sub-party identity (Alice, Bob, and in FLEX2 the permissioned watchtowers) is determined by the dispute instance, and the dispute identifier ctx_i is an explicit parameter of every wire-label derivation, Lamport commitment, and pre-signed transaction. Theorem 4 and the sequential composition of Theorem 5 therefore remain valid under the updated framework.*

9.1. **Input Binding.** We assume $\mathcal{G}_{\mathsf{SNARK}}$ provides completeness, knowledge-soundness, and zero-knowledge with a black-box simulator $\mathsf{Sim}_{\mathcal{G}_{\mathsf{SNARK}}}$ that, for any verifier parameters vp output by $\mathcal{G}_{\mathsf{SNARK}}.\mathsf{Setup}$ and any statement x, outputs an accepting proof $\pi \leftarrow \mathsf{Sim}_{\mathcal{G}_{\mathsf{SNARK}}}(vp, x)$ whose distribution is indistinguishable from real proofs.

Lemma 3 (Input Binding)**.** *Let $Y = \{y_{i,b}\}_{i\in[n],\, b\in\{0,1\}}$ be Alice's commitments and let π_{bind} be a setup proof for $\mathcal{R}_{\mathsf{bind}}$ with $\mathcal{G}_{\mathsf{SNARK}}.\mathsf{Verify}(vp, x_{\mathsf{bind}}, \pi_{\mathsf{bind}}) = 1$. For any PPT adversary $\mathcal{A}$ that, with non-negligible probability, outputs an index $i^* \in [n]$ and two distinct openings (s'_0, s'_1) accepted by the GC input gate against Y, there exist PPT reductions $\mathcal{B}_{\mathsf{know}}, \mathcal{B}_{\mathsf{pre}}, \mathcal{B}_{\mathsf{2pre}}$ such that:*

$$\mathsf{Adv}^{\mathsf{bind}}_{\mathsf{Lamport/Setup}}(\mathcal{A}) \le \mathsf{Adv}^{\mathsf{know}}_{\mathcal{G}_{\mathsf{SNARK}}}(\mathcal{B}_{\mathsf{know}}) + 2n \cdot \Big(\mathsf{Adv}^{\mathsf{pre}}_{\mathcal{G}_{\mathsf{Hash}}}(\mathcal{B}_{\mathsf{pre}}) + \mathsf{Adv}^{\mathsf{2pre}}_{\mathcal{G}_{\mathsf{Hash}}}(\mathcal{B}_{\mathsf{2pre}})\Big) + \mathsf{negl}(\lambda).$$

Proof. We analyze $\mathcal{A}$ through a game and reductions:

Game and success event.

(1) Challenger runs $vp \leftarrow \mathcal{G}_{\mathsf{SNARK}}.\mathsf{Setup}(1^\lambda, \mathcal{R}_{\mathsf{bind}})$.

(2) For each (i, b), sample $s_{i,b} \xleftarrow{\$} \{0,1\}^{n_\lambda}$ and set $y_{i,b} \leftarrow \mathcal{G}_{\mathsf{Hash}}.\mathsf{Hash}(\texttt{FLEX:LAMPORT} \,\|\, s_{i,b})$ and $K_{i,b} \leftarrow \mathsf{HKDF}(s_{i,b}; \langle i, b\rangle, \texttt{FLEX:LABEL})$.

(3) Form $x_{\mathsf{bind}} := (Y, \ell, n_\ell)$ and compute $\pi_{\mathsf{bind}} \leftarrow \mathcal{G}_{\mathsf{SNARK}}.\mathsf{Prove}(vp, x_{\mathsf{bind}}, w_{\mathsf{bind}})$ with witness $w_{\mathsf{bind}} = \{s_{i,b}\}_{i,b}$.

(4) Give $(Y, \pi_{\mathsf{bind}}, vp)$ to $\mathcal{A}$; it outputs (i^*, s'_0, s'_1).

Let us define Succ to be the event that:

(1) $s'_0 \neq s'_1$ and

(2) $\left[\, \mathcal{G}_{\mathrm{Hash}}.\mathsf{Hash}(\texttt{FLEX:LAMPORT} \,\|\, s'_0) = y_{i^*,0} \wedge K_{i^*,0} = \mathsf{HKDF}(s'_0; \langle i^*, 0\rangle, \texttt{FLEX:LABEL}) \,\right]$, and

(3) $\left[\, \mathcal{G}_{\mathrm{Hash}}.\mathsf{Hash}(\texttt{FLEX:LAMPORT} \,\|\, s'_1) = y_{i^*,1} \wedge K_{i^*,1} = \mathsf{HKDF}(s'_1; \langle i^*, 1\rangle, \texttt{FLEX:LABEL}) \,\right]$

and let $\mathsf{Adv}^{\mathsf{bind}}_{\mathsf{Lamport/Setup}}(\mathcal{A}) := \Pr[\mathsf{Succ}]$.

Let E_{badSNARK} be the event:

$$\mathcal{G}_{\mathsf{SNARK}}.\mathsf{Verify}(vp, x_{\mathsf{bind}}, \pi_{\mathsf{bind}}) = 1 \;\text{ and }\; \nexists\, w :\; (x_{\mathsf{bind}}, w) \in \mathcal{R}_{\mathsf{bind}}.$$

By knowledge-soundness there exists $\mathcal{B}_{\mathsf{know}}$ such that $\Pr[E_{\mathsf{badSNARK}}] \leq \mathsf{Adv}^{\mathsf{know}}_{\mathcal{G}_{\mathsf{SNARK}}}(\mathcal{B}_{\mathsf{know}})$. Hence: $\Pr[\mathsf{Succ}] \leq \Pr[E_{\mathsf{badSNARK}}] + \Pr[\mathsf{Succ} \wedge \neg E_{\mathsf{badSNARK}}]$.

Then there exists witness $w_{\mathsf{bind}} = \{s_{i,b}\}$ with $\mathcal{G}_{\mathsf{Hash}}.\mathsf{Hash}(\texttt{FLEX:LAMPORT} \,\|\, s_{i,b}) = y_{i,b}$ and $K_{i,b} = \mathsf{HKDF}(s_{i,b}; \cdot)$ for all (i, b).

Under $\mathsf{Succ} \wedge \neg E_{\mathsf{badSNARK}}$ we have, for $b \in \{0, 1\}$,

$$\left[\, \mathcal{G}_{\mathrm{Hash}}.\mathsf{Hash}(\texttt{FLEX:LAMPORT} \,\|\, s'_b) = y_{i^*,b} \,\right] \text{ and } K_{i^*,b} = \mathsf{HKDF}(s'_b; \langle i^*, b\rangle, \texttt{FLEX:LABEL}).$$

Let $s_{i^*,b}$ be the honest preimages for $y_{i^*,b}$. Define

(1) E_{pre}: $\exists b \in \{0, 1\} :\; s'_b = s_{i^*,b}$ (the adversary outputs an honest preimage), and

(2) E_{2pre}: $\exists b \in \{0, 1\} :\; s'_b \neq s_{i^*,b} \wedge \left[\, \mathcal{G}_{\mathrm{Hash}}.\mathsf{Hash}(\texttt{FLEX:LAMPORT} \,\|\, s'_b) = y_{i^*,b} \,\right]$ (the adversary outputs a second preimage).

Note that E_{pre} and E_{2pre} are not disjoint (one s'_b can be an honest preimage while the other is a second preimage), but they *cover* $\mathsf{Succ} \wedge \neg E_{\mathsf{badSNARK}}$: for every wire $b \in \{0, 1\}$, acceptance of s'_b implies $\left[\, \mathcal{G}_{\mathrm{Hash}}.\mathsf{Hash}(\texttt{FLEX:LAMPORT} \,\|\, s'_b) = y_{i^*,b} \,\right]$, so s'_b is either the honest preimage $s_{i^*,b}$ or a second preimage of $y_{i^*,b}$. Hence

$$\Pr[\mathsf{Succ} \wedge \neg E_{\mathsf{badSNARK}}] \leq \Pr[E_{\mathsf{pre}} \wedge \mathsf{Succ} \wedge \neg E_{\mathsf{badSNARK}}] + \Pr[E_{\mathsf{2pre}} \wedge \mathsf{Succ} \wedge \neg E_{\mathsf{badSNARK}}].$$

We now describe $\mathcal{B}_{\mathsf{know}}$, $\mathcal{B}_{\mathsf{pre}}$, $\mathcal{B}_{\mathsf{2pre}}$ and lower bound their success.

Algorithm 16 $\mathcal{B}_{\mathsf{know}}$ (knowledge-soundness reduction)

State: Access to $\mathcal{G}_{\mathsf{SNARK}}.\mathsf{Setup}, \mathcal{G}_{\mathsf{SNARK}}.\mathsf{Prove}, \mathcal{G}_{\mathsf{SNARK}}.\mathsf{Verify}$ for $\mathcal{R}_{\mathsf{bind}}$.

1: Receive vp from the knowledge-soundness challenger.
2: Honestly sample $\{s_{i,b}\}$, set $y_{i,b} \leftarrow \mathcal{G}_{\mathsf{Hash}}.\mathsf{Hash}(\texttt{FLEX:LAMPORT} \,\|\, s_{i,b})$ and $K_{i,b} \leftarrow \mathsf{HKDF}(s_{i,b}; \langle i, b\rangle, \texttt{FLEX:LABEL})$.
3: Form x_{bind} and compute $\pi_{\mathsf{bind}} \leftarrow \mathcal{G}_{\mathsf{SNARK}}.\mathsf{Prove}(vp, x_{\mathsf{bind}}, \{s_{i,b}\})$.
4: Run $\mathcal{A}(Y, \pi_{\mathsf{bind}}, vp)$ to obtain (i^*, s'_0, s'_1).
5: If $\mathcal{G}_{\mathsf{SNARK}}.\mathsf{Verify}(vp, x_{\mathsf{bind}}, \pi_{\mathsf{bind}}) = 1$ but no witness exists, **return** $(x_{\mathsf{bind}}, \pi_{\mathsf{bind}})$; else **return** $\perp$.

Algorithm 17 $\mathcal{B}_{\mathsf{pre}}$ (preimage reduction)

State: Target digest $y^\star = \mathcal{G}_{\mathsf{Hash}}.\mathsf{Hash}(\texttt{FLEX:LAMPORT} \,\|\, x^\star)$ for hidden $x^\star$.
Interfaces: Access to $\mathcal{G}_{\mathsf{SNARK}}.\mathsf{Setup}, \mathcal{G}_{\mathsf{SNARK}}.\mathsf{Verify}$ and a zero-knowledge simulator $\mathsf{Sim}_{\mathcal{G}_{\mathsf{SNARK}}}$.

Sample $i^\dagger \xleftarrow{\$} [n]$ and $b^\dagger \xleftarrow{\$} \{0,1\}$; get $vp \leftarrow \mathcal{G}_{\mathsf{SNARK}}.\mathsf{Setup}(1^\lambda, \mathcal{R}_{\mathsf{bind}})$.
for all $(i,b) \in [n] \times \{0,1\}$ **do**
  if $(i,b) = (i^\dagger, b^\dagger)$ **then** Set $y_{i,b} \leftarrow y^\star$.
  else Sample $s_{i,b}$ and set $y_{i,b} \leftarrow \mathcal{G}_{\mathsf{Hash}}.\mathsf{Hash}(\texttt{FLEX:LAMPORT} \,\|\, s_{i,b})$.
  end if
end for
$x_{\mathsf{bind}} \leftarrow (Y, \ell, n_\ell)$; obtain $\pi_{\mathsf{bind}} \leftarrow \mathsf{Sim}_{\mathcal{G}_{\mathsf{SNARK}}}(vp, x_{\mathsf{bind}})$.
Run $\mathcal{A}(Y, \pi_{\mathsf{bind}}, vp)$; get (i', s'_0, s'_1).
if $i' = i^\dagger$ and $\big[\ \mathcal{G}_{\mathrm{Hash}}.\mathsf{Hash}(\texttt{FLEX:LAMPORT} \,\|\, s'_0) = y_{i',0}\ \big]$ and $\big[\ \mathcal{G}_{\mathrm{Hash}}.\mathsf{Hash}(\texttt{FLEX:LAMPORT} \,\|\, s'_1) = y_{i',1}\ \big]$ **then**
  if $\mathcal{G}_{\mathsf{Hash}}.\mathsf{Hash}(\texttt{FLEX:LAMPORT} \,\|\, s'_{b^\dagger}) = y^\star$ **then return** $s'_{b^\dagger}$
  end if
end if
return $\perp$

Algorithm 18 $\mathcal{B}_{\mathsf{2pre}}$ (second-preimage reduction)

State: Challenge $x^\star$ with $y^\star = \mathcal{G}_{\mathsf{Hash}}.\mathsf{Hash}(\texttt{FLEX:LAMPORT} \,\|\, x^\star)$.
Interfaces: Access to $\mathcal{G}_{\mathsf{SNARK}}.\mathsf{Setup}, \mathcal{G}_{\mathsf{SNARK}}.\mathsf{Prove}, \mathcal{G}_{\mathsf{SNARK}}.\mathsf{Verify}$.

Sample $i^\dagger \xleftarrow{\$} [n]$ and $b^\dagger \xleftarrow{\$} \{0,1\}$; get $vp \leftarrow \mathcal{G}_{\mathsf{SNARK}}.\mathsf{Setup}(1^\lambda, \mathcal{R}_{\mathsf{bind}})$.
for all (i,b) **do**
  if $(i,b) = (i^\dagger, b^\dagger)$ **then** Set $s_{i,b} \leftarrow x^\star$ and $y_{i,b} \leftarrow y^\star$.
  else Sample $s_{i,b}$ and set $y_{i,b} \leftarrow \mathcal{G}_{\mathsf{Hash}}.\mathsf{Hash}(\texttt{FLEX:LAMPORT} \,\|\, s_{i,b})$.
  end if
end for
$Y \leftarrow \{y_{i,b}\}$, $w_{\mathsf{bind}} \leftarrow \{s_{i,b}\}$, $x_{\mathsf{bind}} \leftarrow (Y, \ell, n_\ell)$, $\pi_{\mathsf{bind}} \leftarrow \mathcal{G}_{\mathsf{SNARK}}.\mathsf{Prove}(vp, x_{\mathsf{bind}}, w_{\mathsf{bind}})$.
Run $\mathcal{A}(Y, \pi_{\mathsf{bind}}, vp)$; get (i', s'_0, s'_1).
if $i' = i^\dagger$ and $\big[\ \mathcal{G}_{\mathrm{Hash}}.\mathsf{Hash}(\texttt{FLEX:LAMPORT} \,\|\, s'_0) = y_{i',0}\ \big]$ and $\big[\ \mathcal{G}_{\mathrm{Hash}}.\mathsf{Hash}(\texttt{FLEX:LAMPORT} \,\|\, s'_1) = y_{i',1}\ \big]$ **then**
  if $s'_{b^\dagger} \neq x^\star$ and $\mathcal{G}_{\mathsf{Hash}}.\mathsf{Hash}(\texttt{FLEX:LAMPORT} \,\|\, s'_{b^\dagger}) = y^\star$ **then return** $(x^\star, s'_{b^\dagger})$
  end if
end if
return $\perp$

Let E_{guess} be the event $i' = i^\dagger$; $\Pr[E_{\mathsf{guess}}] = 1/n$. Let E_{bit} be the event that the differing/target bit equals $b^\dagger$; $\Pr[E_{\mathsf{bit}} \mid E_{\mathsf{guess}}] = 1/2$. Since Succ implies both

$$\big[\ \mathcal{G}_{\mathrm{Hash}}.\mathsf{Hash}(\texttt{FLEX:LAMPORT} \,\|\, s'_0) = y_{i^*,0}\ \big] \text{ and } \big[\ \mathcal{G}_{\mathrm{Hash}}.\mathsf{Hash}(\texttt{FLEX:LAMPORT} \,\|\, s'_1) = y_{i^*,1}\ \big],$$

$$\mathsf{Adv}^{\mathsf{pre}}_{\mathcal{G}_{\mathsf{Hash}}}(\mathcal{B}_{\mathsf{pre}}) \;\geq\; \Pr[\mathsf{Succ} \wedge \neg E_{\mathsf{badSNARK}} \wedge E_{\mathsf{pre}}] \cdot \frac{1}{n} \cdot \frac{1}{2},$$

$$\mathsf{Adv}^{\mathsf{2pre}}_{\mathcal{G}_{\mathsf{Hash}}}(\mathcal{B}_{\mathsf{2pre}}) \;\geq\; \Pr[\mathsf{Succ} \wedge \neg E_{\mathsf{badSNARK}} \wedge E_{\mathsf{2pre}}] \cdot \frac{1}{n} \cdot \frac{1}{2}.$$

Adding the two and using the union-bound cover established above,

$$\Pr[\mathsf{Succ} \wedge \neg E_{\mathsf{badSNARK}}] \;\leq\; 2n \cdot \Big(\mathsf{Adv}^{\mathsf{pre}}_{\mathcal{G}_{\mathsf{Hash}}}(\mathcal{B}_{\mathsf{pre}}) + \mathsf{Adv}^{\mathsf{2pre}}_{\mathcal{G}_{\mathsf{Hash}}}(\mathcal{B}_{\mathsf{2pre}})\Big).$$

Therefore, we have:

$$\mathsf{Adv}^{\mathsf{bind}}_{\mathsf{Lamport/Setup}} \leq \mathsf{Adv}^{\mathsf{know}}_{\mathcal{G}_{\mathsf{SNARK}}} + 2n \cdot \left(\mathsf{Adv}^{\mathsf{pre}}_{\mathcal{G}_{\mathsf{Hash}}} + \mathsf{Adv}^{\mathsf{2pre}}_{\mathcal{G}_{\mathsf{Hash}}}\right) + \mathsf{negl}(\lambda).$$

□

9.2. **CDS secrecy and authenticity.**

Lemma 4 (CDS authenticity)**.** *If the garbling scheme satisfies standard authenticity and input binding holds, then Definition 6 holds.*

Proof sketch. With one label per wire, the evaluator's input (x, y) is fixed, so the decoded output is uniquely $C_{f,P_a,P_b}(x, y) \in \{P_a, P_b\}$. Producing both P_a and P_b would require either (a) evaluating on two different inputs by opening two labels on at least one wire (violates input binding), or (b) forging labels/alternate decodings that pass gate checks (violates GC authenticity). Either event has negligible probability; hence the conjunction does as well. □

Theorem 1 (CDS Secrecy)**.** *Fix a predicate f, secrets (P_a, P_b), and the CDS circuit C_{f,P_a,P_b} that outputs P_a if $f(x, y) = 1$ and P_b otherwise. Let us assume that the garbling scheme is authentic and private with advantages $\mathsf{Adv}^{\mathsf{auth}}_{\mathsf{CDS}}$ and $\mathsf{Adv}^{\mathsf{priv}}_{\mathsf{GC}}$, respectively; and that the label derivation HKDF is a PRF with advantage $\mathsf{Adv}^{\mathit{prf}}_{\mathsf{HKDF}}$. Then for every PPT evaluator $\mathcal{A}$ interacting with a single garbling $\widetilde{C}$ of C_{f,P_a,P_b} and receiving at most one input label per wire:*

(1) Exclusivity: The probability that $\mathcal{A}$ outputs both P_a and P_b is negligible:

$$\Pr\left[\mathcal{A} \textit{ outputs both } P_a \textit{ and } P_b\right] \leq \mathsf{Adv}^{\mathsf{auth}}_{\mathsf{CDS}} + \mathsf{Adv}^{\mathsf{bind}}_{\mathsf{Lamport/Setup}} + \mathsf{negl}(\lambda).$$

(2) Leakage-bounded privacy: Let L be the leakage and let $s \in \{P_a, P_b\}$ be the released secret determined by $f(x, y)$. There exists a PPT simulator Sim such that for every PPT distinguisher $\mathcal{D}$:

$$\begin{aligned} &\left|\Pr[\mathcal{D}(\mathsf{View}_{\mathsf{Real}}) = 1] - \Pr[\mathcal{D}(\mathsf{Sim}(L, s)) = 1]\right| \\ \leq\ & n \cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF}} + \mathsf{Adv}^{\mathsf{priv}}_{\mathsf{GC}} + \mathsf{Adv}^{\mathsf{bind}}_{\mathsf{Lamport/Setup}} + \mathsf{negl}(\lambda). \end{aligned}$$

Proof. Concerning exclusivity, let Both be the event $\mathcal{A}$ *outputs both P_a and P_b* and let $\mathsf{GoodBind}$ be the event that the evaluator has at most one accepted label per wire during evaluation. By Lemma 3 we have that $\Pr[\neg\mathsf{GoodBind}] \leq \mathsf{Adv}^{\mathsf{bind}}_{\mathsf{Lamport/Setup}}$. Then

$$\begin{aligned} \Pr[\mathsf{Both}] &\leq \Pr[\neg\mathsf{GoodBind}] + \Pr[\mathsf{Both} \wedge \mathsf{GoodBind}] \leq \\ &\leq \mathsf{Adv}^{\mathsf{bind}}_{\mathsf{Lamport/Setup}} \ +\ \Pr[\mathsf{Both} \wedge \mathsf{GoodBind}]. \end{aligned}$$

On the other hand, Lemma 4 sets $\Pr[\mathsf{Both} \wedge \mathsf{GoodBind}] \leq \mathsf{Adv}^{\mathsf{auth}}_{\mathsf{CDS}}$, establishing the stated bound. For completeness, we present the authenticity reduction, with the challenger providing the garbling:

Algorithm 19 $\mathcal{B}^{\mathcal{A}}_{\mathsf{know}}$ - Reduction to GC/CDS authenticity

1: **Input from challenger:** A CDS authenticity challenge consisting of a garbling $\widetilde{C}$ for C_{f,P_a,P_b} and, for each input wire i, a single valid label K_{i,b_i} (one-label-per-wire).
2: Give $(\widetilde{C}, \{K_{i,b_i}\}_i)$ to $\mathcal{A}$.
3: **if** $\mathcal{A}$ outputs both P_a and P_b **then return** break.
4: **else return** $\perp$.
5: **end if**

We observe that if $\mathcal{A}$ makes $\mathcal{B}^{\mathcal{A}}_{\mathsf{know}}$ win under one-label-per-wire access, then CDS authenticity is violated, and $\Pr[\mathsf{Both} \wedge \mathsf{GoodBind}] \leq \mathsf{Adv}^{\mathsf{auth}}_{\mathsf{CDS}}$.

Concerning privacy, let us write the evaluator's real view as $\mathsf{View}_{\mathsf{Real}} = (\widetilde{C}, L_x, L_y, o) \parallel \mathsf{meta}$, where $o \in \{P_a, P_b\}$ is the decoded secret and meta is public metadata captured by Leak. We proceed by hybrids and account explicitly for the $\mathsf{GoodBind}$ conditioning.

(1) Hybrid $H_0 \to H_1$ (HKDF labels $\to$ uniform): Replace Alice's input labels $L_x = \{K_{i,x_i}\}$, for $K_{i,x_i} = \mathsf{HKDF}(s_{i,x_i}; \langle i, x_i \rangle, \texttt{FLEX:LABEL})$, with independent uniform strings of the same length. By the PRF/KDF property of HKDF:

$$\big|\Pr[\mathcal{D}(\mathsf{View}_{H_0}) = 1] - \Pr[\mathcal{D}(\mathsf{View}_{H_1}) = 1]\big| \;\leq\; \sum_{j=1}^{n} \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF}}(\mathcal{B}_{\mathsf{PRF},j}) \leq n \cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF}}.$$

Algorithm 20 $\mathcal{B}_{\mathsf{PRF},j}$ – Wire-j step of the CDS-privacy hybrid

State: A PRF oracle $\mathcal{O}_j$ as in Algorithm 25.
1: Sample input $x \in \{0,1\}^n$ as in the real run.
2: **for** $i = 1, \ldots, j-1$ **do** $K_i^\star \xleftarrow{\$} \{0,1\}^{n_\ell}$
3: **end for**
4: $K_j \leftarrow \mathcal{O}_j(\langle j, x_j, \mathsf{ctx} \rangle, \texttt{FLEX:LABEL})$
5: **for** $i = j+1, \ldots, n$ **do** $s_{i,x_i} \xleftarrow{\$} \{0,1\}^{n_\lambda}$; $K_i \leftarrow \mathsf{HKDF}(s_{i,x_i}; \langle i, x_i, \mathsf{ctx} \rangle, \texttt{FLEX:LABEL})$
6: **end for**
7: Build the view $(\widetilde{C}, \{K_i\}_i, L_y, o) \| \mathsf{meta}$ as in the real run and feed it to $\mathcal{D}$.
8: **return** $\mathcal{D}$'s bit.

Therefore: $|\Pr[\mathcal{D}(H_0) = 1] - \Pr[\mathcal{D}(H_1) = 1]| \leq n \cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF}}$.

(2) Hybrid $H_1 \to H_2$ (real garbling $\to$ simulated garbling): Invoke the (standard) GC privacy simulator on the CDS circuit with the output fixed to the released secret o. Let $(\widetilde{C}^\star, L_x^\star, L_y^\star)$ be the simulator's output; form $\mathsf{View}_{H_2} = (\widetilde{C}^\star, L_x^\star, L_y^\star, o) \| \mathsf{meta}$. Under one-label-per-wire (our $\mathsf{GoodBind}$ event), GC privacy gives

$$\big|\Pr[D(\mathsf{View}_{H_1}) = 1] - \Pr[D(\mathsf{View}_{H_2}) = 1]\big| \;\leq\; \mathsf{Adv}^{\mathsf{priv}}_{\mathsf{GC}}(\mathcal{B}_{\mathsf{GC}}).$$

Algorithm 21 $\mathcal{B}_{\mathsf{GC}}$ - Reduction to GC privacy

1: **Input from challenger:** Either a real triple $\left(\widetilde{C}^{(1)}, L_x^{(1)}, L_y^{(1)}\right)$ for C_{f,P_a,P_b} or a simulated triple $\left(\widetilde{C}^{(0)}, L_x^{(0)}, L_y^{(0)}\right)$ with the same arities and output o.
2: For $b \in \{0,1\}$ form $\mathsf{View}^{(b)} = \left(\widetilde{C}^{(b)}, L_x^{(b)}, L_y^{(b)}, o\right) \| \mathsf{meta}$ and feed to $\mathcal{D}$.
3: **Return** $\mathcal{D}$'s bit.

For any $\mathcal{D}$,

$$\big|\Pr[\mathcal{D}(\mathsf{View}_{\mathsf{Real}})=1]-\Pr[\mathcal{D}(\mathsf{View}_{H_2})=1]\big| \le \big|\Pr[\mathcal{D}(\mathsf{View}_{H_0})=1]-\Pr[\mathcal{D}(\mathsf{View}_{H_1})=1]\big| + \big|\Pr[\mathcal{D}(\mathsf{View}_{H_1})=1]-\Pr[\mathcal{D}(\mathsf{View}_{H_2})=1]\big| \le \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF}}(\mathcal{B}_{\mathsf{PRF}})+\mathsf{Adv}^{\mathsf{priv}}_{\mathsf{GC}}(\mathcal{B}_{\mathsf{GC}}).$$

Now decompose on $\mathsf{GoodBind}$:

$$\begin{aligned}&\big|\Pr[\mathcal{D}(\mathsf{View}_{\mathsf{Real}})=1]-\Pr[\mathcal{D}(\mathsf{View}_{H_2})=1]\big|\\ &\le \big|\Pr[\mathcal{D}(\mathsf{View}_{\mathsf{Real}})=1 \mid \mathsf{GoodBind}]-\Pr[\mathcal{D}(\mathsf{View}_{H_2})=1]\big|\cdot\Pr[\mathsf{GoodBind}]\\ &\quad+\Pr[\neg\mathsf{GoodBind}] \le \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF}}(\mathcal{B}_{\mathsf{PRF}})+\mathsf{Adv}^{\mathsf{priv}}_{\mathsf{GC}}(\mathcal{B}_{\mathsf{GC}})+\mathsf{Adv}^{\mathsf{bind}}_{\mathsf{Lamport/Setup}}+\mathsf{negl}(\lambda).\end{aligned}$$

Finally, define the ideal view through the explicit simulator described below:

Algorithm 22 $\mathsf{Sim}_{\mathsf{CDS}}(\mathsf{Leak}, o)$ - Ideal-view simulator

1: Parse Leak to learn arities $|x|, |y|$ and public metadata meta.
2: Deduce the winner bit b from o using the CDS semantics: if $o = P_a$ then $b := 1$; if $o = P_b$ then $b := 0$.
3: Invoke $(\widetilde{C}^\star, L_x^\star, L_y^\star) \leftarrow \mathsf{Sim}_{\mathsf{GC}}(1^\lambda; C_{f,P_a,P_b}, |x|, |y|, o)$.
4: Output $\mathsf{View}_{\mathsf{Ideal}} = (\widetilde{C}^\star, L_x^\star, L_y^\star, o)\|\mathsf{meta}$.

Combining the hybrid bounds with the conditioning step yields the stated leakage-bounded simulation guarantee. This concludes the proof. □

9.3. **Soundness theorem.** For the following result we assume that the Bitcoin consensus enforces Taproot script-path validation and CSV, that the Taproot key path is unspendable, and that the CDS garbling satisfies input binding (Lemma 3) and CDS authenticity (Lemma 4).

Fix a protocol $\Pi \in \{\mathsf{FLEX}, \mathsf{FLEX2}\}$ with winning set $\Omega^{\Pi}_{\mathsf{win}}$ and decoder De^{Π} as defined in Alg. 12 and Alg. 13. Let $\mathsf{WinnerSpend}$ denote the event where some $\mathsf{tx} \in \Omega^{\Pi}_{\mathsf{win}}$ is k-finalized on-chain. Let $\mathsf{De}^{\Pi}(\mathsf{tx}) \in \{0,1\}$ denote the winner bit certified by tx.

Theorem 2 (Soundness)**.** *Fix x and y, let $b^\star := f(x,y) \in \{0,1\}$ be the correct winner bit, and $\mathsf{Wrong} := \big[\mathsf{WinnerSpend} \wedge \mathsf{De}^{\Pi}(\mathsf{tx}) \neq b^\star\big]$. Then for any PPT evaluator adversary $\mathcal{A}$,* $\Pr[\mathsf{Wrong}] \le 2\mathsf{Adv}^{\mathsf{euf}}_{\mathcal{G}_{\mathsf{Sig}}}\big(\mathcal{B}^{(\pi)}_{\mathsf{sig}}\big)+\mathsf{Adv}^{\mathsf{pre}}_{\mathcal{G}_{\mathsf{Hash}}}(\mathcal{B}_{\mathsf{pre}})+\mathsf{Adv}^{\mathsf{auth}}_{\mathsf{CDS}}+\mathsf{Adv}^{\mathsf{bind}}_{\mathsf{Lamport/Setup}}+\mathsf{negl}(\lambda)$.

The factor 2 comes from a uniform side guess over the two honest-party signing keys; the reduction $\mathcal{B}^{(\pi)}_{\mathsf{sig}}$ is given in Algorithm 23. Here $\mathcal{G}_{\mathsf{Sig}}$ is Schnorr, $\mathcal{G}_{\mathsf{Hash}}$ is the hash used in pot hashlock leaves, and the explicit black-box reductions $\mathcal{B}^{(\pi)}_{\mathsf{sig}}, \mathcal{B}_{\mathsf{pre}}$ are given below.

Proof. By protocol design, all winner outcomes are determined on-chain by a k-finalized transaction in $\Omega^{\Pi}_{\mathsf{win}}$. Hence it suffices to analyze how a k-finalized winner-determining tx with $\mathsf{De}(\mathsf{tx}) \neq b^\star$ can arise.

In a Taproot script-path spend of a hashlock leaf, the witness stack contains a control block identifying the leaf (so we learn whether the V_a or V_b leaf was used), a Schnorr signature (R, s) under the leaf key X on the sighash message m, and a preimage p with $\mathcal{G}_{\mathsf{Hash}}.\mathsf{Hash}(p) = V \in \{V_a, V_b\}$.

Case A: tx is a pre-CSV hashlock-leaf spend. Let $(X, (R,s), m)$ and p be parsed as above.

(1) Schnorr forgery: We reduce to Schnorr EUF-CMA, with key separation between pot leaf keys and all transaction-signing keys.

Algorithm 23 $\mathcal{B}_{\mathsf{sig}}^{(\pi)}$ – Reduction to Schnorr EUF-CMA (leaf-key forgery)

State: Oracle access to $\mathsf{Sign}^{\star}(\cdot)$ producing Schnorr signatures under the EUF-CMA challenge key $X^{\star}$.

Sample side guess $\pi \xleftarrow{\$} \{A, B\}$ and let $\bar{\pi}$ denote the other party.
Set $\mathsf{pk}_{\pi} := X^{\star}$ (secret held by the EUF-CMA challenger).
Sample $(\mathsf{sk}_{\bar{\pi}}, \mathsf{pk}_{\bar{\pi}}) \leftarrow \mathcal{G}_{\mathsf{Sig}}.\mathsf{KeyGen}(\mathsf{Schnorr})$ honestly.
Construct $\mathsf{pot}_A, \mathsf{pot}_B$ with the honest leaf-key assignment of Section 5.1, using $\mathsf{pk}_A, \mathsf{pk}_B$ as set above. Hashlock commitments and CSV parameters are unchanged.
Initialise the queried-message log $Q \leftarrow \emptyset$.
procedure SimulateProtocol($\mathcal{A}$)
    Run the protocol with $\mathcal{A}$ in the role of the corrupted party.
    for all honest signatures produced during execution **do**
        Let m be the sighash (via $\mathcal{G}_{\mathsf{Sig}}$.Serialize of Alg. 2).
        if the signature is under pk_{π} **then** $\sigma \leftarrow \mathsf{Sign}^{\star}(m)$; $Q \leftarrow Q \cup \{m\}$
        else $\sigma \leftarrow \mathcal{G}_{\mathsf{Sig}}.\mathsf{Sign}(\mathsf{sk}_{\bar{\pi}}, m)$
        end if
        Attach σ to the corresponding transaction.
    end for
end procedure

procedure Extract(Wrong event)
    if Wrong finalises through a hashlock-leaf spend **then**
        parse witness as $(p, \sigma, \mathsf{ctrlblk})$ identifying leaf ℓ, $\sigma = (R, s)$, sighash m
        if ℓ's designated key is $X^{\star}$ **and** $m \notin Q$ **then**
            return $(m, (R, s))$ to the EUF-CMA challenger
        end if
    end if
    return $\perp$
end procedure

The reduction $\mathcal{B}_{\mathsf{sig}}^{(\pi)}$ returns $(m, (R, s))$ only when $m \notin Q$, so the output is an EUF-CMA forgery. The side guess π is sampled uniformly and is information-theoretically hidden from $\mathcal{A}$ (all leaves are correctly distributed regardless of π), so conditioned on Wrong being realised through a hashlock-leaf spend whose leaf-key is the honest party's key, $\Pr[\pi \text{ matches}] = 1/2$. The residual probability that the extracted m collides with some queried message in Q despite the underlying transactions differing is bounded by the second-preimage advantage of TapSighash (Alg. 2) and is absorbed in $\mathsf{negl}(\lambda)$. Hence

$$\Pr[\mathsf{Wrong} \text{ through (A1)}] \;\le\; 2 \cdot \mathsf{Adv}^{\mathsf{euf}}_{\mathcal{G}_{\mathsf{Sig}}}\big(\mathcal{B}_{\mathsf{sig}}^{(\pi)}\big) + \mathsf{negl}(\lambda).$$

(2) Losing-side preimage present in the witness. Write $V_{\mathsf{lose}} \in \{V_a, V_b\}$ for the losing-side commitment given $b^{\star}$. There are two ways such a preimage can appear:

    (a) Illegitimate learning through CDS: With a single-label-per-wire view, the evaluator can decode only the correct-side secret; learning the losing-side secret contradicts either CDS authenticity or Input Binding. Hence

$$\Pr[2a] \;\le\; \mathsf{Adv}^{\mathsf{auth}}_{\mathrm{CDS}} \;+\; \mathsf{Adv}^{\mathsf{bind}}_{\mathrm{Lamport/Setup}} \;+\; \mathsf{negl}(\lambda).$$

(b) Direct preimage inversion: We reduce to hash preimage. To avoid any 1/2 guessing loss, we embed the same challenge digest into both leaves.

Algorithm 24 $\mathcal{B}_{\mathsf{pre}}$ - Reduction to hash preimage (leaf preimage)

1: **Given:** Preimage challenge digest $y^\star$ for $\mathcal{G}_{\mathsf{Hash}}$.
2: **Embed:** Set both leaf commitments to $V_a := V_b := y^\star$ when constructing the pots. (This is protocol-legitimate: the protocol permits arbitrary commitments; P_a, P_b remain hidden as preimages of $y^\star$.)
3: **Run $\mathcal{A}$:** Execute the protocol honestly using locally generated transaction keys.
4: **On wrong spend:** If Wrong occurs through a hashlock leaf and the witness contains a preimage p with $\mathcal{G}_{\mathsf{Hash}}.\mathsf{Hash}(p) = y^\star$, **return** p.

$$\text{Therefore: } \Pr[2b] \;\le\; \mathsf{Adv}^{\mathsf{pre}}_{\mathcal{G}_{\mathsf{Hash}}}(\mathcal{B}_{\mathsf{pre}}) + \mathsf{negl}(\lambda).$$

Case B: tx is a post-CSV spend through a refund leaf. For $\Pi = \mathsf{FLEX}$ this is $\mathsf{tx} = \mathsf{Reimburse}$, which spends pot_A through ℓ^A_{Refund} after CSV τ_A. For $\Pi = \mathsf{FLEX2}$ it additionally includes $\mathsf{tx} = \mathsf{EarlyRefund}$, which spends pot_A through ℓ^A_{Refund} after CSV τ_A *and* consumes the synchronisation UTXO S created by TryEarlyRefund (Section 7.6). Both refund leaves require a Schnorr signature under pk_A and CSV satisfaction; neither requires a hash preimage.

Case B is subsumed by (2a). A post-CSV refund-leaf spend of pot_A finalises with $\mathsf{De}^\Pi(\mathsf{tx}) \neq b^\star = 0$ only if Bob fails to obtain P_b off-chain — otherwise honest Bob would have broadcast either BobTakesDeposit (consuming pot_A through $\ell^A_{\mathsf{BobWins}}$ before τ_A) or AliceLost (blocking Reimburse), and either spend is incompatible with a subsequent finalisation of Reimburse or EarlyRefund (the latter by Lemma 1 Part (1)). But Bob fails to obtain P_b when $f(x, y) = 0$ only if either the CDS-mode garbling evaluates to a value other than P_b (violating CDS correctness, Definition 6 (1), which holds except with probability $\mathsf{negl}(\lambda)$), or the input labels Bob receives are inconsistent with the binding commitments (which by Lemma 3 reduces to $\mathsf{Adv}^{\mathsf{bind}}_{\mathsf{Lamport/Setup}}$). Both branches are already counted in (2a). Therefore {Case B wrong outcomes} $\subseteq$ (2a) up to the absorbed $\mathsf{negl}(\lambda)$ slack.

Union bound and conclusion. The analysis yields exactly three mutually exclusive cryptographic mechanisms by which Wrong can occur:

(1) Schnorr forgery under the leaf key (1): $\Pr[\mathsf{Wrong} \text{ through } 1] \le 2 \cdot \mathsf{Adv}^{\mathsf{euf}}_{\mathcal{G}_{\mathsf{Sig}}}\big(\mathcal{B}^{(\pi)}_{\mathsf{sig}}\big)$;

(2) CDS authenticity (2a): $\Pr[2a] \le \mathsf{Adv}^{\mathsf{auth}}_{\mathrm{CDS}} + \mathsf{Adv}^{\mathsf{bind}}_{\textsc{Lamport/Setup}}$;

(3) Direct hash preimage inversion (2b): $\Pr[2b] \le \mathsf{Adv}^{\mathsf{pre}}_{\mathcal{G}_{\mathsf{Hash}}}(\mathcal{B}_{\mathsf{pre}})$.

By the argument above, Case B is subsumed in (2a). A union bound over (1), (2a), (2b) therefore gives:

$$\Pr[\mathsf{Wrong}] \le 2\mathsf{Adv}^{\mathsf{euf}}_{\mathcal{G}_{\mathsf{Sig}}}\big(\mathcal{B}^{(\pi)}_{\mathsf{sig}}\big) + \mathsf{Adv}^{\mathsf{pre}}_{\mathcal{G}_{\mathsf{Hash}}}(\mathcal{B}_{\mathsf{pre}}) + \mathsf{Adv}^{\mathsf{auth}}_{\mathsf{CDS}} + \mathsf{Adv}^{\mathsf{bind}}_{\mathsf{Lamport/Setup}} + \mathsf{negl}(\lambda).$$

which proves the claim. □

9.4. **Privacy theorem.**

Theorem 3 (Privacy)**.** *Let π_{FLEX} and π_{FLEX2} be the real protocols and* FLEX, FLEX2 *their ideal functionalities. Assume that* HKDF *is a PRF, the garbling scheme has privacy against an evaluator with one label per input wire, and $\mathcal{G}_{\mathsf{SNARK}}$ is zero-knowledge. Then for any PPT adversary $\mathcal{A}$:*

(1) ***Standalone privacy***: *There exists a PPT simulator* $\mathsf{Sim}_{\mathsf{st}}$ *such that for every PPT distinguisher* $\mathcal{D}$*,*

$$\left|\Pr[\mathcal{D}(\mathsf{View}_{\mathsf{Real}}) = 1] - \Pr[\mathcal{D}(\mathsf{Sim}_{\mathsf{st}}(\mathsf{Leak}(\mathsf{View}_{\mathsf{Real}}), o)) = 1]\right| \leq \epsilon_{\mathsf{priv}},$$

where $o \in \{P_a, P_b\}$ *is the released secret.*

(2) ***UC-compatible privacy***: *There exists a PPT simulator* Sim *that, receiving only* $(\mathsf{Leak}(\mathsf{View}_{\mathcal{F}}), o)$ *from the ideal functionality* $\mathcal{F} \in \{\mathsf{FLEX}, \mathsf{FLEX2}\}$*, produces a view satisfying* $\left|\Pr[\mathcal{D}(\mathsf{View}_{\mathsf{Real}}) = 1] - \Pr[\mathcal{D}(\mathsf{Sim}^{\mathcal{F}}(\cdot)) = 1]\right| \leq \epsilon_{\mathsf{priv}}$*, where* $\mathsf{Sim}^{\mathcal{F}}$ *denotes* Sim *interacting with* $\mathcal{F}$*.*

In both cases, the advantage bound is:

$$\varepsilon_{\mathsf{priv}} \;:=\; n \cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF}} + \mathsf{Adv}^{\mathsf{priv}}_{\mathsf{GC}} + \mathsf{Adv}^{\mathsf{zk}}_{\mathcal{G}_{\mathsf{SNARK}}} + \mathsf{Adv}^{\mathsf{bind}}_{\mathsf{Lamport/Setup}} + \mathsf{negl}(\lambda),$$

where the factor n *on* $\mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF}}$ *comes from the wire-by-wire hybrid in the* $H_0 \to H_1$ *step of the proof.*

Proof. We first prove Part 1, then derive UC-compatible privacy via Lemma 2.

Part 1: Standalone privacy. We prove indistinguishability through hybrids, conditioning throughout on the binding event $\mathsf{GoodBind}$ (that the evaluator receives at most one input label per wire). By Lemma 3, $\Pr[\neg\mathsf{GoodBind}] \leq \mathsf{Adv}^{\mathsf{bind}}_{\mathsf{Lamport/Setup}}$. Let H_0 be the real execution.

(1) Hybrid $H_0 \to H_1$ (HKDF labels $\Rightarrow$ uniform): Under $\mathsf{GoodBind}$ the evaluator receives exactly one label $K_{i,x_i} = \mathsf{HKDF}(s_{i,x_i}; \langle i, x_i, \mathsf{ctx}\rangle, \texttt{FLEX:LABEL})$ per input wire $i \in [n]$. Each label is derived from a wire-specific seed s_{i,x_i}, and the n seeds are mutually independent; we therefore cannot in a single step apply single-key PRF security to switch all n labels to uniform. We use a standard wire-by-wire hybrid.

Introduce $n+1$ intermediate hybrids $H_0 = H_0^{(0)}, H_0^{(1)}, \ldots, H_0^{(n)} = H_1$ where

(a) $H_0^{(j)}$ delivers uniform labels $K_i^\star \xleftarrow{\$} \{0,1\}^{n_\ell}$ for $i \leq j$;

(b) $H_0^{(j)}$ delivers $K_i = \mathsf{HKDF}(s_{i,x_i}; \langle i, x_i, \mathsf{ctx}\rangle, \texttt{FLEX:LABEL})$ as in the real run for $i > j$.

The reduction $\mathcal{B}_{\mathsf{PRF},j}$ (Algorithm 25 below) embeds its PRF challenge in wire j's seed and samples honestly for the other $n-1$ wires; by single-key PRF security of HKDF, $\left|\Pr[\mathcal{D}(H_0^{(j-1)}) = 1] - \Pr[\mathcal{D}(H_0^{(j)}) = 1]\right| \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF}}(\mathcal{B}_{\mathsf{PRF},j})$. Summing over $j \in [n]$ yields:

$$\left|\Pr[\mathcal{D}(H_0) = 1] - \Pr[\mathcal{D}(H_1) = 1]\right| \leq \sum_{j=1}^{n} \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF}}(\mathcal{B}_{\mathsf{PRF},j}) \leq n \cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF}}.$$

Algorithm 25 $\mathcal{B}_{\mathsf{PRF},j}$ – Reduction for $H_0^{(j-1)} \Rightarrow H_0^{(j)}$

State: A PRF oracle $\mathcal{O}_j$ that is either $\mathsf{HKDF}(s^\star;\cdot)$ for a hidden seed $s^\star \xleftarrow{\$} \{0,1\}^{n_\lambda}$ or a uniformly random function $\{0,1\}^* \to \{0,1\}^{n_\ell}$.

Sample the honest sender's input $x = (x_1, \ldots, x_n) \in \{0,1\}^n$ as in the real run; fix the dispute context ctx as in the protocol.
for $i = 1, \ldots, j-1$ **do** $K_i^\star \xleftarrow{\$} \{0,1\}^{n_\ell}$ ▷ already-swapped wires
end for
$K_j \leftarrow \mathcal{O}_j(\langle j, x_j, \mathsf{ctx}\rangle, \texttt{FLEX:LABEL})$ ▷ embedded wire
for $i = j+1, \ldots, n$ **do**
    $s_{i,x_i} \xleftarrow{\$} \{0,1\}^{n_\lambda}$; $K_i \leftarrow \mathsf{HKDF}(s_{i,x_i}; \langle i, x_i, \mathsf{ctx}\rangle, \texttt{FLEX:LABEL})$ ▷ real wires
end for
for each $i \in [n]$ **do** sample $s_{i,1-x_i}$ honestly ▷ needed to build Y and $\widetilde{C}$
end for
Compute $Y = \{y_{i,b}\}$ as in the real run.
Construct the garbling $\widetilde{C}$ and SNARK proofs as in the real run.
Deliver $\{K_i\}_{i\in[n]}$ to $\mathcal{A}$ and complete the ledger interaction honestly.
return $\mathcal{D}$'s decision bit.

When $\mathcal{O}_j$ is the real PRF, K_j is distributed as in $H_0^{(j-1)}$. When $\mathcal{O}_j$ is a random function, K_j is uniform in $\{0,1\}^{n_\ell}$, matching $H_0^{(j)}$. All other components of the view are independent of $\mathcal{O}_j$'s choice. Hence $\mathcal{B}_{\mathsf{PRF},j}$'s advantage equals $\mathcal{D}$'s distinguishing advantage between $H_0^{(j-1)}$ and $H_0^{(j)}$.

(2) Hybrid $H_1 \to H_2$ (real garbling ⇒ simulated garbling): Replace $(\widetilde{C}, L_x, L_y)$ by

$$(\widetilde{C}^{\mathsf{sim}}, L_x^{\mathsf{sim}}, L_y^{\mathsf{sim}}) \leftarrow \mathsf{Sim}_{\mathsf{GC}}(1^\lambda, |x|, |y|, o)$$

given only Leak (sizes and the single output o). By GC privacy:

$$\left|\Pr[\mathcal{D}(H_1) = 1] - \Pr[\mathcal{D}(H_2) = 1]\right| \;\le\; \mathsf{Adv}^{\mathsf{priv}}_{\mathsf{GC}}(\mathcal{B}_{\mathsf{GC}}),$$

since GC privacy implies that for any inputs x, y and output $o = C(x, y)$, the tuple $(\widetilde{C}, L_x, L_y)$ is computationally indistinguishable from $\mathsf{Sim}_{\mathsf{GC}}(1^\lambda, |x|, |y|, o)$, even to a distinguisher that also sees one label per input wire and the circuit output. In H_1, labels for x are already uniform and independent of the garbling; H_2 replaces the garbling triple with $\mathsf{Sim}_{\mathsf{GC}}$'s output. The reduction $\mathcal{B}_{\mathsf{GC}}$ simply forwards the challenger's choice to $\mathcal{D}$, and the bound follows from the definition of $\mathsf{Adv}^{\mathsf{priv}}_{\mathsf{GC}}$.

Algorithm 26 $\mathcal{B}_{\mathsf{GC}}$ - Reduction for $H_1 \Rightarrow H_2$

Receive challenge $(\widetilde{C}^?, L_x^?, L_y^?)$ (real or simulated).
Use $(\widetilde{C}^?, L_x^?, L_y^?)$ in place of $(\widetilde{C}, L_x, L_y)$; keep the rest as in H_1.
Output $\mathcal{D}$'s bit.

(3) Hybrid $H_2 \to H_3$ (real SNARK proofs ⇒ simulated ZK proofs): For every public instance x, replace the real proof π by $\pi^{\mathsf{sim}} \leftarrow \mathsf{Sim}_{\mathsf{SNARK}}(vp, x)$. By zero-knowledge:

$$\left|\Pr[\mathcal{D}(H_2) = 1] - \Pr[\mathcal{D}(H_3) = 1]\right| \;\le\; \mathsf{Adv}^{\mathsf{zk}}_{\mathcal{G}_{\mathsf{SNARK}}}(\mathcal{B}_{\mathsf{ZK}}),$$

Zero-knowledge of $\mathcal{G}_{\mathsf{SNARK}}$ means that for any statement x in the language and any valid witness w, the distribution of $\pi \leftarrow \mathsf{Prove}(vp, x, w)$ is computationally indistinguishable from $\mathsf{Sim}_{\mathsf{SNARK}}(vp, x)$. In H_2, all other components (garbling,

labels) have already been simulated; H_3 replaces each real proof with a simulated one on the same x. Reduction $\mathcal{B}_{\mathsf{ZK}}$ forwards the challenger's choice for each proof occurrence, hence the advantage bound is exactly $\mathsf{Adv}^{\mathsf{zk}}_{\mathcal{G}_{\mathsf{SNARK}}}$.

Algorithm 27 $\mathcal{B}_{\mathsf{ZK}}$ - Reduction for $H_2 \Rightarrow H_3$

1: **for** each public instance x **do**
2: Output $\pi \leftarrow \mathsf{Sim}_{\mathsf{SNARK}}(vp, x)$ instead of a real proof.
3: **end for**
4: Run $\mathcal{D}$ on the resulting view and return its bit.

(4) Hybrid $H_3 \to H_4$ (make simulator control honest randomness): In H_4 we make explicit that the simulator chooses all honest randomness/keys with the same key-generation algorithms as in H_3. Hence $\Pr[\mathcal{D}(H_3) = 1] = \Pr[\mathcal{D}(H_4) = 1]$, indeed: we note that the only difference is expository since in H_4 we make explicit that all honest-party randomness (keys, nonces, timing choices) is sampled by the simulator using the same key-generation and sampling algorithms as in H_3. Since the distributions of all already-simulated objects (garblings, labels, SNARK proofs) are identical and the honest randomness is drawn from the same distributions, the joint transcript distributions in H_3 and H_4 are identical.

Explicit simulator Sim *and ledger simulation:* The view in H_4 is precisely the output of the following PPT simulator.

Algorithm 28 $\mathsf{Sim}_{\mathsf{st}}$ / $\mathsf{Sim}^{\mathcal{F}}$ - Privacy simulator

1: **Input:** Leakage L and released output $o \in \{P_a, P_b\}$.
2: (In standalone mode: $L = \mathsf{Leak}(\mathsf{View}_{\mathsf{Real}})$; in UC mode: $L = \mathsf{Leak}(\mathsf{View}_{\mathcal{F}})$ received from $\mathcal{F}$.)
3: **Parse** L to obtain public parameters, sizes ($|x|, |y|$), and metadata.
4: **Honest keys:** Sample all honest parties' transaction-signing keypairs via real key-gen.
5: **Simulate garbling:** $(\widetilde{C}, L_x, L_y) \leftarrow \mathsf{Sim}_{\mathsf{GC}}(1^\lambda, |x|, |y|, o)$.
6: **Simulate labels:** For each wire i, sample a single label $K_i \xleftarrow{\$} \{0,1\}^{n_\ell}$.
7: **Simulate SNARK proofs:** For each public instance x, set $\pi \leftarrow \mathsf{Sim}_{\mathsf{SNARK}}(vp, x)$.
8: **Ledger transcript:** Using simulator-controlled keys, construct and submit all honest transactions to $\mathcal{G}_{\mathsf{Ledger}}$; record confirmations and timing as delivered by $\mathcal{G}_{\mathsf{Ledger}}$.
9: **Output:** $\mathsf{View}_{\mathsf{Ideal}} = (\widetilde{C}, L_x, L_y, \{K_i\}, \{\pi\}, \text{ledger transcript}, \text{meta})$.

Since garblings, labels, and proofs are already simulated (hybrids H_1–H_3) and honest signatures/transactions are produced with the real algorithms against $\mathcal{G}_{\mathsf{Ledger}}$, the ledger transcript distribution in H_4 matches the real one.

CDS integration. If the variant includes CDS, Theorem 1 guarantees the adversary learns at most one of $\{P_a, P_b\}$ except with probability

$$\Pr[\text{both secrets}] \le \mathsf{Adv}^{\mathsf{auth}}_{\mathsf{CDS}} + \mathsf{Adv}^{\mathsf{bind}}_{\mathsf{Lamport/Setup}} + \mathsf{negl}(\lambda).$$

Since both the real and ideal views reveal exactly one secret, CDS gives no distinguishing advantage beyond the already-accounted $\mathsf{Adv}^{\mathsf{bind}}_{\mathsf{Lamport/Setup}}$ term (residual $\mathsf{negl}(\lambda)$).

Unconditioning on GoodBind. All hybrid bounds are proved conditioned on GoodBind. Unconditioning yields:

$$\begin{aligned}
&\big|\Pr[\mathcal{D}(\mathsf{View}_{\mathsf{Real}}) = 1] - \Pr[\mathcal{D}(\mathsf{View}_{\mathsf{Ideal}}) = 1]\big| \\
&= \big|\Pr[\mathcal{D}(\mathsf{View}_{\mathsf{Real}}) = 1 \mid \mathsf{GoodBind}]\Pr[\mathsf{GoodBind}] \\
&+ \Pr[\mathcal{D}(\mathsf{View}_{\mathsf{Real}}) = 1 \mid \neg\mathsf{GoodBind}]\Pr[\neg\mathsf{GoodBind}] - \Pr[\mathcal{D}(\mathsf{View}_{\mathsf{Ideal}}) = 1]\big| \\
&\le \big|\Pr[\mathcal{D}(\mathsf{View}_{\mathsf{Real}}) = 1 \mid \mathsf{GoodBind}] \\
&- \Pr[\mathcal{D}(\mathsf{View}_{\mathsf{Ideal}}) = 1]\big|\Pr[\mathsf{GoodBind}] + \Pr[\neg\mathsf{GoodBind}] \\
&\le \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF}} + \mathsf{Adv}^{\mathsf{priv}}_{\mathsf{GC}} + \mathsf{Adv}^{\mathsf{zk}}_{\mathcal{G}_{\mathsf{SNARK}}} + \mathsf{Adv}^{\mathsf{bind}}_{\mathsf{Lamport/Setup}} + \mathsf{negl}(\lambda),
\end{aligned}$$

since $\Pr[\mathsf{GoodBind}] \le 1$ and $\Pr[\neg\mathsf{GoodBind}] \le \mathsf{Adv}^{\mathsf{bind}}_{\mathsf{Lamport/Setup}}$. This establishes Part 1.

Part 2: UC-compatible privacy. We now show that the same simulator works when receiving leakage from $\mathcal{F}$ instead of from the real execution.

Define $\mathsf{Sim}^{\mathcal{F}}$ as follows: upon receiving $(\mathsf{Leak}(\mathsf{View}_{\mathcal{F}}), o)$ from $\mathcal{F}$, run the standalone simulator $\mathsf{Sim}_{\mathsf{st}}$ (Algorithm 28) with input $(\mathsf{Leak}(\mathsf{View}_{\mathcal{F}}), o)$.

By Lemma 2, $\mathsf{Leak}(\mathsf{View}_{\mathcal{F}}) \equiv \mathsf{Leak}(\mathsf{View}_{\mathsf{Real}})$. Therefore, for any distinguisher $\mathcal{D}$:

$$\begin{aligned}
\Pr[\mathcal{D}(\mathsf{Sim}^{\mathcal{F}}(\cdot)) = 1] &= \Pr[\mathcal{D}(\mathsf{Sim}_{\mathsf{st}}(\mathsf{Leak}(\mathsf{View}_{\mathcal{F}}), o)) = 1] \\
&= \Pr[\mathcal{D}(\mathsf{Sim}_{\mathsf{st}}(\mathsf{Leak}(\mathsf{View}_{\mathsf{Real}}), o)) = 1],
\end{aligned}$$

where the second equality uses the identical distribution of leakages. Combined with the standalone bound from Part 1: $\big|\Pr[\mathcal{D}(\mathsf{View}_{\mathsf{Real}}) = 1] - \Pr[\mathcal{D}(\mathsf{Sim}^{\mathcal{F}}(\cdot)) = 1]\big| \le \epsilon_{\mathsf{priv}}$. This establishes UC-compatible privacy, completing the proof. □

9.5. **UC security theorem.** This UC proof will not rely on a simulatable/programmed CRS. It relies on the PRF security of HKDF for label pseudorandomness; the privacy of the garbling scheme through $\mathsf{Sim}_{\mathsf{GC}}$; and the zero-knowledge property of $\mathcal{G}_{\mathsf{SNARK}}$ through $\mathsf{Sim}_{\mathsf{SNARK}}$. Combining all parts, for every PPT $\mathcal{A}$ there exists PPT Sim such that for every PPT $\mathcal{Z}$, the executions are computationally indistinguishable, which establishes the UC realization claims for both π_{FLEX}/FLEX and π_{FLEX2}/FLEX2.

Theorem 4 (UC Realization)**.** *In the* $(\mathcal{G}_{\mathsf{Ledger}}, \mathcal{G}_{\mathsf{Hash}}, \mathcal{G}_{\mathsf{Sig}}, \mathcal{G}_{\mathsf{SNARK}}, \mathcal{G}_{\mathsf{GC}})$*-hybrid model,* π_{FLEX} *UC-realizes* FLEX *and* π_{FLEX2} *UC-realizes* FLEX2*. Formally, for every PPT adversary* $\mathcal{A}$ *there exists a PPT simulator* Sim *such that for every PPT environment* $\mathcal{Z}$*,*

$$\mathsf{Real}_{\pi_{\mathsf{FLEX}},\mathcal{A},\mathcal{Z}} \approx_{\mathsf{c}} \mathsf{Ideal}_{\mathsf{FLEX},\mathsf{Sim},\mathcal{Z}} \quad \textit{and} \quad \mathsf{Real}_{\pi_{\mathsf{FLEX2}},\mathcal{A},\mathcal{Z}} \approx_{\mathsf{c}} \mathsf{Ideal}_{\mathsf{FLEX2},\mathsf{Sim},\mathcal{Z}}.$$

The simulator Sim *receives* $(\mathsf{Leak}(\mathsf{View}_{\mathcal{F}}), o)$ *from the ideal functionality as specified in Algorithms 12 and 13, and produces its simulation using Algorithm 28.*

The distinguishing advantage satisfies the composite bound:

$$\mathsf{Adv}^{\mathsf{euf}}_{\mathcal{G}_{\mathsf{Sig}}} + \mathsf{Adv}^{\mathsf{pre}}_{\mathcal{G}_{\mathsf{Hash}}} + \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF}} + \mathsf{Adv}^{\mathsf{priv}}_{\mathsf{GC}} + \mathsf{Adv}^{\mathsf{auth}}_{\mathsf{GC}} + \mathsf{Adv}^{\mathsf{zk}}_{\mathcal{G}_{\mathsf{SNARK}}} + \mathsf{Adv}^{\mathsf{bind}}_{\mathsf{Lamport/Setup}} + \mathsf{negl}(\lambda).$$

Proof. By definition of UC realization, it suffices to show that for every PPT $\mathcal{A}$ there exists PPT Sim such that for every PPT $\mathcal{Z}$,

$$\big|\Pr[\mathcal{Z} \text{ outputs 1 in } \mathsf{Real}_{\pi,\mathcal{A},\mathcal{Z}}] - \Pr[\mathcal{Z} \text{ outputs 1 in } \mathsf{Ideal}_{\mathcal{F},\mathsf{Sim},\mathcal{Z}}]\big| \le \mathsf{negl}(\lambda),$$

with $(\pi, \mathcal{F}) \in \{(\pi_{\mathsf{FLEX}}, \mathsf{FLEX}), (\pi_{\mathsf{FLEX2}}, \mathsf{FLEX2})\}$.

The proof combines three ingredients:

(1) Simulation (privacy): By Theorem 3 (Part 2, UC-compatible privacy), the simulator Sim receiving $(\mathsf{Leak}(\mathsf{View}_{\mathcal{F}}), o)$ from $\mathcal{F}$ produces a view indistinguishable from the real adversary's view.

(2) Leakage equivalence: By Lemma 2, the leakage $\mathsf{Leak}(\mathsf{View}_{\mathcal{F}})$ provided by $\mathcal{F}$ is identically distributed to $\mathsf{Leak}(\mathsf{View}_{\mathsf{Real}})$, ensuring the simulator has access to the same information in both formulations.

(3) Correctness (soundness): By Theorem 2, the protocol's on-chain outcome matches the ideal functionality's output except with negligible probability.

Simulator Sim *(explicit construction).* We instantiate Sim by case on the corrupted party; the two cases are symmetric.

Algorithm 29 Sim - UC simulator (corrupted evaluator)

1: **Interface with** $\mathsf{FLEX}/\mathsf{FLEX2}$**:** Obtain leakage Leak (public sizes, structure) and the released output $o \in \{P_a, P_b\}$ from the ideal functionality.
2: **Garbling:** $(\widetilde{C}, L_x, L_y) \leftarrow \mathsf{Sim}_{\mathsf{GC}}(1^\lambda, |x|, |y|, o)$.
3: **Labels:** For each input wire delivered to the evaluator, sample a single label $K_i \xleftarrow{\$} \{0,1\}^{n_\ell}$.
4: **Proofs:** For each public instance x, set $\pi \leftarrow \mathsf{Sim}_{\mathsf{SNARK}}(vp, x)$ (zero-knowledge).
5: **Keys & ledger:** Generate honest-party keys through the real key-generation algorithms; construct and sign honest transactions and submit them to $\mathcal{G}_{\mathsf{Ledger}}$; record confirmations/timing meta-data.
6: **I/O:** Forward $\mathcal{A}$'s messages to/from the functionality as prescribed by $\mathsf{FLEX}/\mathsf{FLEX2}$; deliver to $\mathcal{Z}$ the simulated transcript (garbling, labels, proofs, and ledger view).

Algorithm 30 Sim - UC simulator (corrupted sender/garbler)

1: **Relay:** Forward the adversary's messages to $\mathsf{FLEX}/\mathsf{FLEX2}$.
2: **When honest artifacts are needed:** Use $\mathsf{Sim}_{\mathsf{GC}}/\mathsf{Sim}_{\mathsf{SNARK}}$ as above and uniform single labels to respond, consistent with the output o provided by the functionality.
3: **Keys & ledger:** As in the evaluator-corrupted case.

Hybrid outline (privacy component). Let H_0 be the real execution conditioned on the events $\mathsf{GoodBind}$, where the evaluator holds at most one input label per wire, and Corr, where the real and ideal outputs coincide. Exactly as in Theorem 3, we apply the hybrids

$$H_0 \xrightarrow[\mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF}}]{\text{labels} \to \text{uniform}} H_1 \xrightarrow[\mathsf{Adv}^{\mathsf{priv}}_{\mathsf{GC}}]{\text{garbling} \to \text{sim}} H_2 \xrightarrow[\mathsf{Adv}^{\mathsf{zk}}_{\mathcal{G}_{\mathsf{SNARK}}}]{\text{proofs} \to \text{sim}} H_3 \equiv H_4,$$

yielding, for any PPT distinguisher $\mathcal{D}$,

$$\big|\Pr[\mathcal{D}(H_0) = 1] - \Pr[\mathcal{D}(H_4) = 1]\big| \;\le\; \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF}} + \mathsf{Adv}^{\mathsf{priv}}_{\mathsf{GC}} + \mathsf{Adv}^{\mathsf{zk}}_{\mathcal{G}_{\mathsf{SNARK}}} + \mathsf{negl}(\lambda). \tag{1}$$

Note that H_4 is the ideal-world view produced by Sim under the same conditioning $(\mathsf{GoodBind} \wedge \mathsf{Corr})$.

Removing the conditioning and final bound. We now remove the conditioning and apply a union bound:

$$\begin{aligned}
&\big|\Pr[\mathcal{D}(\mathsf{View}_{\mathsf{Real}}) = 1] - \Pr[\mathcal{D}(\mathsf{View}_{\mathsf{Ideal}}) = 1]\big| \\
&\le \big|\Pr[\mathcal{D}(\mathsf{View}_{\mathsf{Real}}) = 1 \mid \mathsf{GoodBind} \wedge \mathsf{Corr}] - \Pr[\mathcal{D}(\mathsf{View}_{\mathsf{Ideal}}) = 1]\big| \\
&\quad + \Pr[\neg\mathsf{GoodBind}] \;+\; \Pr[\neg\mathsf{Corr}].
\end{aligned}$$

The first term is bounded by (1), giving $\mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF}} + \mathsf{Adv}^{\mathsf{priv}}_{\mathsf{GC}} + \mathsf{Adv}^{\mathsf{zk}}_{\mathcal{G}_{\mathsf{SNARK}}} + \mathsf{negl}(\lambda)$. By Lemma 3, $\Pr[\neg\mathsf{GoodBind}] \le \mathsf{Adv}^{\mathsf{bind}}_{\mathsf{Lamport/Setup}}$.

By Theorem 2,

$$\Pr[\neg\mathsf{Corr}] \le \mathsf{Adv}^{\mathsf{euf}}_{\mathcal{G}_{\mathsf{Sig}}} + \mathsf{Adv}^{\mathsf{pre}}_{\mathcal{G}_{\mathsf{Hash}}} + \mathsf{Adv}^{\mathsf{auth}}_{\mathsf{CDS}} + \mathsf{Adv}^{\mathsf{bind}}_{\mathsf{Lamport/Setup}} + \mathsf{negl}(\lambda).$$

Note that $\mathsf{Adv}^{\mathsf{auth}}_{\mathsf{CDS}} \le \mathsf{Adv}^{\mathsf{auth}}_{\mathsf{GC}} + \mathsf{Adv}^{\mathsf{bind}}_{\mathsf{Lamport/Setup}}$ by Lemma 4.

Combining all terms:

$$\begin{aligned}&\big|\Pr[\mathcal{D}(\mathsf{View}_{\mathsf{Real}}) = 1] - \Pr[\mathcal{D}(\mathsf{View}_{\mathsf{Ideal}}) = 1]\big| \\ &\le 2\mathsf{Adv}^{\mathsf{euf}}_{\mathcal{G}_{\mathsf{Sig}}} + \mathsf{Adv}^{\mathsf{pre}}_{\mathcal{G}_{\mathsf{Hash}}} + n \cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF}} + \mathsf{Adv}^{\mathsf{priv}}_{\mathsf{GC}} + \mathsf{Adv}^{\mathsf{auth}}_{\mathsf{CDS}} \\ &\quad + \mathsf{Adv}^{\mathsf{zk}}_{\mathcal{G}_{\mathsf{SNARK}}} + 2\mathsf{Adv}^{\mathsf{bind}}_{\mathsf{Lamport/Setup}} + \mathsf{negl}(\lambda),\end{aligned}$$

where the factor 2 on $\mathsf{Adv}^{\mathsf{euf}}_{\mathcal{G}_{\mathsf{Sig}}}$ comes from the side guess in Algorithm 23, the factor n on $\mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF}}$ comes from the wire-by-wire hybrid, and the $2\mathsf{Adv}^{\mathsf{bind}}_{\mathsf{Lamport/Setup}}$ collects the two uses of input binding in the analysis ($\Pr[\neg\mathsf{GoodBind}] \le \mathsf{Adv}^{\mathsf{bind}}_{\mathsf{Lamport/Setup}}$ from Lemma 3, and $\mathsf{Adv}^{\mathsf{auth}}_{\mathsf{CDS}} \le \mathsf{Adv}^{\mathsf{auth}}_{\mathsf{CDS}} + \mathsf{Adv}^{\mathsf{bind}}_{\mathsf{Lamport/Setup}}$ via Lemma 4).

Since $\mathsf{Adv}^{\mathsf{bind}}_{\mathsf{Lamport/Setup}}$ is itself negligible (by the hash assumptions in Lemma 3), the constant factor does not affect the asymptotic bound, and we obtain the stated composite advantage.

Ledger transcript correctness. In H_4 the simulator constructs honest transactions using honestly generated keys and submits them to $\mathcal{G}_{\mathsf{Ledger}}$, which enforces the same validity and confirmation rules as in the real world. Since the cryptographic payloads were already simulated in hybrids H_1 to H_3, the ledger transcript distribution in H_4 matches the real execution under ($\mathsf{GoodBind} \wedge \mathsf{Corr}$); the residual probabilities of violating these conditions are accounted for above. □

9.6. **Sequential composition theorem.**

Theorem 5 (Sequential composition)**.** *Let $\pi \in \{\pi_{\mathsf{FLEX}}, \pi_{\mathsf{FLEX2}}\}$ and let $\{\pi_i\}_{i=1}^{T}$ be T sequential executions of π in the $(\mathcal{G}_{\mathsf{Ledger}}, \mathcal{G}_{\mathsf{Hash}}, \mathcal{G}_{\mathsf{Sig}}, \mathcal{G}_{\mathsf{SNARK}}, \mathcal{G}_{\mathsf{GC}})$-hybrid model such that:*

(1) instance $i+1$ starts only after all transactions of instance i are k-final on $\mathcal{G}_{\mathsf{Ledger}}$

(2) the per-instance UTXOs and all per-instance secrets/labels are disjoint and domain-separated across $i \in [T]$.

Then for every PPT $\mathcal{A}$ and environment $\mathcal{Z}$ there exists a PPT simulator $\mathsf{Sim}^{\mathsf{seq}}$ such that $\mathsf{Real}_{\langle\pi_1;\ldots;\pi_T\rangle,\mathcal{A},\mathcal{Z}} \approx_{\mathsf{c}} \mathsf{Ideal}_{\langle\mathcal{F}_1;\ldots;\mathcal{F}_T\rangle,\mathsf{Sim}^{\mathsf{seq}},\mathcal{Z}}$, where each $\mathcal{F}_i$ is the ideal functionality corresponding to π_i (FLEX or FLEX2). Moreover, letting $\Delta_\star := n \cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF}} + \mathsf{Adv}^{\mathsf{priv}}_{\mathsf{GC}} + \mathsf{Adv}^{\mathsf{zk}}_{\mathcal{G}_{\mathsf{SNARK}}} + \mathsf{Adv}^{\mathsf{bind}}_{\mathsf{Lamport/Setup}} + \mathsf{negl}(\lambda)$ be the single-instance distinguishing bound from Theorem 4, the overall distinguishing advantage is upper-bounded by $\sum_{i=1}^{T} \Delta_\star = T \cdot \Delta_\star$, which is negligible in λ whenever $T = \mathsf{poly}(\lambda)$ and the underlying advantages are negligible.

Proof. Let π_i denote the i-th execution of π and $\mathcal{F}_i$ its ideal counterpart. Let $\mathsf{View}^{(i)}_{\mathsf{Real}}$ (resp. $\mathsf{View}^{(i)}_{\mathsf{Ideal}}$) be the per-instance transcript/view exposed to $\mathcal{Z}$ in the real (resp. ideal) world. Write $\mathsf{GoodBind}_i$ for the event that input-binding (one-label-per-wire) holds in instance i, Corr_i that the real and ideal outputs coincide in instance i, and $\mathsf{Wrong}_i := \neg\mathsf{Corr}_i$ the wrong-outcome event for i (as in Theorem 2).

Assumption 1 ensures that instance $i+1$ starts from a ledger state where all transactions of i are k-final; assumption 2 ensures domain separation across all inputs (HKDF domains, garbled circuit labels, hash prefixes, etc.) and disjoint UTXO sets across instances.

Remark 6 (Achieving domain separation)**.** *In practice, domain separation is enforced by the* ctx *field in the HKDF salt (Section 7.1, line defining* $K_{i,b}$*). Each dispute instance uses a unique identifier* ctx_i*, for instance: a monotonically increasing instance counter agreed upon by the two parties at setup, or a freshly sampled nonce from* $\{0,1\}^\lambda$*. (Deriving* ctx_i *from the* *StartDispute* *TXID is circular, since the TXID depends on the transaction body and the body depends on* ctx_i *through the pre-signed transactions.) Under either choice,* $K_{i,b}^{(\text{instance } j)} \neq K_{i,b}^{(\text{instance } j')}$ *for* $j \neq j'$ *with overwhelming probability. Similarly, the Lamport commitments* $y_{i,b}$ *include the domain tag* `FLEX:LAMPORT`$\|\mathsf{ctx}_i$*, preventing cross instance collisions.*

Simulator. $\mathsf{Sim}^{\mathsf{seq}}$ runs the single-instance simulator (from the UC Realization theorem) sequentially, respecting the k-finalization barrier.

Algorithm 31 $\mathsf{Sim}^{\mathsf{seq}}$ - sequential simulator

1: **for** $i = 1$ to T **do**
2: Receive leakage Leak_i and released output o_i from $\mathcal{F}_i$.
3: Invoke the single-instance simulator Sim for π_i with input (Leak_i, o_i), producing simulated garbling/labels/proofs and honest-party transactions; submit them to $\mathcal{G}_{\mathsf{Ledger}}$.
4: Wait until all submitted transactions are k-final (through $\mathcal{G}_{\mathsf{Ledger}}$ notifications); deliver the simulated view of instance i to $\mathcal{Z}$.
5: **end for**

Since instances are domain-separated and non-overlapping in time (by assumption 1), the per-instance calls to Sim are independent and their distributions match the single-instance ideal executions.

Hybrid across instances. Let us define hybrids $H^{(j)}$ for $j = 0, \ldots, T$ as follows: $H^{(j)}$ is the execution where the first j instances are run in the ideal world (through Sim as above), and the remaining $T-j$ instances are real. Thus $H^{(0)} = \mathsf{Real}_{\langle \pi_1;\ldots;\pi_T\rangle,\mathcal{A},\mathcal{Z}}$ and $H^{(T)} = \mathsf{Ideal}_{\langle \mathcal{F}_1;\ldots;\mathcal{F}_T\rangle,\mathsf{Sim}^{\mathsf{seq}},\mathcal{Z}}$. Using the triangle inequality:

$$\big|\Pr[\mathcal{Z}(H^{(0)}) = 1] - \Pr[\mathcal{Z}(H^{(T)}) = 1]\big| \;\leq\; \sum_{j=1}^{T} \big|\Pr[\mathcal{Z}(H^{(j-1)}) = 1] - \Pr[\mathcal{Z}(H^{(j)}) = 1]\big|.$$

Fix $j \in [T]$. The only difference between $H^{(j-1)}$ and $H^{(j)}$ is that instance j is switched from real to ideal; all earlier instances are already ideal and later ones have not started (due to assumption 1). Therefore the distinguishing gap for this step is bounded by the single-instance UC bound, namely $\big|\Pr[\mathcal{Z}(H^{(j-1)}) = 1] - \Pr[\mathcal{Z}(H^{(j)}) = 1]\big| \;\leq\; \Delta_\star$, where $\Delta_\star = \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF}} + \mathsf{Adv}^{\mathsf{priv}}_{\mathsf{GC}} + \mathsf{Adv}^{\mathsf{zk}}_{\mathcal{G}_{\mathsf{SNARK}}} + \mathsf{Adv}^{\mathsf{bind}}_{\mathsf{Lamport/Setup}} + \mathsf{negl}(\lambda)$. Summing over j yields

$$\big|\Pr[\mathcal{Z}(H^{(0)}) = 1] - \Pr[\mathcal{Z}(H^{(T)}) = 1]\big| \;\leq\; T \cdot \Delta_\star.$$

When $T = \mathsf{poly}(\lambda)$ and the underlying advantages are negligible, $T \cdot \Delta_\star = \mathsf{negl}(\lambda)$, which proves indistinguishability of the concatenated real and ideal executions. □

10. Conclusion and Future Work

We have presented a formal analysis of FLEX and its enhancement FLEX2, two Bitcoin-layer dispute protocols built on conditional disclosure of secrets via garbled circuits. The main quantitative results are:

(1) *Soundness* (Theorem 2): the probability of a wrong-winner outcome is bounded by $2\mathsf{Adv}^{\mathsf{euf}}_{\mathcal{G}_{\mathsf{Sig}}} + \mathsf{Adv}^{\mathsf{pre}}_{\mathcal{G}_{\mathsf{Hash}}} + \mathsf{Adv}^{\mathsf{auth}}_{\mathsf{CDS}} + \mathsf{Adv}^{\mathsf{bind}}_{\mathsf{Lamport/Setup}} + \mathsf{negl}(\lambda)$, where the factor 2 on $\mathsf{Adv}^{\mathsf{euf}}_{\mathcal{G}_{\mathsf{Sig}}}$ comes from the side guess over the two honest-party signing keys and $\mathsf{Adv}^{\mathsf{auth}}_{\mathsf{CDS}}$ subsumes the CDS-authenticity contribution via Lemma 4.

(2) *Privacy* (Theorem 3): the evaluator's real view is computationally indistinguishable from the simulated view, with advantage at most $n \cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF}} + \mathsf{Adv}^{\mathsf{priv}}_{\mathsf{GC}} + \mathsf{Adv}^{\mathsf{zk}}_{\mathcal{G}_{\mathsf{SNARK}}} + \mathsf{Adv}^{\mathsf{bind}}_{\mathsf{Lamport/Setup}} + \mathsf{negl}(\lambda)$. The factor n on $\mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF}}$ reflects the wire-by-wire hybrid in the $H_0 \to H_1$ step.

(3) *UC realization* (Theorem 4): π_{FLEX} UC-realizes FLEX and π_{FLEX2} UC-realizes $\mathsf{FLEX2}$ in the $(\mathcal{G}_{\mathsf{Ledger}}, \mathcal{G}_{\mathsf{Hash}}, \mathcal{G}_{\mathsf{Sig}}, \mathcal{G}_{\mathsf{SNARK}}, \mathcal{G}_{\mathsf{GC}})$-hybrid model, with composite distinguishing bound $2\mathsf{Adv}^{\mathsf{euf}}_{\mathcal{G}_{\mathsf{Sig}}} + \mathsf{Adv}^{\mathsf{pre}}_{\mathcal{G}_{\mathsf{Hash}}} + n \cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF}} + \mathsf{Adv}^{\mathsf{priv}}_{\mathsf{GC}} + \mathsf{Adv}^{\mathsf{auth}}_{\mathsf{CDS}} + \mathsf{Adv}^{\mathsf{zk}}_{\mathcal{G}_{\mathsf{SNARK}}} + 2\mathsf{Adv}^{\mathsf{bind}}_{\mathsf{Lamport/Setup}} + \mathsf{negl}(\lambda)$.

(4) *Sequential composition* (Theorem 5): k disjoint, k-finalised instances of either protocol preserve the single-instance guarantees up to a factor k in the privacy/UC bounds, enabling safe pipelining of disputes.

10.1. **Technical contributions.** Beyond the headline theorems, the analysis introduces or refines several constructions of independent interest:

(1) A precise specification of the FLEX transaction DAG with explicit conflict edges (Sections 5.1 and 7.6), including the two-input $\mathsf{EarlyRefund}$ design that creates conflicts on both the synchronisation UTXO S and on pot_A and so supports the mutual-exclusion guarantee of Lemma 1.

(2) A CDS-mode specialisation of garbled circuits that releases exactly one of two preimages (Section 7.3), together with a Lamport-based input-binding mechanism (Section 7.1) and a succinct setup proof of seed knowledge (Section 7.2).

(3) A leakage-bounded privacy notion (Definition 13) that captures the minimal on-chain information any open dispute must reveal, and a leakage-equivalence lemma (Lemma 2) showing that the real and ideal leakages match up to negligible error.

10.2. **Future work.** Several directions remain open for future research.

(1) *Adaptive corruption.* Our analysis is for static corruption. Adaptive corruption would require a simulator that can equivocate prior commitments and may interact non-trivially with the CDS-mode garbling. The recent literature on adaptively secure garbling offers a starting point but does not directly cover the one-preimage-release variant used here.

(2) *Permissionless watchtowers.* FLEX2's watchtower role is permissioned. Extending it to a permissionless setting with economic incentives is a natural next step and would interact with both the soundness budget and the assumption underlying Lemma 1 Part (2).

(3) *Concrete parameters.* The window-schedule contract (Definition 3) specifies asymptotic margins. A concrete parameter recommendation for Bitcoin mainnet would

require characterising worst-case mempool behaviour (in particular pinning attacks and fee-bumping latency) under realistic load.

(4) *Multi-party generalisation.* FLEX is a two-party protocol. An n-party generalisation, in which the CDS garbling releases the secret of the unique winner among n participants, would require generalising both the transaction DAG and the input-binding mechanism.

(5) *Post-quantum migration.* The Schnorr EUF-CMA assumption is the only signature-related cryptographic dependency outside of the hash function. Adapting the analysis to a post-quantum signature scheme, or to a hybrid scheme during a transition period, is a near-term direction as the Bitcoin ecosystem itself begins to consider this migration.

10.3. **Closing remarks.** FLEX and FLEX2 occupy a distinctive position in the Bitcoin-layer dispute landscape: they combine capital efficiency (compared to fully bonded fraud-proof systems) with cryptographic soundness (CDS-mode garbling rather than economic-only guarantees). The formal analysis presented here ties these design choices to specific cryptographic assumptions and quantitative bounds, providing a basis for both deployment and further protocol refinement.

References

[1] Ramon Amela, Shreemoy Mishra, Sergio Demian Lerner, and Javier Álvarez Cid-Fuentes. Union: A trust-minimized bridge for rootstock. 2025.

[2] Lukas Aumayr, Zeta Avarikioti, Robin Linus, Matteo Maffei, Andrea Pelosi, Christos Stefo, and Alexei Zamyatin. BitVM: Quasi-turing complete computation on bitcoin. 2024.

[3] Amos Beimel and Naty Peter. Optimal linear multiparty conditional disclosure of secrets protocols. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 332–362. Springer, 2018.

[4] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE, 2001.

[5] Céline Chevalier and Éric Sageloli. Diving deep into UC: Uncovering and resolving issues in universal composability. Cryptology ePrint Archive, Paper 2025/934, 2025.

[6] Nico Döttling, Sanjam Garg, Vipul Goyal, and Giulio Malavolta. Laconic conditional disclosure of secrets and applications. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 661–685. IEEE, 2019.

[7] Ariel Futoransky, Fadi Barbara, Ramses Fernandez, Gabriel Larotonda, and Sergio Demian Lerner. TOOP: A transfer of ownership protocol over bitcoin. 2025.

[8] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 151–160, 1998.

[9] Sergio Demian Lerner, Ramon Amela, Shreemoy Mishra, Martin Jonas, and Javier Álvarez Cid-Fuentes. Bitvmx: A cpu for universal computation on bitcoin. 2024.

[10] Sergio Demian Lerner, Javier Alvarez Cid-Fuentes, Julian Len, Ramses Fernandez-Valencia, Patricio Gallardo, Nicolas Vescovo, Raul Laprida, Shreemoy Mishra, Federico Jinich, and Diego Masini. RSK: A bitcoin sidechain with stateful smart-contracts. 2022.

[11] Sergio Demian Lerner and Ariel Futoransky. FLEX — capital-efficient optimistic bridges with on-demand security bonds for bitcoin. 2025.

[12] Robin Linus, Lukas Aumayr, Zeta Avarikioti, Matteo Maffei, Andrea Pelosi, Orfeas Thyfronitis Litos, Christos Stefo, David Tse, and Alexei Zamyatin. Bridging bitcoin to second layers via BitVM2. 2025.

[13] Diego Nehab, Gabriel Coutinho de Paula, and Augusto Teixeira. Dave: a decentralized, secure, and lively fraud-proof algorithm. *Distributed Ledger Technologies: Research and Practice*, 2024.

[14] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th annual symposium on foundations of computer science (Sfcs 1986)*, pages 162–167. IEEE, 1986.