BitPriv: A Privacy-Preserving Protocol for DeFi Applications on Bitcoin

Ioannis Alexopoulos*, Zeta Avarikioti*†, Paul Gerhart*‡, Matteo Maffei*, Dominique Schröder*
*TU Wien †Common Prefix ‡Friedrich-Alexander-Universität Erlangen-Nürnberg
{ioannis.alexopoulos, georgia.avarikioti, paul.gerhart, matteo.maffei, dominique.schroeder}@tuwien.ac.at

***Abstract*— Bitcoin secures over a trillion dollars in assets, yet its scripting language limits expressive decentralized-finance applications. BitVM enables arbitrary off-chain computation to be enforced through Bitcoin fraud proofs, but it does not provide privacy for inputs such as bids, quotes, and reserve prices.**

We present BitPriv, to our knowledge the first protocol that conditions payments on the outcome of an arbitrary secure two-party computation over private inputs on unmodified Bitcoin. Its core primitive, the Garbled Hashed Timelock Contract (GHTLC), binds Bitcoin payments to garbled-circuit output labels. Parties evaluate the circuit off-chain, while cut-and-choose detects malformed garbling and BitVM punishes detected violations. Under application-specific bounds, collateral can make deviations unprofitable for rational adversaries.

We instantiate BitPriv in BitMarket, a Bitcoin-native sealed-bid auction, and BitBridge, an intent-based cross-chain bridge. Inputs remain private until the protocol execution fixes the outcome. Optimistic settlement costs less than $3 in both applications. Depending on the backend, BitMarket disputes cost $47–$312 and BitBridge disputes cost $47–$657. BitPriv provides a blueprint for enforceable, privacy-preserving DeFi primitives on Bitcoin without trusted hardware, sidechains, or consensus changes.

I. Introduction

Bitcoin's deliberately constrained scripting language cannot enforce arbitrary application logic on-chain, leaving BTC largely unable to participate natively in the broader DeFi ecosystem. Demand is nevertheless substantial and is currently met through wrapped assets and bridges, exposing users to custody and bridge risk [1]. BitVM [2], [3] offers a path beyond this limitation by encoding arbitrary off-chain computation as a pre-signed challenge-response transaction graph. Two parties can thereby enforce its outcome through on-chain fraud proofs without changing Bitcoin's consensus rules [4], [5].

This ecosystem is already moving into deployment. Bitlayer, backed by $25 million across three private funding rounds, launched a BitVM2-based bridge in mainnet beta [3], [6], [7]. Babylon reports over $6 billion in native BTC staking and is extending native BTC into lending through Aave V4, including fixed-rate borrowing with Aegis [8], [9]. Together, these developments expand Bitcoin-native DeFi from bridging and staking toward credit and other conditional financial applications.

This expansion exposes a missing capability: BitVM's enforcement model requires all computation inputs to be known to both parties. For applications such as sealed-bid auctions, private order matching, or MEV-resistant exchanges, the inputs (bids, quotes, reserve prices) must remain hidden from the counterparty and from the blockchain until settlement. In these settings, revealing inputs before the outcome is fixed enables front-running and order-manipulation attacks: rational parties with access to private inputs can exploit them to extract value at the honest party's expense [10]. BitVM provides no mechanism to evaluate a function over private inputs and enforce the result on-chain, since any fraud-proof dispute requires both parties to reproduce the computation, which they cannot do without knowing each other's inputs.

The problem can be stated precisely: we seek a *privacy-preserving conditioned payment*, namely a Bitcoin settlement that distributes funds according to $g(f(x, y))$, where f is an agreed-upon function, x and y are the respective private inputs of two parties, and neither input is revealed on-chain or to the counterparty beyond what the output implies. This primitive underlies a range of DeFi applications that are well-studied on Ethereum but remain unachievable on Bitcoin.

Prior work and its limits. On expressive platforms, privacy-preserving conditioned payments have been realized using ZK proofs [11], [12], [13], [14], [15], [16], MPC [17], [18], [19], [20], and trusted execution environments [21]. All of these approaches rely on functionality that Bitcoin does not provide: the ability to verify a ZK proof or to maintain private contract state on-chain. Therefore they cannot be ported to native Bitcoin.

Within the Bitcoin literature, three families of approaches fall short of the general primitive. *Application-specific protocols* [22], [23], [24], [25], [26] handle specific functions (coin-flipping, lotteries) using hash-based commitments, but do not generalize to arbitrary f. *Theoretical frameworks for fair MPC* [27], [28], [29], [30], [31] provide strong guarantees but rely on stateful contracts not natively supported by Bitcoin Script. *Off-chain execution systems* [32], [21] achieve privacy by relocating computation to TEEs or separate chains, trading Bitcoin's native security model for external trust assumptions.

To our knowledge, no existing system achieves a general, privacy-preserving two-party conditioned payment that settles natively on Bitcoin without consensus changes, trusted hardware, or a separate execution environment.

The GHTLC primitive. We present BitPriv, built around a primitive we call the *Garbled Hashed Timelock Contract (GHTLC)*. The construction rests on a structural observation about Yao garbled circuits and Bitcoin's Taproot scripting: in a garbled circuit evaluation, the output labels $\{L_j^b\}$, pseudo-random strings encoding each output bit b on wire j, are by

This is the full version of the paper that will appear in the proceedings of the Network and Distributed System Security Symposium (NDSS) 2027.

construction hash preimages. Each label satisfies $H(L_j^b) = c_j^b$ for a publicly known commitment c_j^b. Bitcoin's Taproot outputs can embed these commitments as spending conditions, yielding a payment tree whose leaves are unlockable only by the correct output labels for f.

A standard HTLC conditions payment on knowledge of a hash preimage for an *arbitrary* value: there is no semantic constraint on what that preimage encodes. The GHTLC conditions payment on knowledge of a preimage that is semantically tied to the honest evaluation of $f(x, y)$: only a party who correctly evaluated the garbled circuit on consistent inputs can produce the required labels. This binding is not a property of the on-chain script, since Bitcoin Script cannot verify the computation, but a property of the garbling scheme: a maliciously constructed circuit cannot cause an honest evaluator to produce incorrect output labels without risking cut-and-choose detection. The on-chain script checks that the provided preimages match the committed values. The cryptographic binding between those preimages and the correct function output is enforced off-chain by the structure of garbling.

This observation requires no new Bitcoin opcodes, no script extensions, and no on-chain computation.

Architecture. The GHTLC realizes private conditioned payments against semi-honest adversaries. Extending BitPriv to active security requires addressing two further attack surfaces.

The first is *circuit malformation* by the Garbler: since the Garbler constructs the circuit, it can in principle produce a garbled circuit whose output labels correspond to an incorrect or nonexistent payout for a given input. BitPriv counters this with a cut-and-choose protocol [33]: the Garbler commits to n independently garbled circuits, and the Evaluator opens $n-1$ of them for verification after committing to an evaluation index. This process yields a detection probability of $\epsilon = 1 - 1/n$. The second is *equivocation* by the Evaluator: after the Garbler reveals the check-circuit seeds, the Evaluator holds valid output labels for $n-1$ circuits and could attempt to use a check-circuit's labels to claim a fraudulent payout. BitPriv prevents this by requiring the Evaluator to bind its evaluation index via a Lamport one-time signature [34], whose preimages are hardcoded into the spending scripts. Any equivocation attempt exposes conflicting preimages, constituting a publicly verifiable fraud proof that the Garbler can submit on-chain.

If the Evaluator detects that the Garbler's revealed circuit data is inconsistent with its prior commitments, it can trigger a BitVM-based dispute [2], [3], which adjudicates a misbehavior predicate Φ_{mis} and redistributes collateral accordingly. BitPriv uses BitVM as a black-box interface with a fixed two-algorithm API (Compile, Enforce), so improvements to BitVM backends that reduce dispute cost, latency, or setup overhead reduce BitPriv 's dispute costs without any change to the protocol logic.

Security model. BitPriv realizes *publicly verifiable covert security* [35], [36], [20] with deterrence parameter ϵ (Corollary 1). In this model, the Garbler is treated as a *covert adversary*: one who may deviate from the protocol, but whose deviation will be detected with probability at least ϵ and then penalized on-chain. The Evaluator is treated as *fully malicious*, capturing worst-case withholding and equivocation attacks. Honest executions always deliver the correct allocation.

It is important to be precise about what this model guarantees. In the undetected fraction of Garbler deviations (probability $1-\epsilon$), the security theorem makes no claim about the Evaluator's output or the payout: function manipulation can force settlement on an unfavorable pre-signed allocation, while denial-of-service can leave the Evaluator without spendable output labels and expose its timeout penalty. The rational analysis prices this residual branch (Theorems 2 and 3): with collateral scaled to the application value and detection probability ϵ, every deviation strategy has strictly negative expected utility for a rational Garbler. The resulting guarantee combines cryptographic binding with economic deterrence against rational deviations.

Applications. We instantiate BitPriv in two foundational applications that exemplify, rather than exhaust, the primitive's scope. BitMarket is the first sealed-bid Bitcoin marketplace: a Garbler and one or more Evaluators privately commit to prices, and Bitcoin settlement is conditioned on the auction outcome. Losing bids are never revealed on-chain or to the counterparty before the outcome is fixed: after settlement, the Garbler can reconstruct quotes from revealed labels, so privacy is scoped to the pre-settlement window (§VII-C). This structure prevents pre-settlement bid leakage, which is the specific information asymmetry that enables front-running and order-manipulation in public-input settings, yet it does not address MEV sources that arise elsewhere in the settlement stack. Private auctions also provide a building block for order books, price discovery, and solver markets. BitBridge composes BitMarket with a second, privacy-free BitPriv instance to construct the first intent-based cross-chain bridge from Bitcoin to other expressive chains. Intent-based bridging is already deployed by systems such as Across and deBridge [37], [38]: instead of forcing users to manually choose routes, swap tokens, and manage multiple chains, it lets users express a high-level goal like "exchange 10 BTC for 35 ETH" and competing solvers execute the fastest and cheapest cross-chain path in the background. In our instantiation, Alice locks BTC and receives a destination-chain asset from the winning solver, with BTC release conditioned on a SNARK light-client proof of delivery. Both BitMarket and BitBridge follow the same modular structure: private function evaluation bound to on-chain settlement via GHTLC, with no trusted hardware, trusted party, sidechains, or consensus changes.

Performance. In the optimistic case (no disputes), BitPriv requires two on-chain transactions and approximately $1.68 in fees (at 3 sat/vB, BTC ≈ $81,377, May 2026). For BitMarket, the full pessimistic path costs $47–$312 depending on the BitVM backend [39], [5], [40], [41]. BitBridge's settlement-stage dispute costs $106–$657 depending on backend and disputed path (§VIII-C). The misbehaving party bears these costs, which deter misbehavior when collateral requirements are sized per our compliance analysis. The three backends expose different setup, storage, and on-chain trade-offs. Table II reports these values with their provenance and gives hardware-independent on-chain footprints. For the off-chain computation,

BitPriv's lightweight cut-and-choose (as few as $n = 4$ circuits with appropriate collateral) saves up to 13 hours of garbling time compared to a fully malicious cut-and-choose baseline.

Contributions. Our contributions are summarized as follows:

- We formalize the GHTLC and integrate it into BitPriv, a protocol that enforces arbitrary 2PC outcomes on unmodified Bitcoin without additional trust assumptions (§V).
- We prove BitPriv secure against a publicly verifiable covert Garbler and a fully malicious Evaluator, and derive collateral conditions that deter rational deviations (§VI).
- We instantiate BitPriv in BitMarket, a private sealed-bid marketplace, and BitBridge, an intent-based cross-chain bridge, demonstrating the composition of private auctions with verifiable settlement (§VII).
- We implement BitPriv and quantify its off-chain, presigning, on-chain, and dispute costs across BitVMX, BitVM3, and BABE, including the trade-off between circuit count and collateral (§VIII).

II. Model

In this section, we informally present our model, assumptions, and protocol goals. The formal model is given in Appendix B.

We consider two mutually distrusting parties who wish to jointly compute the outcome of a function without disclosing their private inputs and to condition payments on the Bitcoin blockchain based on such an outcome.

Network and Blockchain Model. We assume parties communicate over secure and authenticated channels, and that messages are delivered within a known bounded delay (synchrony). Furthermore, parties have access to the Bitcoin blockchain, which provides a public, append-only transaction ledger that guarantees (i) safety, i.e., once a transaction is confirmed, it is immutable, and (ii) liveness, i.e., each valid transaction will be included in the ledger within a known time bound [42]. We discuss resilience to adversarial network conditions in §X.

Cryptographic Assumptions. We assume the following standard cryptographic primitives: (i) digital signatures secure against existential forgery under chosen-message attacks (EUF-CMA), and secure one-time signatures; (ii) hash functions modeled as random oracles; (iii) a non-interactive commitment scheme (in the random oracle model) that is hiding and binding; (iv) encryption schemes secure against chosen-plaintext attacks. We provide formal definitions in Appendix C.

Threat Model. Either party may be corrupted. We assume adversaries are polynomial-time bounded, but *rational:* they deviate only when the expected gain exceeds the expected penalty. Corruption is static, fixed before protocol execution. An adversary may attempt to bias the computation, abort prematurely, send inconsistent messages, or otherwise deviate arbitrarily. However, since deviations can be detected and publicly proven on-chain, dishonest behavior is unprofitable. We model this formally in Appendix B using the notion of *covert adversaries* [35].

Protocol Goals. At a high level, our protocol aims to realize the following properties:

Privacy: The parties' inputs remain hidden, except for what can be inferred from the function output.

Correctness: The parties' funds are distributed according to the output of the secure computation, which is correct with high probability.

Financial Fairness: No party can learn the outcome without either delivering it to the other party or losing collateral as compensation.

Guaranteed Output Delivery: If one party aborts or deviates, the honest party can, with high probability, recover their fair outcome or obtain compensation by claiming the adversary's collateral on-chain.

III. Background

A. Garbled Circuits and Oblivious Transfer

We use Yao-style garbled circuits (GC) [43], [44] with the seeded, deterministic garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De}, \mathsf{Ve})$ of [20]. Given a function f, security parameter κ, and a seed, Gb deterministically outputs (F, e, d), comprising the garbled circuit and its encoding and decoding information. An input x is encoded as labels $X \leftarrow \mathsf{En}(e, x)$, evaluated as $Y \leftarrow \mathsf{Ev}(F, X)$, and decoded as $y \leftarrow \mathsf{De}(d, Y)$. The algorithm $\mathsf{Ve}(f, \text{seed}, F, d)$ verifies correct circuit generation. We rely on the standard correctness, privacy, authenticity, and verifiability properties. Optimizations such as Free-XOR [45] and Half-Gates [46], [47] are compatible with our protocol provided they preserve deterministic garbling.

In two-party computation, the Garbler G constructs the circuit and knows its encoding and decoding information. The Evaluator E obtains its input labels through OT [48], [49], learning only the labels corresponding to its input while hiding that input from G. After OT, E holds $X = (X_G \parallel X_E)$ and computes $Y \leftarrow \mathsf{Ev}(F, X)$. Finally, either E sends Y to G for decoding, or G provides the decoding information to E.

Label notation. For circuit F_i, let $L^b_{P,i,j}$ denote party $P \in \{\mathrm{G}, \mathrm{E}\}$'s input-wire label for bit j encoding $b \in \{0, 1\}$. Let $L^b_{\text{out},i,j}$ denote the output-wire label for bit j. We write $L^x_{P,i}$ for the collection of input labels encoding x, and $L^z_{\text{out},i}$ for the collection of output labels encoding z in the circuit with index i.

B. UTXO Model

Bitcoin uses a UTXO model, in which coins are represented as *unspent transaction outputs* (UTXOs). A transaction body $[\mathsf{tx}]$ consists of *inputs*, which consume existing UTXOs, and *outputs*, which create new ones. Each output specifies a value and a locking script. An input takes the form $(\mathsf{txid}, k, \varphi)$, identifying output k of the transaction with identifier $\mathsf{txid} := \mathcal{H}([\mathsf{tx}])$, locked under condition φ. A complete transaction comprises its body together with a *witness* providing the data required to satisfy the locking scripts of the consumed inputs. The sum of output values cannot exceed the sum of input values, and the difference is paid as a transaction fee.

Bitcoin Script supports standard conditions such as signature checks, multisignatures, timelocks, and hash-preimage locks. Taproot [50] lets a UTXO commit to multiple alternative

spending conditions in a Merkle tree, revealing only the executed script leaf, witness, and inclusion proof at spend time.

Pre-Signed Transaction Graphs as Covenants. Bitcoin *covenants* [51] are proposed extensions that allow an output to restrict the scripts of its future spending transactions, but they are not deployed in Bitcoin at the time of writing. We emulate covenant semantics by constraining subsequent spends with multisignature authorization from the protocol parties as demonstrated in [2], [3]. Since transaction identifiers are deterministic prior to broadcast[1], we can presign and uniquely identify all transactions in our protocol. For clarity, we henceforth refer to transactions by descriptive names (e.g., `CommitEvaluationIndex`) rather than their raw txid. Concretely, participating parties pre-sign a directed acyclic graph of transactions (DAG) in which each UTXO encodes a contract *state* and each pre-signed transaction encodes an allowed *transition*. All nonterminal transitions require joint authorization (e.g., an n-of-n multisignature), preventing unilateral deviations from the protocol flow. Terminal transitions provide unilateral, time-locked refunds or punitive exits to guarantee liveness and accountability. Because any off-graph spend lacks the requisite signatures, funds can only move along the pre-authorized path.

C. BitVM

BitVM enables optimistic computation on Bitcoin by executing programs off-chain and enforcing their outcomes through pre-signed dispute transactions. Variants resolve disputes through binary search over an execution trace (BitVM(one)/X [2], [39]), chunked re-execution (BitVM2 [3]), or cryptographic techniques such as garbled circuits and witness encryption (BitVM3-style designs and BABE [5], [40], [41]). BitPriv treats the dispute mechanism as modular: any backend capable of verifying its misbehavior predicate can provide enforcement. These mechanisms do not themselves protect private application inputs. Further details appear in Appendix A.

IV. Technical Overview

This section introduces the core components and design rationale behind BitPriv. We first define the security goal via an ideal functionality, then introduce a novel cryptographic primitive with which we realize this goal in the semi-honest setting, and finally describe how BitPriv extends this primitive to achieve security against rational adversaries.

A. Private Computation Dependent Contract (PCDC)

Our protocol allows two parties, P_A and P_B, to securely evaluate a function $f : \mathcal{I} \times \mathcal{I} \to O$ while keeping their private inputs x and y secret. The primary goals are: **(i) privacy**, i.e., neither party learns more about the other's input than what is implied by $z = f(x, y)$; **(ii) enforceable correctness**, i.e., funds are distributed on-chain according to the computed output z; **(iii) financial fairness**, i.e., no party can keep the outcome secret without losing collateral; and **(iv) guaranteed output delivery**, i.e., after setup no party can abort without incurring a penalty.

[1]Prior to the SegWit update, the txid calculation included the unlocking script (signature), making transactions malleable

To achieve this, the computation is financially backed. Both parties lock an *input amount* and a slashable *collateral amount* into an on-chain UTXO. A payout function $g : O \times C_A \times C_B \to C_A \times C_B$ deterministically allocates these funds based on the output of f, where C_X denotes the set of outputs spendable by party X. In the honest case, parties recover their collateral. However, the collateral of a party can be slashed in the case of misbehavior (e.g., provable deviation from the protocol or unresponsiveness).[2]

We capture the security requirements of this goal via the ideal functionality $\mathcal{F}_{\mathsf{PCDC}}$ (informally). The functionality interacts with the two parties P_A, P_B and the ledger $\mathcal{L}$ as follows:

Ideal Functionality $\mathcal{F}_{\mathsf{PCDC}}$ (Informal)

Parameters: $f : \mathcal{I} \times \mathcal{I} \to O$, payout function g, input/collateral amounts c. **Leakage:** reveal only what is implied by $z = f(x, y)$ and public parameters.

1) **Setup.** On input (f, g, c) from both parties, abort if inconsistent. Receive y from P_B.
2) **Fund.** Wait until both parties lock the agreed funds c on $\mathcal{L}$; otherwise abort.
3) **Evaluate.** Receive x from P_A; if P_A fails to provide x, instruct $\mathcal{L}$ to apply the slashing policy in g and halt. Compute $z \leftarrow f(x, y)$ and send z to P_B.
4) **Settle.** Upon instruction `CONTINUE/ABORT` from P_B: (i) if `CONTINUE`, send z to P_A, instruct $\mathcal{L}$ to unlock funds according to $g(z)$, and halt; (ii) else instruct $\mathcal{L}$ to apply the slashing policy in g and halt.

B. The GHTLC Primitive

To realize the ideal functionality $\mathcal{F}_{\mathsf{PCDC}}$ on Bitcoin, we introduce a novel primitive: the *Garbled Hashed Timelock Contract* (GHTLC). We instantiate this primitive using Yao's Garbled Circuits, assigning the role of Garbler (G) to one party and Evaluator (E) to the other. Conceptually, a GHTLC generalizes the standard HTLC. While an HTLC conditions payment on the revelation of a single random preimage, a GHTLC conditions payment on the outcome of an arbitrary computation $f(x, y)$.

Concretely, a GHTLC is implemented as a Bitcoin locking script with multiple alternative spending conditions. Each condition corresponds to a possible output z of the function f and is locked by the set of the garbled output wire labels associated with that result. Analogous to a standard HTLC, the script includes a timelock spending condition to prevent funds from being permanently locked if the counterparty fails to reveal the necessary secrets. This structure strictly binds on-chain settlement to the off-chain execution result. We directly instantiate the GHTLC primitive in our protocol BitPriv below.

C. BitPriv Protocol Overview

Next, we present BitPriv with a strawman design. We begin with a protocol (S0) that remains secure in the presence of semi-honest adversaries utilizing a GHTLC. We then extend

[2]In some applications, becoming unresponsive may forfeit only enough collateral to cover the counterparty's fees.

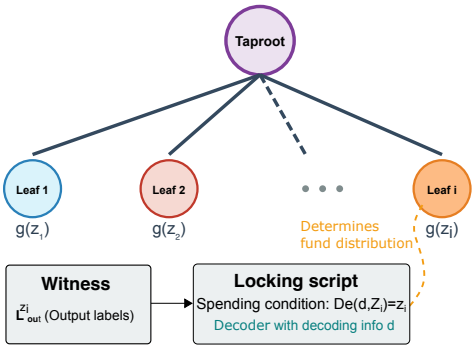

Fig. 1: Taptree implementing the *decoder* of the garbling scheme $\mathcal{G}$. Only the satisfied leaf is revealed on spend (the witness contains the leaf script and a control block proving inclusion).

S0 to security against covert adversaries by introducing three additional techniques, summarized as S1, S2, and S3 in Fig. 2.

S0: Semi-honest realization. The execution begins with G garbling the function f into a circuit F, while E obtains her input labels L_y via Oblivious Transfer (OT). To enforce the binding mechanism, the parties then pre-sign the following sequence of Bitcoin transactions:

(1) Setup locks the parties' inputs and collateral into a shared funding UTXO.

(2) Reveal allows G to spend the Setup output by revealing his input labels L_x within a timelock Δ. The resulting output is a GHTLC: a Taproot output where the leaves of the taptree correspond strictly to the decoding table De of the garbling scheme $\mathcal{G}$ (see Fig. 1).

(3) $\mathsf{Close}^{(z)}$ is a *family* of pre-signed transactions, with one template per possible computation outcome $z \in O$. The spending condition for $\mathsf{Close}^{(z)}$ requires the preimage corresponding to the circuit's output label for z. Thus, once E evaluates the circuit and obtains the valid output label set L_z, she possesses the witness for the corresponding script-path leaf in the Reveal Taproot, so broadcasting it validates $\mathsf{Close}^{(z)}$ and finalizes the fund allocation according to $g(z)$.

S1: Probabilistic Verification (Cut-and-Choose). While S0 successfully realizes $\mathcal{F}_{\mathsf{PCDC}}$ in the semi-honest setting, it is vulnerable to a malicious Garbler G. In particular, S0 remains secure against a malicious Evaluator E who only learns a single output label set due to the authenticity property of the garbled circuit. E can therefore only choose not to spend the GHTLC, refunding G via the timelock spending condition. In contrast, a malicious G could construct a corrupt circuit $F' \neq \mathsf{Gb}(f)$ or manipulate the Taproot leaves such that the output labels allow E to unlock the wrong payout (or no payout at all).

To neutralize G's informational advantage, we employ a cut-and-choose mechanism. G generates n distinct garbled circuits and commits to their descriptions and the random seeds used in their creation. To further ensure the honest execution of the Oblivious Transfer (OT), both parties also commit to the public OT transcript and their associated randomness. E then challenges G to open $n-1$ of these circuits. Using each revealed seed, E deterministically reconstructs the corresponding garbled circuit, confirming its correctness, consistency of labels received through OT, and the binding of output labels to Taproot branches. This technique yields *probabilistic correctness* where the probability of undetected misbehavior is $\frac{1}{n}$.

S2: On-chain Proof of Misbehavior (BitVM). Probabilistic verification is insufficient to prevent cheating by a rational adversary, as the victim needs a way to prove and financially punish the misbehavior on the blockchain.

To achieve this, we use BitVM as a two-party *black-box* mechanism that decides a predicate $\Phi_f(w, o)$ stating that output o is a correct evaluation of a fixed function f on input w.

BitVM Interface

- $\mathsf{BitVM.Compile}(\Phi, \mathsf{Alloc}, P_{\text{params}}) \to C$
- $\mathsf{BitVM.Enforce}(C;\ w, o) \to \mathsf{payout}$

In the BitVM interface, Compile takes a predicate Φ, a payout policy Alloc, and party parameters, and returns a contract object C containing all pre-signed transactions for BitVM, including the funding transaction $C.\mathsf{setupTx}$. Once $C.\mathsf{setupTx}$ confirms on-chain, Enforce ensures eventual settlement conditioned on the value of $\Phi(w, o)$.

During the off-chain setup the parties call BitVM.Compile with a *misbehavior predicate* Φ_{mis} (c.f. §V-C), which captures all possible violations of G, to obtain C. E can fund $C.\mathsf{setupTx}$ dynamically during execution.

Detected deviations are punishable through the BitVM contract: publishing $C.\mathsf{setupTx}$ places the locked funds under the dispute contract, where a valid claim slashes G's collateral and a false claim slashes E's. Thus rational parties dispute only when they can prove misbehavior, and G complies whenever the expected penalty from detection, including collateral and reputation loss, exceeds the gain from an undetected deviation. §VI-B gives the formal bounds.

S3: Preventing Evaluation Index Equivocation. In the cut-and-choose phase (S1), G generates n circuits and E selects one index $m \in [\{1, \ldots, n\}]$ as the *evaluation circuit* that determines the payout, while the remaining $n-1$ are *check circuits* whose secrets G reveals for verification. Bitcoin's stateless scripts cannot enforce that the off-chain verified index m matches the on-chain settlement index. This creates a critical attack: after G reveals the check-circuit seeds, a malicious E learns their complete logic and could use output labels from check circuit m' to unlock funds meant for evaluation circuit m, forcing a favorable payout by equivocating on her index choice.

We prevent this equivocation using a Bitcoin Script-verifiable Lamport one-time signature (OTS) [34]. The OTS binds E irrevocably to m across the entire transaction graph: we hardcode the same Lamport public key into all relevant locking scripts, including payout paths for all n circuits. Settling on index m requires E to sign m, so attempting to settle on $m' \neq m$ requires signing a different value, exposing conflicting preimages for the same public key bits. This conflicting signature is a fraud proof allowing G to slash E's collateral. The binding enables a highly efficient *optimistic path*: honest parties settle privately with

just two transactions, bypassing on-chain adjudication entirely (detailed in §V-B).

Lamport OTS is more than the index-binding mechanism, since it is already implicit throughout the protocol. Revealing one of two committed preimages per output bit in Close *is* a Lamport signature on z under the output-label commitment vector. Revealing input-bit preimages at RS likewise constitutes a Lamport signature (if not for the random permutation hiding G's input). This uniformity enables the cross-instance binding in BitBridge (§VII), where auction-side and settlement-side Lamport keys are tied via script-level equality.

V. BitPriv Protocol Description

This section presents the full BitPriv protocol following the design from Section IV-C. Figure 2 shows the complete flow and how S0-S3 combine.

The protocol runs in two phases: an off-chain setup phase, where the parties generate cryptographic commitments and pre-sign all necessary transactions, followed by an on-chain execution phase. The latter is enacted on the Bitcoin ledger, resulting in either cooperative settlement or dispute resolution.

A. BitPriv: *Off-Chain Setup*

Before any on-chain activity, the parties perform an interactive setup protocol to generate and commit to all necessary cryptographic data. An abort by either party during this phase results in no financial loss. The protocol flow is as follows:

(1) Circuit Generation and Commitment. For each $i \in [n]$, G generates a garbled circuit $(F_i, e_i, d_i) \leftarrow \mathsf{Gb}(1^\kappa, f, \mathsf{seed}_{G,i})$ using an independent random seed $\mathsf{seed}_{G,i}$. G sends commitments $\mathsf{Com}(\mathsf{seed}_{G,i})$, $\mathsf{Com}(F_i)$, and $\mathsf{Com}(d_i)$ to E, along with the plaintext garbled circuits F_i which serve as openings to commitments $\mathsf{Com}(F_i)$. E verifies the structural integrity of each F_i and its corresponding commitment.

(2) Output Label Commitment. For each output wire $j \in [k]$, where k denotes the number of output bits, G constructs two vectors: $\mathsf{Output}_j^0 = [\mathsf{Com}(L^0_{\mathsf{out},i,j})]_{i\in[n]}$ and $\mathsf{Output}_j^1 = [\mathsf{Com}(L^1_{\mathsf{out},i,j})]_{i\in[n]}$, where $L^b_{\mathsf{out},i,j}$ denotes the wire label corresponding to output bit $b \in \{0,1\}$ on wire j in circuit F_i. These vectors are then sent to E. This step ensures that the output decoding (and thus payout conditions) is fixed across all circuits.

(3) Garbler's Input Commitment. To commit to his input x for each circuit F_i while preserving privacy, G commits to his input labels in a randomly permuted order. For each input bit $j \in [l]$ of circuit F_i, G commits to the pair $(L^b_{G,i,j}, L^{1-b}_{G,i,j})$ for a random bit b, and sends the commitment $\mathsf{Com}(X_{i,j})$ to E.

(4) Oblivious Transfer (OT) and Transcript Commitment. For each circuit $i \in [n]$ and each of her input bits $\lambda \in [l]$, E engages in an OT protocol with G to receive her corresponding input label $L^y_{E,i} := L^{y_\lambda}_{E,i,\lambda}$ for all $\lambda \in [l]$. To ensure privacy if the OT transcript becomes part of a fraud proof, E uses her real input y only for a single secret evaluation index m, and dummy inputs (e.g., 0^l) for all other $i \neq m$. To make the protocol reproducible, E also commits to a random seed $\mathsf{seed}_{E,i}$ from which her randomness is drawn throughout the execution for $i \in [n]$ and sends $\mathsf{Com}(\mathsf{seed}_{E,i})$ to G. The parties commit to the full public transcripts of all OT instances, enabling post-facto verification by the BitVM instance.

(5) BitVM Preparation and Pre-signing. The parties fix a predicate Φ_{mis} that, given per-circuit seeds and transcript commitments, (i) re-garbles f from $\mathsf{seed}_{G,i}$ to deterministically reconstruct (F_i, e_i, d_i) and check all relevant commitments, and (ii) reconstructs the public OT transcript from $(\mathsf{seed}_{G,i}, \mathsf{seed}_{E,i})$ and verifies each message against its commitment. Any failure flags misbehavior. Using the interface of §IV-C, they compile $C \leftarrow \mathsf{BitVM.Compile}(\Phi_{\mathsf{mis}}, \mathsf{Alloc}, P_{\mathrm{params}})$. They then collaboratively pre-sign the transaction graph, embedding the necessary commitments and C in the spending scripts. §V-C shows the concrete checks, allocation policy and witness format.

B. BitPriv: *On-Chain Execution*

This phase begins once the funding transaction Setup, which locks all the coins to be distributed according to the protocol logic, is confirmed on the stable prefix of the ledger.

Optimistic Path. If both parties are honest, only the settlement occurs on-chain. Over the authenticated off-chain channel, E sends G a Lamport signature σ_m on her chosen evaluation index m. As explained in §IV-C, G can safely reveal the seed decommitments of the $n-1$ check circuits, and sends E his input labels for F_m, which reveal nothing about x while the seed of F_m stays secret. If E's verification passes and she obtains valid output labels from evaluating F_m, she spends the Setup output via the corresponding Close leaf to distribute funds according to $g(f(x,y))$, completing settlement in two on-chain transactions. If E posts Close using a check-circuit index $m' \neq m$, G detects the equivocation on-chain and slashes E's output via the PunishEquiv path by revealing the conflicting Lamport preimages.

Pessimistic Path. If either party deviates or becomes unresponsive, the protocol defaults to a sequence of on-chain transactions that publicly resolve the outcome, depicted in Fig. 2:

(1) First, E broadcasts a CommitEvaluationIndex (CEI) transaction spending the Setup UTXO. Its witness includes a Lamport signature σ_m on her chosen index m, fixing it on-chain.

(2) G must then spend the CEI output by broadcasting a RevealSeeds (RS) transaction. Its witness contains decommitments for all seeds seed_i with $i \neq m$ and G's input labels for the evaluation circuit F_m. A dedicated Taproot leaf in the CEI output implements the verification logic for each possible m utilizing the commitments established during Setup. The witness also includes a Lamport signature, so G can only reuse the signature from the CEI witness.

(3) Upon observing RS, E verifies the $n-1$ opened circuits and associated transcripts (§V-C), with three possible outcomes:

(a) If all checks pass, E evaluates F_m and broadcasts the Close transaction by proving knowledge of the correct output labels. Funds are then allocated per g (including the equivocation check as described in the optimistic path).

(b) If any of G's revealed data is inconsistent, E initiates a dispute by spending the output of RS with the BitVM

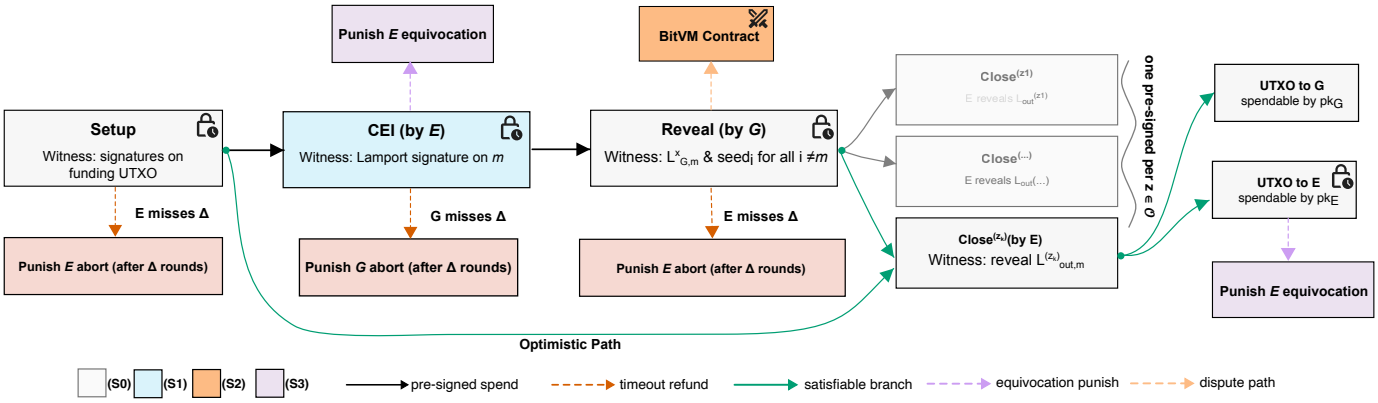

Fig. 2: BitPriv protocol flow. The protocol builds on the foundation of the base protocol (S0), with a cut-and-choose (S1), the BitVM dispute (S2) and finally the equivocation punishment (S3)

contract's funding transaction, C.setupTx. This triggers BitVM.Enforce which handles the distribution of funds according to Alloc (see §V-C).

(c) If E does not spend the output of RS within Δ, either E is withholding the output (violating fairness), or G cheated during cut-and-choose without detection. In either case, G posts $\mathsf{Timeout}_E$[3] to claim this penalty amount.[4]

Transaction specs can be found in Appendix D-D.

C. Dispute Resolution via BitVM

If E detects that G revealed invalid data at RS, she triggers a dispute by publishing the BitVM contract's setup transaction. The dispute mechanism resolves a single question: did G cheat during the cut-and-choose protocol? The answer is determined by a misbehavior predicate Φ_{mis} that replays G's setup commitments against the seeds and transcripts G later revealed.

The misbehavior predicate. Φ_{mis} takes as input a check-circuit index $i^* \in [n] \setminus \{m\}$, an OT instance index $\lambda^* \in [\ell]$, and the revealed seeds $(\mathsf{seed}^*_G, \mathsf{seed}^*_E)$. It deterministically reconstructs what G *should* have committed to during setup and checks whether the revealed data matches those commitments. The predicate returns 1 (misbehavior detected) if any of the following checks fail:

(1) *Seed validity:* The revealed seeds $\mathsf{seed}^*_G, \mathsf{seed}^*_E$ open their respective commitments $\mathsf{Com}(\mathsf{seed}_{G,i^*})$, $\mathsf{Com}(\mathsf{seed}_{E,i^*})$.

(2) *Garbling consistency:* Re-garble circuit i^* from seed^*_G to obtain $(F_{i^*}, e_{i^*}, d_{i^*})$. Check that the recomputed circuit F_{i^*}, decoding info d_{i^*}, and output-label commitments match the values G committed to at Setup.

(3) *Input-label consistency:* For each of G's input wires j, regenerate the permuted input-label pair $X_{i^*,j}$ of circuit i^* from seed^*_G and verify that it matches the commitment $\mathsf{Com}(X_{i^*,j})$ posted at Setup.

(4) *OT transcript consistency:* Reconstruct the public OT transcript for instance (i^*, λ^*) from the revealed seeds and compare it with the transcript committed at Setup. A mismatch in a message of G is misbehavior. A mismatch in a message of E, who supplied seed^*_E herself, invalidates the claim.

If all checks pass or the claim is invalid, $\Phi_{\mathsf{mis}} = 0$ and the claim is false. If a check fails on G's side, $\Phi_{\mathsf{mis}} = 1$, a publicly verifiable fraud proof that G cheated. BitVM.Enforce, instantiated with the allocation policy Alloc fixed at setup, redistributes collateral accordingly: G's deposit is slashed and paid to E if $\Phi_{\mathsf{mis}} = 1$, and E's deposit is slashed if $\Phi_{\mathsf{mis}} = 0$.

VI. BitPriv Security Analysis

A. Security Result

We define security in the standard real/ideal paradigm: a real-world protocol $\mathcal{P}$ is compared to an ideal functionality $\mathcal{F}$, and indistinguishability of the two ensures that any adversary against $\mathcal{P}$ learns no more than $\mathcal{F}$ permits. BitPriv's ideal functionality $\mathcal{F}_{\mathsf{PCDC}}$ extends the publicly verifiable covert (PVC) model with a *fund allocation function* that distributes locked coins according to the computed output. The garbler may deviate to learn the evaluator's input or to violate the allocation, but every such deviation is detected with probability ϵ. The evaluator, who learns the output first, can only violate fairness. On honest execution, the correct allocation is enforced on the ledger $\mathcal{L}$. On detected cheating, $\mathcal{F}_{\mathsf{PCDC}}$ invokes $\mathcal{F}_{\mathsf{BitVM}}$ to slash the misbehaving party so the honest participant receives either the output or compensation. Liveness properties i.e., financial fairness and guaranteed output delivery are enforced through $\mathcal{L}$. If cheating goes undetected (probability $1-\epsilon$), $\mathcal{F}_{\mathsf{PCDC}}$ offers no guarantees to the evaluator.

Theorem 1. *Assume a straight-line extractable commitment scheme in the global random oracle model, the garbling scheme is secure, Π_{OT} UC-realizes $\mathcal{F}_{OT}$, and the signature scheme is EUF-CMA. Then* BitPriv *securely realizes $\mathcal{F}_{\mathsf{PCDC}}$ in the $(\mathcal{F}_{BitVM}, \mathcal{L}, \mathcal{F}_{OT})$-hybrid model.*

[3] $\mathsf{Timeout}_X$ denotes the transaction punishing party X for being unresponsive.

[4] There is no way to distinguish between the two cases, so we design the "cheating" penalty to be higher than the timeout penalty in expectation to align incentives, ensuring this path will not be followed in practice. See §VI-B.

The proof is deferred to Appendix F. We provide a high-level intuition here. Parties cannot deviate from the protocol encoded by the pre-signed transaction graph without forging signatures or violating the guarantees of the ideal ledger $\mathcal{L}$. Privacy and correctness follow from the security of the garbled-circuit protocol and OT. The correct allocation is enforced since payouts are locked behind label-hash preimages, modeled as a random oracle. Fairness and guaranteed output delivery are ensured by timelocked Bitcoin transactions. If the garbler deviates and is detected, the evaluator invokes $\mathcal{F}_{\mathsf{BitVM}}$ to publish a fraud proof and slash the garbler's collateral. Soundness and completeness of this proof follow from the binding of the commitment scheme and the authenticity and verifiability of the garbling scheme.

B. Compliance of Rational Parties

Corollary 1 guarantees detection with probability $\epsilon = \frac{n-1}{n}$. Honesty additionally requires every deviation to be a net expected loss. The pre-signed transaction graph fixes all on-chain outcomes at Setup. This leaves G two attack strategies: *function manipulation* (bias the payout or leak σ bits of E's input) and *denial-of-service* (produce unsatisfiable output labels with probability p), while E can only withhold the output.

Theorem 2 (Garbler Compliance, simplified)**.** *Let COL_G be G's slashable collateral, IN_G its locked input, R its detection reputation cost, $u_{G,\max}$ the largest pre-signed G-payout, $\alpha(\sigma)$ its utility from σ-bit leakage, $\alpha_{\text{DoS}}(p)$ its utility from denial of service with probability p, and γ the evaluator timeout penalty. Every G-deviation is a net expected loss whenever*

$$\text{COL}_G + \text{IN}_G + R > \frac{1}{n-1} \max\left(\alpha(\sigma^*) + u_{G,\max},\ \alpha_{\text{DoS}}(p^*) + \gamma + \text{IN}_G\right).$$

Theorem 3 (Evaluator Compliance)**.** *With withholding utility β and r_E the reputation cost, E complies whenever $\gamma + r_E > \beta$.*

The RHS of the first bound is G's undetected gain, and the LHS is its loss. The factor $1/(n-1) = (1-\epsilon)/\epsilon$ shows that larger n relaxes the collateral requirement. E's bound is simpler: withholding is detected deterministically. The analysis above ignores transaction fees. In practice, the collateral of both parties must also exceed the dispute cost, ensuring that whichever party triggers the dispute can recover its fees from the slashed collateral. Proofs and the derivation of σ^*, p^* are in Appendix G.

C. Collateral with Dispute Fees

Let D denote the backend-specific dispute cost reported in §VIII-C and Table II. We include D in the collateral requirement so that an honest disputer can recover its fees without reducing the economic penalty required by Theorem 2.

We fix the parties' locked inputs and conservatively set $u_{G,\max} = \text{IN}_G + \text{IN}_E$, corresponding to the largest pre-signed allocation that G can receive. We also assume no reputational costs ($R = r_E = 0$). Because information value is application-specific, we use the conservative estimate $\alpha(\sigma^*) = \alpha_{\text{DoS}}(p^*) = \alpha_{\text{full}}$. This values both partial leakage and the one-bit signal from selective denial of service as highly as learning E's full input.

Under these parameters, the minimum total slashable stake of G is $C_G^{\min} = D + (\alpha_{\text{full}} + \text{IN}_G + \max\{\text{IN}_E, \gamma\})/(n-1)$, and the separate collateral deposit is $\text{COL}_G^{\min} = \max\{0, C_G^{\min} - \text{IN}_G\}$. The locked input IN_G is subtracted because it is already slashable. In an honest execution, the collateral is returned and the inputs are distributed according to the computed outcome. If G is proven to have deviated, its input and collateral are slashed: E recovers D, and the remainder provides compensation. Figure 3 shows $C_G^{\min}$ as a function of n, γ, and α_{full} using the BitVM3 dispute backend. For example, at n=4, α_{full}=\$500, and $\gamma \le$ \$1,000, $C_G^{\min}$ evaluates to \$878, which is below the \$1,000 locked input, so the input alone covers the requirement.

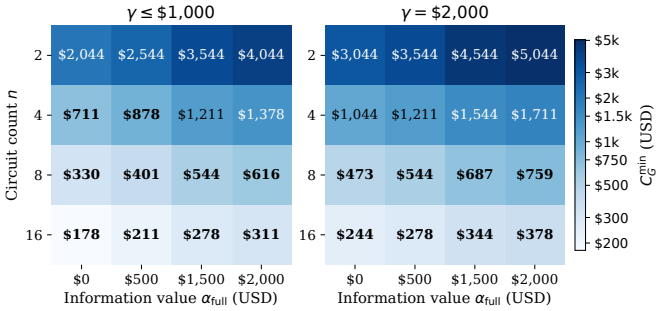

Fig. 3: Minimum total slashable Garbler stake $C_G^{\min}$ using the BitVM3 dispute cost D = \$43.75. Each party locks \$1,000, so $\text{IN}_G = \text{IN}_E$ = 1,000 and $u_{G,\max}$ = 2,000. We set $R = r_E = 0$ and $\alpha(\sigma^*) = \alpha_{\text{DoS}}(p^*) = \alpha_{\text{full}}$. Bold entries satisfy $C_G^{\min} \le \text{IN}_G$, meaning that the locked input alone covers the requirement.

The deployment declares the information-value ceiling used to derive its collateral requirement before participation. In practice, each party runs the protocol only if it considers the counterparty's posted collateral sufficient for its application-specific value at risk. Further parameter sweeps appear in Appendix G-C. Cases in which no finite collateral suffices are discussed in Appendix K.

VII. Applications enabled by BitPriv

A. BitMarket

In a sealed-bid auction, a *seller* with a private reserve price (the minimum they will accept, denoted ask) sells an asset to one of t *buyers*[5], each of whom submits a private bid. The winner is the highest bid exceeding the ask. The auction fails otherwise. The clearing rule is parametric: the winner pays either its own bid (first-price [52]) or the runner-up's bid (second-price [53]). Sealed-bid auctions are central to mechanism design because they enable price discovery without bid leakage. The second-price variant is *incentive-compatible*, meaning each buyer's dominant strategy is to bid their true valuation [53].

Realizing such an auction trustlessly on a blockchain requires keeping bids hidden until the outcome is fixed and binding settlement to the auction result. On expressive chains this has been achieved through ZK proofs over private contract state [11], [14], [19], trusted execution environments [54], [21], or a trusted auctioneer [55], none of which transfers directly to Bitcoin.

We close this gap by presenting BitMarket, the first Bitcoin-native sealed-bid auction protocol secure against rational adver-

[5]The construction also supports the reverse setting of a single buyer and multiple sellers without modification.

saries [56], built by instantiating BitPriv's GHTLC primitive with a sealed-bid clearing circuit. BitMarket requires neither consensus changes nor external trust.

Multi-party adaptation. The auction involves t buyers $B_1, \ldots, B_t$ and one seller acting as garbler. Base BitPriv requires two changes: *(i)* the seller runs a separate OT with each buyer per garbled circuit, so each B_i obtains labels for its bid; *(ii)* after the seller reveals check-circuit seeds and its own input labels in RS, each buyer reveals its own input labels for index m on-chain, allowing any buyer to aggregate and evaluate F_m off-chain. Fig. 4 shows this auction stage within the BitBridge on-chain flow.

RS output structure. The RS spend creates one Taproot output with two classes of leaves. *Dispute leaves* (one per buyer) fund a dispute against the seller via BitVM. Since RS is a single UTXO, at most one dispute can execute, capping worst-case cost for the seller independently of t. The *transition leaf* becomes spendable after Δ_{disp} and creates t+1 UTXOs: per-buyer input-reveal UTXOs, where each B_i reclaims a deposit by revealing its F_m labels or forfeits it to the other parties on timeout; and an extended Close Taproot output, replicated per buyer so that the broadcaster is identifiable and the equivocation penalty can be attributed (cf. §V, S3). The seller's payout is further locked behind a trade-fulfillment condition, making payment atomic with delivery. Non-delivery slashes the seller's collateral, as instantiated in §VII-B.

Execution sketch. After Setup: buyers jointly[6] select m and post CEI with σ_m. The seller posts RS; during $[0, \Delta_{\mathsf{disp}}]$ any buyer may dispute; otherwise a buyer spends the transition leaf, each buyer reveals its labels to reclaim its deposit, and any buyer evaluates F_m and broadcasts Close (Appendix H).

B. BitBridge

Bitcoin secures over a trillion dollars in assets, yet moving BTC to another chain at a competitive price without trusting a centralized intermediary remains an open problem, despite clear demand: tens of billions of dollars in BTC already circulate as wrapped or bridged assets on other chains, and custodial bridge failures account for billions in losses to date [1]. We use BitMarket to close this gap, presenting BitBridge: the first trust-minimized intent-based Bitcoin bridge. To motivate our design, we present the existing approaches and their limitations.

Atomic swaps [57] are trustless and require no custodian, but impose a significant burden on the user: they must run a light client or trust a wallet to verify that the counterparty's setup on the destination chain is correct before committing funds, and they negotiate bilaterally, forfeiting price competition.

Lock-and-mint bridges are the dominant deployed solution but are custodial: their security reduces to a multisig or validator set, which has been the primary source of cross-chain exploits in practice [1]. Recent BitVM2-based designs [3] enable trust-minimized minting of wrapped Bitcoin, but require an on-chain Bitcoin light client on the destination chain, incurring prohibitive cost and latency.

[6]For example, drawn from a public randomness beacon or Bitcoin itself.

Intent-based bridges [37], [38] overcome both limitations. The user declares an intent: deliver at least $a_{\mathsf{eth}}^{\min}$ ETH to a_{addr} in exchange for a_{btc} BTC. Professional solvers, holding liquidity on multiple chains, compete to fulfill this intent. The winning solver immediately fronts the destination-chain asset. The user's BTC is released only after verified delivery. This reduces latency to a single source-chain confirmation and shifts destination-chain verification to the solver. Despite being the de facto design in practice, existing deployments rely on a trusted auctioneer to select the winning solver and clearing price, reintroducing exactly the trust assumption the design was meant to eliminate.

BitBridge is the first intent-based bridge on Bitcoin to remove this final trust assumption, achieving both *trustless price discovery and settlement on Bitcoin* itself. To do so, it instantiates two BitPriv instances: a *sealed-bid auction* in which Alice privately commits to a reserve price, solvers privately submit quotes, and the best quote above the reserve wins (BitMarket, §VII-A); and a *privacy-free settlement* that enforces cross-chain delivery atomically against that outcome. The two instances are produced by different parties (Alice garbles the auction circuit, while the winning solver MM_w garbles the settlement circuit) and are bound together via a chain of Lamport one-time signatures, as described below. The full protocol pseudocode is in Appendix H.

(1) Auction layer (BitMarket). The auction layer is a multi-party adaptation of BitPriv with Alice as Garbler and t solvers $MM_1, \ldots, MM_t$ as parallel input contributors on the Evaluator side. Alice's private input is her reserve price p_{reserve}. Each solver MM_i inputs a private quote b_i. The auction circuit is

$$f_{\mathsf{auc}}(p_{\mathsf{reserve}}, b_1, \ldots, b_t) = \begin{cases} (w, a_{\mathsf{eth}}) & \text{if } \max_i b_i \geq p_{\mathsf{reserve}}, \\ \bot & \text{otherwise,} \end{cases}$$

where $w = \arg\max_i b_i$ is the winning index. The clearing amount a_{eth} is determined by a public clearing rule fixed at setup (first-price: $a_{\mathsf{eth}} = a_{\mathsf{btc}} \cdot b_w$; second-price [53]: $a_{\mathsf{eth}} = a_{\mathsf{btc}} \cdot b_{w'}$). The construction is independent of the rule.

(2) Settlement layer. The settlement layer is a two-party privacy-free BitPriv instance between Alice and MM_w, with roles flipped: MM_w acts as Garbler and Alice as Evaluator. The garbled circuit implements the cross-chain delivery predicate $\Phi_{\mathsf{settle}}(w, a_{\mathsf{eth}}; \pi)$, which is satisfied if and only if π is a SNARK proof (Appendix C-A) that a finalized destination-chain transaction transfers at least a_{eth} from MM_w to a_{addr}. Because the settlement instance has no private inputs on Alice's side, the OT phase and input-label permutation of base BitPriv are dropped.

At setup, *every* solver MM_i prepares its own settlement instance with Alice: it garbles its copy of Φ_{settle}, posts the cut-and-choose commitments, and pre-signs the corresponding transaction graph. After the auction, only the instance belonging to MM_w is activated, and the remaining $t-1$ are discarded. This pre-commitment is required for atomicity: the auction output must immediately hand control of Alice's BTC to a settlement path that is already fixed and enforceable. Without it, a winner could refuse to set up settlement after learning it won.

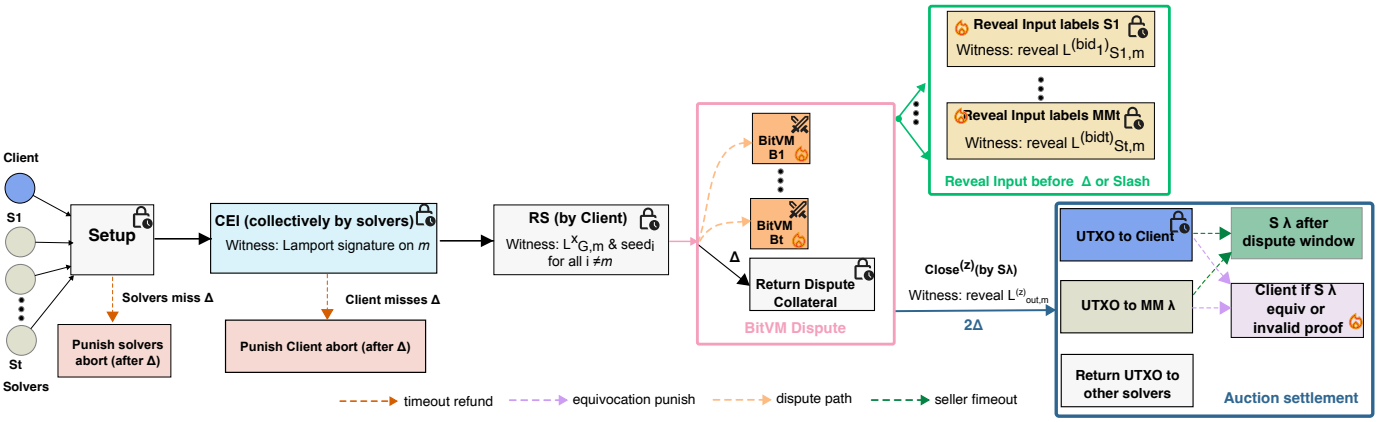

Fig. 4: BitBridge on-chain flow. (a) Pink: per-solver dispute leaves, used if any solver contests Alice's auction setup. (b) Green: solver input-label reveals, used to reclaim reveal deposits and make the auction evaluable off-chain. (c) Blue: cross-chain settlement through the winning solver's pre-committed BitPriv instance. The figure labels Alice as Client and the solvers MM_i as S_i.

(3) Binding auction to settlement. Without an explicit link, MM_w could wire its settlement circuit to verify delivery for some tuple $(w', a'_{\text{eth}}) \neq (w, a_{\text{eth}})$. We rule this out with three Lamport OTS keys, two script-level equality checks, and one equivocation check. Recall that GC output/input labels can be interpreted as Lamport signatures (cf. Section IV-C).

(a) ℓ_{auc} is implicit in the auction circuit's output-label commitments: producing the labels for (w, a_{eth}) is equivalent to $\sigma_{\ell_{\text{auc}}}(w, a_{\text{eth}})$, unforgeable except by honest evaluation of F_m.

(b) ℓ_b is a fresh Lamport key generated by MM_w at setup and used to bridge the auction and settlement instances.

(c) ℓ_{stmt} is implicit in the public-statement input-label commitments of MM_w's settlement circuit: revealing the statement labels for ϕ in settlement RS is equivalent to $\sigma_{\ell_{\text{stmt}}}(\phi)$.

The auction Close script requires $\sigma_{\ell_{\text{auc}}}(w, a_{\text{eth}})$ and $\sigma_{\ell_b}(\phi_1)$, and checks $\phi_1 = (w, a_{\text{eth}})$. Hence MM_w signs the auction outcome under ℓ_b. The settlement RS script requires $\sigma_{\ell_b}(\phi_2)$ and the settlement statement labels, equivalently $\sigma_{\ell_{\text{stmt}}}(\phi_3)$, and checks $\phi_2 = \phi_3$. Hence the statement wired into the settlement verifier equals the value signed under ℓ_b.

These checks imply $\phi_1 = (w, a_{\text{eth}})$ and $\phi_2 = \phi_3$. The only remaining gap, $\phi_1 = \phi_2$, is enforced by the one-time property of ℓ_b: if $\phi_1 \neq \phi_2$, the two witnesses reveal conflicting Lamport preimages, which Alice submits as a fraud proof to slash MM_w's collateral. Thus $\phi_3 = \phi_2 = \phi_1 = (w, a_{\text{eth}})$, so settlement verifies exactly the auction outcome.

C. Discussion

Privacy. Solver quotes remain hidden until the auction outcome is fixed by the on-chain transaction graph. After the solvers reveal their input labels for F_m, Alice, as the Garbler, can reconstruct each quote. This leakage occurs only *after* Alice can no longer alter the current auction outcome without being slashed. It therefore protects the economically relevant phase. Across executions, Alice may learn information about quote distributions, as in standard repeated auctions.

Coalitions. In a permissionless setting, Alice may control one or more solvers. To prevent a coalition from triggering slashing to recover the penalty after extracting competitor information, we first reimburse the honest disputer's backend cost D, then burn a fraction ω of the remaining penalty and redistribute only $(1-\omega)$. Appendix G-D derives a sufficient condition on ω.

Reducing setup latency. The dominant settlement computation is garbling the fixed verifier and constructing its commitments. This is independent of Alice's private input and can be performed speculatively by professional solvers before an auction. This keeps most setup work outside the time-sensitive interval between fixing the auction outcome and completing destination-chain delivery, during which market prices may move. The pre-signed settlement graph remains compact as its binary output requires only $2n$ Close templates. In §X we discuss scalability tradeoffs and Appendix K-D gives further measurements.

Collateral sizing. As a concrete example, take n=16, a \$1,000 bridge amount, an information value α_{MEV} = \$250, and a timeout penalty $\gamma = 2\alpha_{\text{MEV}}$ = \$500. Under BitVM3 or BABE, the client's slashable bridge input covers its requirement in full, while each solver locks \$500. Appendix G-C gives the derivation and other backends.

Beyond cross-chain swaps. BitMarket and BitBridge instantiate one general primitive across a privacy-critical auction circuit and a large privacy-free settlement circuit. The composition pattern (private auction ∘ privacy-free verifier) generalizes to any setting where the settlement condition is a verifiable predicate. Instantiations include fair data exchange for off-chain digital goods [58], rollup data-availability auctions [59], and Bitcoin-native block-space auctions. BitPriv's costs depend on output size and circuit structure (§VIII-A), enabling projection from the target circuit. We sketch these in Appendix H-A.

VIII. Implementation and Evaluation

A central design goal of BitPriv is *modularity*: its intrinsic costs (garbling, oblivious transfer, and on-chain settlement) are independent of the dispute backend. We measure these costs using the BitMarket auction circuit, compare BitVMX [39],

BitVM3 [5], [40], and BABE [41], derive end-to-end costs for BitMarket and BitBridge, and compare BitPriv's collateral-backed cut-and-choose construction with a fully malicious baseline for BitBridge's larger settlement circuit.

Experimental Setup. We benchmarked BitPriv, BitVMX dispute execution, and SP1/Groth16 proof generation on an AMD EPYC 7B12 (32 vCPUs, 125 GB RAM, Linux 6.17), measuring BitVMX transaction sizes on Bitcoin regtest. BitVM3/BABE setup and on-chain figures are from [41] (Ryzen 7 7840U, 16 threads). Unless stated otherwise, fees assume 3 sat/vB and a BTC price of $81,377 (May 2026).

A. Off-Chain Costs

We instantiate garbling with half-gates [60] and OT with Simplest OT [49] with the garbling and OT protocols being themselves modular: any standard instantiation can be substituted. Results are reported per BitPriv instance: BitMarket uses one instance, while BitBridge composes two (§VIII-B).

BitMarket. The BitMarket circuit implements a sealed-bid auction for t buyers and 16-bit prices using parallel comparisons and tournament argmax. Measured gate counts include winner-selection and output logic. A 1,000-satoshi tick gives 2^{16} levels up to 0.65535 BTC (≈ $53,330), at ≈ $0.81 granularity.

Table I reports setup versus t for n=16, using EMP [60]. Both columns cover all n circuits the garbler produces at setup, at 26 ns per AND gate and 117–134 μs per OT.

TABLE I: Measured BitMarket off-chain setup costs (k=n=16). Garbling and OT scale linearly with t, and OT takes ≥99% of total time, while evaluating one circuit takes < 1 ms throughout.

t	AND	E-input bits	Garble	OT	Storage
1	47	16	0.019 ms	34.3 ms	53.0 KB
4	338	64	0.141 ms	120.1 ms	282.0 KB
8	733	128	0.313 ms	248.0 ms	591.0 KB

Full pre-signing cost. Setup constructs and signs Close for every evaluation index and output, while BitMarket replicates the payout tree per buyer (§VII-A): $tn2^k$ transactions with t+1 signatures each, of which the designated broadcaster withholds its own until broadcast and the remaining t are exchanged during setup. Each template corresponds to one Taproot leaf, so the template and leaf counts coincide. For n=16, k=16, and t=1, the 1,048,576 templates take 247.1 s and yield 134 MB of signature material. Each party constructs and maintains the fully signed graph locally (2.24 GB). The template count scales linearly with n, increasing t raises both the template count and the signatures per template, and packing reduces serialization overhead (Appendix I-A).

BitBridge reuses the auction setup above and adds one privacy-free settlement instance per solver. The public Groth16 verifier [61] has 2.7×10^9 non-free gates. Garbling one circuit takes 269.6 s and Merkle committing to it brings a complete setup to 599.1 s. Batched setup at n=4 takes 2,145 s and produces 347.5 GB (1.29 GB with the DV SNARK of Appendix K-D) of material per solver. Its binary output requires only $2n$ Close templates per solver. The complete n- and solver-scaling results are in Appendix K-D.

TABLE II: Dispute costs by workload and backend. *Active* includes evaluation and SP1/Groth16 proof generation; BitVMX entries pair seconds/steps. M/D/R: measured/derived/reported.

Backend	Setup (s)	Active (s)	Peak RAM	Backend vB / USD
Auction predicate				
BitVMX^{M+D}	89.4	1.40/6.82M	38.5 MB	≈ 126,421/309
BitVM3^{M+R}	14,241	845.2	29.55 GB	17,920/44
BABE^{M+R}	842.5	620.8	29.55 GB	26,789/65
Settlement predicate				
BitVMX^{M+D}	10,484	3.87/17.17M	43.7 MB	243,436/594
BitVM3^{M+R}	14,241	2,875.0	106.93 GB	17,920/44
BABE^{M+R}	842.5	2,650.6	106.93 GB	26,789/65

B. On-Chain Costs

BitMarket. Using Packed-SECP, the optimistic path broadcasts only Setup and Close, for a total of 690 vB ($1.68). If off-chain cooperation fails, the protocol proceeds on-chain through CEI and RS, at a total cost of 1,367 vB ($3.34). Appendix I-C gives the per-transaction breakdown and the asymptotic scaling.

BitBridge. BitBridge composes the auction path above with a settlement instance whose input is a Groth16 proof ($l_{\mathsf{set}} \approx$ 1,024 bits). The approximate footprints are 1,205 vB ($2.94) for optimistic execution and 25,680 vB ($62.69) for the settlement liveness path, the latter including a 23,400-vB settlement RS.

C. Dispute Mechanisms and Costs

BitPriv exposes enforcement through a black-box interface. Therefore, any mechanism capable of verifying the misbehavior predicate Φ_{mis} on Bitcoin can serve as its dispute backend. This separation allows BitPriv to adopt new BitVM constructions without changing the protocol logic. Only backend-specific setup and enforcement costs change. Table II and Figure 5 compare BitVMX, BitVM3, and BABE for both applications.

BitMarket. The misbehavior predicate Φ_{mis} determines whether the Garbler deviated during setup. For the selected check circuit, it re-garbles the function from the revealed Garbler seed and verifies the circuit, decoding information, output-label commitments, and Garbler input-label commitments against the values fixed at setup. It also re-derives one challenged OT instance from the revealed seeds and checks its transcript against the corresponding commitment. For the two-buyer (t=2) auction with n=16, executing Φ_{mis} on the BitVMX RISC-V emulator produces a 6.82-million-step trace, takes 1.40 s, and uses 38.5 MB of peak memory. OT verification accounts for 83.1% of the trace. Scaling with t, the OT component remains nearly constant and the increase in total trace length is therefore driven by re-garbling the larger auction circuit (Appendix I-B). Including the liveness path, BitMarket disputes cost $47–$312, depending on the backend (Table XI).

BitBridge. The BitMarket predicate verifies a single digest by re-garbling the entire circuit. Applying the same approach to the 2.7×10^9-gate settlement verifier in the naive way would require roughly two months. To avoid this overhead, we slightly

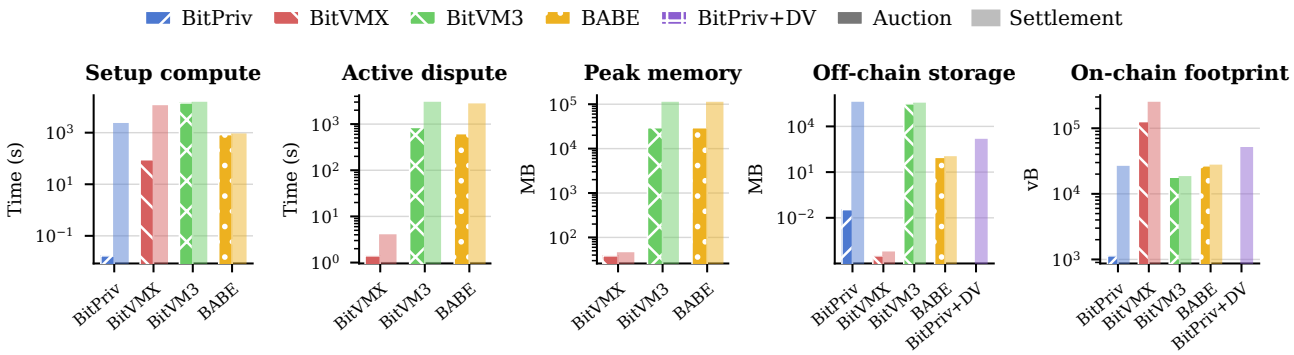

Fig. 5: BitPriv intrinsic costs and dispute-backend trade-offs for the auction and settlement predicates (log scale). The BitPriv bars use n=4 (t=2 for the auction) and report setup time, storage, and liveness-path on-chain costs. Settlement setup includes one Merkle commitment, while BABE/BitVM3 setup and storage are shared across workloads. Detailed measurements and provenance appear in Table II and Appendix I-D. We discuss the DV SNARK option in Appendix K-D.

enhance the garbling tables so that, in the event of a dispute, only a single gate needs to be recomputed. In our enhanced strategy, we store for each garbled table the XOR between its output label and a secret derived from the seed, and commit a single Merkle root over these extended garbled tables.[7] By doing this, every wire label is recomputable from the seed, and recomputing a dispute requires only the contested gate's input labels, opening the Merkle commitment, and regarbling a single gate. With our enhanced strategy, the BitVMX predicate is then practical with 17.17M steps and 3.87 s computation time. Our enhanced strategy preserves the guarantees of §V-C, as it only changes how check (2) of Φ_{mis} is evaluated (Appendix I-D). Our Groth16 proof generation takes 2,650 s with 106.93 GB peak memory. Settlement disputes cost \$106–\$657, depending on the backend. Because the auction-dispute and settlement-transition paths spend the same RS output, the two stages are mutually exclusive. Reachable BitBridge disputes therefore span \$47–\$657 (Appendix I-D).

Backend trade-offs. BitVMX has the lowest dispute-time memory but the largest on-chain footprint. BitVM3 makes the opposite trade, with the smallest footprint but the greatest setup time and storage. BABE lies between both. The BitVM3 and BABE runtimes come from [41] and indicate order of magnitude only, not same-platform performance. Their on-chain sizes and round counts are hardware independent. Latency also differs, as with one-day challenge windows, the measured BitVMX auction path spans ≈11 days, against 2–3 for the SNARK-backed paths. The SNARK backends further require ≈30 GB to prove an auction dispute and ≈107 GB for settlement, a cost incurred only on that path. The cost of renting appropriate hardware can therefore be incorporated in the refundable collateral.

Fee sensitivity. The costs above assume 3 sat/vB and scale linearly with the feerate. Table XI reports complete BitMarket and BitBridge paths for 3 – 300 sat/vB.

D. Comparison with Fully Malicious Cut-and-Choose

***n* as a free parameter.** In BitPriv, n is not fixed. The simplified compliance condition requires total slashable stake $\mathrm{COL}_G + \mathrm{IN}_G + R > \frac{v}{(n-1)}$, where v is the maximum cheating gain (Theorem 2). Reducing n raises the required collateral, but collateral is *fully refunded* to honest parties and costs only the time-value of locked capital, while garbling, OT, and storage are always paid. This yields a direct trade-off: fewer circuits at higher collateral.

[7]Our enhanced strategy is inspired by [62], where gate outputs are provided as auxiliary information to evaluate multiplication gates non-interactively.

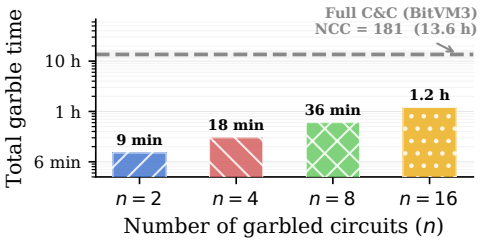

Fig. 6: Single-thread garble time for BitBridge's Groth16 verifier [61] as a function of n. The dashed line marks the full-malicious C&C threshold [41].

Case study: BitBridge settlement at n=4. Full-malicious C&C at 2^{-40} soundness requires NCC=181 garbled circuits [41]. With n=4, BitPriv garbles 177 fewer circuits, saving $(181-4) \times 269.6\,\mathrm{s} \approx 13.3\,\mathrm{h}$ of off-chain compute per execution (c.f. Fig. 6). The enforcement layer adds a one-time setup cost (Table II).

IX. Related Work

(Publicly verifiable) MPC with covert adversaries. The concept of security against *honest-looking* adversaries, those who deviate only when detection is unlikely, was introduced by Canetti–Ostrovsky [63]. Aumann and Lindell [35] provided the first formalizations, later extended to the dishonest-majority setting by Goyal et al. [64]. To strengthen deterrence, Asharov and Orlandi introduced *covert security with public verifiability* (PVC), allowing an honest party to produce a public proof of cheating. Subsequent work made PVC more efficient through signed OT extensions and simpler defenses against selective-OT attacks [36], [65], [66].

Zhu et al. [20] brought this paradigm on-chain, implementing a 2PC judge as an Ethereum smart contract to enable *financially secure* 2PC. The model was further refined with the notion of *robust PVC security* to limit the benefits of undetected

cheating [67], and generalized by Faust et al. [68], who formalized *financially backed covert security (FBC)*. We extend this paradigm to Bitcoin, where the absence of expressive smart contracts requires a novel enforcement mechanism.

Private smart contracts on expressive blockchains. Privacy on expressive blockchains has been pursued through several cryptographic paradigms. Early zkSNARK-based systems like Hawk [11] required a trusted manager, a centralization point that persists in later function-hiding protocols like ZEXE and VeriZEXE [12], [13], while other tools have suffered from information leakage or anonymity breaks [16], [15]. Efforts to decentralize this trust with MPC, as in zkHawk [17], [18], introduced high client availability requirements, whereas others like Eagle [14] shift trust to an external committee. Alternative approaches rely on homomorphic encryption, which can expose client identities [69], [70], or on trusted execution environments (TEEs), which introduce trusted hardware assumptions [21]. While recent work targets even stronger guarantees like *doubly private* contracts [19], they fundamentally depend on either introducing external trust assumptions or requiring expressive on-chain logic that is unavailable in Bitcoin.

Secure computation with public ledgers. Early work used public ledgers to achieve *fairness* in specific applications like lotteries and games, often assuming an enhanced version of Bitcoin [22], [23], [24]. This approach was later generalized to arbitrary secure function evaluation and to stateful settings with *secure cash distribution with penalties* [25], [26]. The interaction between MPC and ledgers was formalized in the GUC framework [27], [28], inspiring a line of work on reducing on-chain costs for output fairness [29], [71], [30], [72], [73]. This research culminated in protocols like *Insured MPC*, which provides UC-secure functionalities with identifiable aborts [31]. However, no Bitcoin instantiation of this line conditions payment on the output of a general computation. Lottery-style protocols run on current Bitcoin Script but realize only specific functionalities [23], while deposit-based fair 2PC [22] provides abort compensation but does not allocate the deposits according to the computed output. The claim-or-refund functionality underlying [24], [25], [26] requires the ledger to validate an arbitrary predicate, which Bitcoin Script has no opcodes to express. Concurrent BitVM constructions compile verification into privacy-free garbled circuits [5], [40], [41], but compute on public inputs only. BitPriv is, to our knowledge, the first construction combining arbitrary single-round 2PC, private inputs, and output-dependent payment on unmodified Bitcoin. Table XII compares the closest approaches along these axes.

X. Discussion

Security and economic scope. BitPriv achieves publicly verifiable covert security rather than full malicious security. If malformed garbling escapes cut-and-choose, function manipulation may select an unfavorable pre-signed allocation, while denial-of-service may leave an honest Evaluator unable to settle and expose its timeout penalty. Collateral deters these deviations rather than preventing them, provided that the fee-inclusive compliance condition in §VI-C is satisfied. When the value of leaked information is unbounded, the required collateral is impractical, or adversaries are not economically rational, deployments must instead use a stronger malicious-security construction. Appendix K-A examines these limits and possible mitigations. In multi-user applications such as BitMarket, threshold authorization for Close can additionally reduce griefing and delays caused by non-participating users.

Bitcoin deployment assumptions. The pre-signed transaction graph fixes the authorized payouts, but it does not ensure that the corresponding transactions confirm before their timeouts. In practice, to accommodate fee volatility, each pre-signed transaction includes a dedicated fee-bump output that either party can spend through a CPFP child without changing the parent transaction identifier or invalidating its signatures. Our BitVMX evaluation already includes these. This mechanism addresses ordinary fee spikes and congestion, but not sustained miner censorship. Deployments must therefore choose confirmation depths and timelocks that account for the expected inclusion delay, fee ceiling, and reorganization margin. If these timing assumptions fail, signatures continue to prevent unauthorized allocations, but protocol liveness is no longer guaranteed. Appendix K-B and Table XIprovide the complete analysis.

Privacy and coalitions. BitMarket protects quotes until the committed execution fixes the auction outcome. It does not provide post-settlement secrecy or general multi-party privacy, both of which require additional techniques (§VII-C). Partial burning can deter a Garbler–solver coalition from recovering a colluding member's penalty, provided that the coalition's gain is bounded. The sufficient burn condition is given in Eq. (6).

Practicality and scope. The complete pre-signed transaction graph grows exponentially with the output width. Although BitBridge's binary settlement output keeps this graph small, its setup cost is dominated by garbling and committing the settlement verifier and scales with both the circuit count and the number of solvers. Designated-verifier circuits and recent garbling optimizations may reduce this overhead [74], [75], [76]. Appendix K-D examines these alternatives. The present construction targets stateless, single-round 2PC on unmodified Bitcoin, while general MPC requires additional techniques. Stateful applications can be supported by chaining a committed state root across successive protocol instances using mechanisms already present in BitPriv, in particular Lamport one-time signatures. Appendix K-E outlines this extension.

XI. Conclusion

We introduced BitPriv, to our knowledge the first protocol for enforcing arbitrary single-round 2PC over private inputs on unmodified Bitcoin. Its GHTLC primitive binds payouts to garbled-circuit output labels, while cut-and-choose, collateral, and BitVM disputes provide pre-settlement privacy and publicly verifiable covert security, with sufficient collateral deterring rational deviations under our economic and deployment assumptions. Optimistic settlement costs below $3 for both applications, with quantified setup, storage, and dispute costs across three enforcement backends. BitMarket and BitBridge exemplify

how BitPriv enables a new class of privacy-preserving DeFi applications on Bitcoin.

Acknowledgments. This work was partially supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement 101141432-BlockSec), by the Austrian Science Fund (FWF) through the SFB SpyCode project (F8510-N and F8512-N), by the Vienna Science and Technology Fund (WWTF) through the projects 10.47379/ICT25056 and 10.47379/ICT22045, and by the Austrian Research Promotion Agency (FFG) through the SBA-K1 NGC COMET center.

Ethical Considerations

The primary objective of BitPriv is to advance the fairness and security of decentralized finance on Bitcoin by reducing reliance on centralized intermediaries and mitigating Maximal Extractable Value (MEV) attacks, such as front-running in auctions. While improving privacy for legitimate financial applications is a net benefit for market integrity, we acknowledge the dual-use nature of such technologies. Increased privacy can theoretically appeal to illicit actors seeking to obscure transaction histories, and the complexity of DeFi protocols can pose financial risks to inexperienced users.

We argue that specific architectural choices in BitPriv mitigate these risks:

- **Pseudonymity, not Anonymity:** BitPriv does not introduce new anonymity primitives (e.g., mixers) beyond what the Bitcoin base layer provides. All collateral locking, disputes, and settlements remain publicly visible on the blockchain, preserving the ability of observers to trace flows of funds.
- **Economic Friction:** The protocol requires significant financial collateral to secure the 2PC execution. This high capital requirement aligns the protocol's utility with high-value, legitimate commercial applications rather than low-value, high-volume illicit transfers.
- **Transparency of Misbehavior:** Unlike trusted platforms where fraud may be hidden, BitPriv renders any deviation from the protocol publicly auditable via on-chain fraud proofs.

Furthermore, we emphasize that BitPriv is a protocol-level building block, not a consumer product. The security of user funds relies on the careful selection of system parameters (e.g., collateral amounts and timeout durations) by implementers. Poor parameterization could expose honest users to loss; therefore, deployments must prioritize conservative defaults and transparent user interfaces over aggressive optimization.

Finally, this research used purely synthetic data and simulations. It did not involve human subjects, user studies, or the analysis of personal financial data.

References

[1] DefiLlama, "Bridge hacks and total value locked," https://defillama.com/protocols/Bridge, 2026.
[2] L. Aumayr, Z. Avarikioti, R. Linus, M. Maffei, A. Pelosi, C. Stefo, and A. Zamyatin, "BitVM: Quasi-turing complete computation on bitcoin," Cryptology ePrint Archive, Paper 2024/1995, 2024.
[3] R. Linus, L. Aumayr, Z. Avarikioti, M. Maffei, A. Pelosi, O. T. Litos, C. Stefo, D. Tse, and A. Zamyatin, "Bridging bitcoin to second layers via BitVM2," Cryptology ePrint Archive, Paper 2025/1158, 2025.
[4] W. Chen, "SoK: BitVM with succinct on-chain cost," Cryptology ePrint Archive, Paper 2025/1253, 2025.
[5] R. L. Woll, I. Alexopoulos, L. Aumayr, Z. Avarikioti, M. Maffei, and D. Tse, "BitVM3: Efficient bitcoin bridges via garbled circuits," Cryptology ePrint Archive, Paper 2026/933, 2026.
[6] Bitlayer, "Bitlayer BitVM Bridge initiates mainnet beta," https://blog.bitlayer.org/_bl_product/en/2025-07-18-bitlayer_bitvm_bridge_mainnet/, 2025.
[7] ——, "Bitlayer secures $9 million in extended a round, led by Polychain Capital," https://blog.bitlayer.org/Bitlayer_Secures_%249_Million_in_Extended_A_Round_Led_by_Polychain_Capital/, 2024.
[8] Babylon Labs, "Babylon Labs and Aave Labs partner to bring native Bitcoin-backed lending to Aave V4," https://babylonlabs.io/blog/babylon-labs-and-aave-labs-partner-to-bring-native-bitcoin-backed-lending-to-aave-v4, 2025.
[9] ——, "Babylon and Aegis partner to bring fixed-rate borrowing to Bitcoin holders," https://babylonlabs.io/blog/babylon-and-aegis-partner-to-bring-fixed-rate-borrowing-to-bitcoin-holders, 2026.
[10] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels, "Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability," in *IEEE S&P*, 2020.
[11] A. E. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *IEEE S&P*, 2016.
[12] S. Bowe, A. Chiesa, M. Green, I. Miers, P. Mishra, and H. Wu, "ZEXE: enabling decentralized private computation," in *IEEE S&P*, 2020.
[13] A. L. Xiong, B. Chen, Z. Zhang, B. Bünz, B. Fisch, F. Krell, and P. Camacho, "Verizexe: Decentralized private computation with universal setup," in *USENIX Security*, 2023.
[14] C. Baum, J. H. Chiang, B. David, and T. K. Frederiksen, "Eagle: Efficient privacy preserving smart contracts," in *FC*, 2023.
[15] S. Steffen, B. Bichsel, and M. T. Vechev, "Zapper: Smart contracts with data and identity privacy," in *CCS*, 2022.
[16] S. Steffen, B. Bichsel, M. Gersbach, N. Melchior, P. Tsankov, and M. T. Vechev, "zkay: Specifying and enforcing data privacy in smart contracts," in *CCS*, 2019.
[17] A. Banerjee, M. Clear, and H. Tewari, "zkhawk: Practical private smart contracts from mpc-based hawk," in *BRAINS*, 2021.
[18] A. Banerjee and H. Tewari, "Multiverse of hawkness: A universally-composable mpc-based hawk variant," *Cryptogr.*, vol. 6, no. 3, p. 39, 2022.
[19] S. Garg, A. Goel, D. Kolonelos, and R. Sinha, "Jigsaw: Doubly private smart contracts," Cryptology ePrint Archive, Paper 2025/1147, 2025.
[20] R. Zhu, C. Ding, and Y. Huang, "Efficient publicly verifiable 2pc over a blockchain with applications to financially-secure computations," in *CCS*, 2019.
[21] F. Zhang, W. He, R. Cheng, J. Kos, N. Hynes, N. M. Johnson, A. Juels, A. Miller, and D. Song, "The ekiden platform for confidentiality-preserving, trustworthy, and performant smart contracts," *IEEE Secur. Priv.*, vol. 18, no. 3, pp. 17–27, 2020.
[22] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek, "Fair two-party computations via bitcoin deposits," in *FC Workshops*, 2014.
[23] ——, "Secure multiparty computations on bitcoin," in *IEEE S&P*, 2014.
[24] R. Kumaresan and I. Bentov, "How to use bitcoin to incentivize correct computations," in *CCS*, 2014.
[25] I. Bentov and R. Kumaresan, "How to use bitcoin to design fair protocols," in *CRYPTO*, 2014.
[26] R. Kumaresan, T. Moran, and I. Bentov, "How to use bitcoin to play decentralized poker," in *CCS*, 2015.
[27] R. Canetti, Y. Dodis, R. Pass, and S. Walfish, "Universally composable security with global setup," in *TCC*, 2007.
[28] A. Kiayias, H. Zhou, and V. Zikas, "Fair and robust multi-party computation using a global transaction ledger," in *EUROCRYPT*, 2016.
[29] R. Kumaresan and I. Bentov, "Amortizing secure computation with penalties," in *CCS*, 2016.
[30] I. Bentov, R. Kumaresan, and A. Miller, "Instantaneous decentralized poker," in *ASIACRYPT*, 2017.
[31] C. Baum, B. David, and R. Dowsley, "Insured MPC: efficient secure computation with financial penalties," in *FC*, 2020.
[32] H. A. Kalodner, S. Goldfeder, X. Chen, S. M. Weinberg, and E. W. Felten, "Arbitrum: Scalable, private smart contracts," in *USENIX Security*, 2018.
[33] R. Zhu, Y. Huang, J. Katz, and A. Shelat, "The Cut-and-Choose game and its application to cryptographic protocols," in *USENIX Security*, 2016.

[34] L. Lamport, "Constructing digital signatures from a one way function," SRI International, Tech. Rep. CSL-98, 1979.

[35] Y. Aumann and Y. Lindell, "Security against covert adversaries: Efficient protocols for realistic adversaries," *J. Cryptol.*, vol. 23, no. 2, pp. 281–343, 2010.

[36] G. Asharov and C. Orlandi, "Calling out cheaters: Covert security with public verifiability," in *ASIACRYPT*, 2012.

[37] Across. Across Documentation. [Online]. Available: https://docs.across.to/

[38] deBridge. deBridge. [Online]. Available: https://debridge.com/

[39] S. D. Lerner, R. Amela, S. Mishra, M. Jonas, and J. Álvarez Cid-Fuentes, "Bitvmx: A cpu for universal computation on bitcoin," arXiv:2405.06842, 2024.

[40] L. Eagen, "Glock: Garbled locks for bitcoin," Cryptology ePrint Archive, Paper 2025/1485, 2025.

[41] S. Garg, D. Kolonelos, M. Sergeevitch, S. Sridhar, and D. Tse, "BABE: Verifying proofs on bitcoin made 1000x cheaper," Cryptology ePrint Archive, Paper 2026/065, 2026.

[42] J. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," *J. ACM*, vol. 71, no. 4, 2024.

[43] A. C. Yao, "Protocols for secure computations (extended abstract)," in *FOCS*, 1982.

[44] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game or A completeness theorem for protocols with honest majority," in *STOC*, 1987.

[45] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free XOR gates and applications," in *ICALP*, 2008.

[46] S. Zahur, M. Rosulek, and D. Evans, "Two halves make a whole - reducing data transfer in garbled circuits using half gates," in *EUROCRYPT*, 2015.

[47] M. Naor, B. Pinkas, and R. Sumner, "Privacy preserving auctions and mechanism design," in *ACM EC*, 1999.

[48] D. Beaver, "Correlated pseudorandomness and the complexity of private computations," in *STOC*, 1996.

[49] T. Chou and C. Orlandi, "The simplest protocol for oblivious transfer," in *LATINCRYPT*, 2015.

[50] P. Wuille, J. Nick, and A. Towns, "Bip 341: Taproot: Segwit version 1 spending rules," Bitcoin Improvement Proposal, 2020. [Online]. Available: https://bips.dev/341/

[51] M. Bartoletti, S. Lande, and R. Zunino, "Bitcoin covenants unchained," *CoRR*, vol. abs/2006.03918, 2020.

[52] B. B. Elyakime, J.-J. Laffont, P. Loisel, and Q. Vuong, "First-price sealed-bid auctions with secret reservation prices," *Annales d'Economie et de Statistique*, no. 34, pp. 115–141, 1994, egalement paru dans : INRA-ESR Toulouse Série D ; 93-03D.

[53] W. Vickrey, "Counterspeculation, auctions, and competitive sealed tenders," *J. Finance*, vol. 16, no. 1, 1961.

[54] I. Bentov, Y. Ji, F. Zhang, L. Breidenbach, P. Daian, and A. Juels, "Tesseract: Real-time cryptocurrency exchange using trusted hardware," in *CCS*, 2019.

[55] V. Kolesnikov, A. Sadeghi, and T. Schneider, "Improved garbled circuit building blocks and applications to auctions and computing minima," Cryptology ePrint Archive, Paper 2009/411, 2009.

[56] C. Ganesh, B. Kanukurthi, and G. Shankar, "Secure auctions in the presence of rational adversaries," in *CCS*, 2022.

[57] S. A. K. Thyagarajan, G. Malavolta, and P. Moreno-Sanchez, "Universal atomic swaps: Secure exchange of coins across all blockchains," in *IEEE S&P*, 2022.

[58] E. N. Tas, I. A. Seres, Y. Zhang, M. Melczer, M. Kelkar, J. Bonneau, and V. Nikolaenko, "Atomic and fair data exchange via blockchain," in *CCS*, 2024.

[59] Optimism Foundation, "Rollup protocol overview," https://docs.optimism.io/stack/rollup/overview, 2025.

[60] X. Wang, A. J. Malozemoff, and J. Katz, "EMP-toolkit: Efficient MultiParty computation toolkit," https://github.com/emp-toolkit, 2016.

[61] BitVM, "Garbled snark verifier," https://github.com/BitVM/garbled-snark-verifier, 2025.

[62] H. Corrigan-Gibbs and D. Boneh, "Prio: Private, robust, and scalable computation of aggregate statistics," in *NSDI*, 2017.

[63] R. Canetti and R. Ostrovsky, "Secure computation with honest-looking parties: What if nobody is truly honest? (extended abstract)," in *STOC*, 1999.

[64] V. Goyal, P. Mohassel, and A. D. Smith, "Efficient two party and multi party computation against covert adversaries," in *EUROCRYPT*, 2008.

[65] V. Kolesnikov and A. J. Malozemoff, "Public verifiability in the covert model (almost) for free," in *ASIACRYPT*, 2015.

[66] C. Hong, J. Katz, V. Kolesnikov, W. Lu, and X. Wang, "Covert security with public verifiability: Faster, leaner, and simpler," Cryptology ePrint Archive, Paper 2018/1108, 2018.

[67] Y. Liu, J. Lai, Q. Wang, X. Qin, A. Yang, and J. Weng, "Robust publicly verifiable covert security: Limited information leakage and guaranteed correctness with low overhead," in *ASIACRYPT*, 2023.

[68] S. Faust, C. Hazay, D. Kretzler, and B. Schlosser, "Financially backed covert security," in *PKC*, 2022.

[69] S. Steffen, B. Bichsel, R. Baumgartner, and M. T. Vechev, "Zeestar: Private smart contracts by homomorphic encryption and zero-knowledge proofs," in *IEEE S&P*, 2022.

[70] R. Solomon, R. Weber, and G. Almashaqbeh, "smartfhe: Privacy-preserving smart contracts from fully homomorphic encryption," in *IEEE EuroS&P*, 2023.

[71] R. Kumaresan, V. Vaikuntanathan, and P. N. Vasudevan, "Improvements to secure computation with penalties," in *CCS*, 2016.

[72] F. Benhamouda, S. Halevi, and T. Halevi, "Supporting private data on hyperledger fabric with secure multiparty computation," *IBM J. Res. Dev.*, vol. 63, no. 2/3, pp. 3:1–3:8, 2019.

[73] B. David, R. Dowsley, and M. Larangeira, "Kaleidoscope: An efficient poker protocol with payment distribution and penalty enforcement," in *FC*, 2018.

[74] A. Futoransky, F. Barbàra, R. Fernandez, G. Larotonda, and S. D. Lerner, "TRAPGC-DV: Trapdoor garbled circuit designated verification of groth16 on bitcoin," Cryptology ePrint Archive, Paper 2026/1430, 2026.

[75] L. Eagen and Y. T. Lai, "Argo MAC: Garbling with elliptic curve MACs," Cryptology ePrint Archive, Paper 2026/049, 2026.

[76] N. Khambhati, A. Bhattacharya, and D. Heath, "Duty-free bits: Projectivizing garbling schemes," Cryptology ePrint Archive, Paper 2026/476, 2026.

[77] M. Bartoletti, R. Marchesin, and R. Zunino, "Secure compilation of rich smart contracts on poor UTXO blockchains," in *IEEE EuroS&P*, 2024.

[78] P. Das, L. Eckey, T. Frassetto, D. Gens, K. Hostáková, P. Jauernig, S. Faust, and A.-R. Sadeghi, "FastKitten: Practical smart contracts on bitcoin," in *USENIX Security*, 2019.

[79] T. Frassetto, P. Jauernig, D. Koisser, D. Kretzler, B. Schlosser, S. Faust, and A. Sadeghi, "POSE: practical off-chain smart contract execution," in *NDSS*, 2023.

[80] V. Madathil, S. A. K. Thyagarajan, D. Vasilopoulos, L. Fournier, G. Malavolta, and P. Moreno-Sanchez, "Cryptographic oracle-based conditional payments," in *NDSS*, 2023.

[81] R. Canetti, B. Riva, and G. N. Rothblum, "Refereed delegation of computation," *Inf. Comput.*, vol. 226, pp. 16–36, 2013.

[82] L. Aumayr, O. Ersoy, A. Erwig, S. Faust, K. Hostáková, M. Maffei, P. Moreno-Sanchez, and S. Riahi, "Generalized channels from limited blockchain scripts and adaptor signatures," in *ASIACRYPT*, 2021.

[83] M. Graf, D. Rausch, V. Ronge, C. Egger, R. Küsters, and D. Schröder, "A security framework for distributed ledgers," in *CCS*, 2021.

[84] C. Badertscher, U. Maurer, D. Tschudi, and V. Zikas, "Bitcoin as a transaction ledger: A composable treatment," in *CRYPTO*, 2017.

[85] J. Groth, "On the size of pairing-based non-interactive arguments," in *EUROCRYPT*, 2016.

[86] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending oblivious transfers efficiently," in *CRYPTO*, 2003.

[87] R. Cleve, "Limits on the security of coin flips when half the processors are faulty (extended abstract)," in *STOC*, 1986.

[88] K. Qin, L. Zhou, and A. Gervais, "Quantifying blockchain extractable value: How dark is the forest?" in *IEEE S&P*, 2022.

[89] E. Bal, L. Aumayr, A. İyidoğan, G. Scaffino, H. Karakuş, C. E. Aslan, and O. S. T. Litos, "Clementine: A collateral-efficient, trust-minimized, and scalable bitcoin bridge," Cryptology ePrint Archive, Paper 2025/776, 2025.

[90] R. Amela, S. Mishra, S. D. Lerner, and J. Álvarez Cid-Fuentes, "Union: A trust-minimized bridge for rootstock," arXiv:2501.07435, 2025.

[91] L. Aumayr, Z. Avarikioti, M. Maffei, G. Scaffino, and D. Zindros, "Blink: An optimal proof of proof-of-work," Cryptology ePrint Archive, Paper 2024/692, 2024.

[92] B. Bünz, L. Kiffer, L. Luu, and M. Zamani, "Flyclient: Super-light clients for cryptocurrencies," in *IEEE S&P*, 2020.

[93] P. Gazi, A. Kiayias, and D. Zindros, "Proof-of-stake sidechains," in *IEEE S&P*, 2019.

[94] A. Kiayias, A. Miller, and D. Zindros, "Non-interactive proofs of proof-of-work," in *FC*, 2020.
[95] C. Rodarmor and contributors, *Ordinal Theory Handbook*, 2023. [Online]. Available: https://docs.ordinals.com/
[96] C. Goes, "The interblockchain communication protocol: An overview," arXiv:2006.15918, 2020.
[97] mempool.space, "Block fee rates," https://mempool.space/graphs/mining/block-fee-rates#1y, 2026.
[98] GOAT Network, "bitvm2-gc: Bitvm with verifiable garbled circuit optimized," https://github.com/GOATNetwork/bitvm-gc, 2026.

Appendix A
BitVM

Bitcoin's intentionally non-Turing-complete Script precludes direct on-chain verification of arbitrary computations. Proposed workarounds [51], [77], [78], [79], [80] either require consensus changes, incur significant on-chain costs, or introduce additional trust assumptions. BitVM [2] instead executes computations off-chain while retaining on-chain enforceability through pre-signed dispute transactions between a *prover* P and a *verifier* V. The parties agree on a program Π and a fund-allocation function mapping its outputs to spending conditions, after which P commits to an output.

BitVM1 [2] resolves a dispute through binary search over the $|C|$ execution steps, isolating a single contested operation for evaluation in Bitcoin Script with $O(\log |C|)$ on-chain transactions [81]. BitVMX [39] applies this approach to arbitrary RISC-V programs through an off-chain virtual CPU. BitVM2 [3] instead commits intermediate verifier states, allowing a dispute to isolate and re-execute one program chunk in Bitcoin Script.

BitVM3-style designs [5], [40] move dispute verification off-chain using garbled circuits, allowing a challenger to evaluate the verifier locally and submit compact evidence of an invalid claim. BABE [41] combines garbling-based cryptographic primitives with witness encryption to reduce off-chain costs. Although these designs employ garbled circuits internally, they do not provide privacy for application inputs.

These variants offer different setup, storage, interaction, and on-chain costs while providing the same enforcement abstraction used by BitPriv. Any backend capable of verifying the misbehavior predicate Φ_{mis} can therefore instantiate BitPriv's dispute layer.

Appendix B
Formal Model

We now present the formal model. We first abstract the transaction ledger, BitVM, and oblivious transfer (OT) as ideal functionalities. We then define the adversarial model using publicly verifiable covert (PVC) security and formalize security for BitPriv via an ideal functionality that captures the protocol goals. The network model and cryptographic assumptions are as in Section II.

A. Transaction Ledger Model

a) Transaction Structure: We adopt the notation of [82]. Formally, an output θ is a pair (cash, φ), where cash denotes the coin amount, and φ is a script (spending condition) that must be satisfied to unlock the output. A transaction consumes existing outputs and creates new ones. Inputs correspond to references to previously unspent outputs. Formally, a transaction tx is a tuple:

$$\mathsf{tx} := (\mathsf{txid}, \mathsf{In}, \mathsf{Out}, \mathsf{Witness}),$$

where $\mathsf{txid} \in \{0,1\}^*$ is a unique identifier, computed as $\mathsf{txid} := \mathcal{H}([\mathsf{tx}])$, $\mathcal{H}$ is a hash function modeled as a random oracle, $[\mathsf{tx}] := (\mathsf{In}, \mathsf{Out})$ is the transaction body, In is a vector of identifiers for previously unspent outputs, $\mathsf{Out} = (\theta_1, \ldots, \theta_n)$ is a list of new outputs, and $\mathsf{Witness} \in \{0,1\}^*$ contains the unlocking data required to satisfy each input's script.

TABLE III: A transaction in our notation. The input spends output 0 of txid_1, locked under $\texttt{CheckSig}_{\mathsf{pk}_A}$, using A's signature σ_A. The output is a Taproot UTXO with two spending paths: A alone can spend by revealing a preimage of h, or A and B can co-sign after a relative delay of 10 blocks.

Example Transaction	
Inputs	(0) $(\mathsf{txid}_1,\ 0,\ \texttt{CheckSig}_{\mathsf{pk}_A})$
Outputs	(0) $\Big(5\,\text{BTC},\ \big\langle \texttt{CheckSig}_{\mathsf{pk}_A} \wedge \text{Hashlock}(h),\ \text{RelTimelock}(10) \wedge \texttt{CheckMultiSig}_{\mathsf{pk}_{AB}} \big\rangle\Big)$
Witness	(0) σ_A

Bitcoin Script supports standard conditions including signature checks ($\texttt{CheckSig}_{\mathsf{pk}}$), multisignature checks ($\texttt{CheckMultiSig}_{\mathsf{pk}_{1\ldots n}}$), absolute and relative timelocks ($\text{AbsTimelock}(t)$, $\text{RelTimelock}(t)$), and hash-preimage locks ($\text{Hashlock}(h)$). We write pk_{AB} as shorthand for the key set $\{\mathsf{pk}_A, \mathsf{pk}_B\}$. Taproot [50] allows a UTXO to commit to multiple alternative spending conditions arranged in a Merkle tree. To spend a Taproot output, the spender reveals only the executed script leaf, its satisfying witness, and a Merkle proof of inclusion. We denote a Taproot output by $\langle \cdot \rangle$, and illustrate the notation in Table III.

b) Ledger Ideal Functionality $\mathcal{L}$: We slightly adapt the ledger functionality $\mathcal{L}$ from [82], which captures the money mechanics of UTXO-based cryptocurrencies. The functionality is parameterized by a liveness parameter u, bounding the number of rounds for a transaction to be confirmed by the ledger, a digital signature scheme Σ, and a set V of valid output conditions, including signature checks with respect to Σ. After initialization, the state of $\mathcal{L}$ is public and can be accessed by all parties, using a read message. At any point, a party $P \in \mathcal{P}$ may post a transaction tx by sending $(\mathsf{post}, \mathsf{tx})$ to the ledger. The ledger waits up to u rounds (determined by $\mathcal{A}$), validates the transaction and appends it to TX. This abstraction omits block structure and dynamic participation, which are treated in detail in prior works [83], [84], but suffices to capture the properties needed for our analysis.

Ideal Functionality $\mathcal{L}(u, \Sigma, \mathcal{V})$

The functionality accepts messages from all parties in the set $\mathcal{P}$ and maintains a PKI *for those parties, the set* TX *of all accepted transactions, and the set* UTXO *of all unspent transaction outputs.*

Initialize public keys: Upon $(\texttt{REGISTER},\ pk_P) \overset{\tau_0}{\hookleftarrow} P$ and it is the first time P sends a registration message, add (pk_P, P) to

PKI.

Post transaction: Upon $(\texttt{POST}, tx) \xleftarrow{\tau_0} P$, check that $|\mathsf{PKI}| = |\mathcal{P}|$. If not, drop the message, else wait until round $\tau_1 \leq \tau_0 + u$ (the exact value of τ_1 is determined by the adversary). Then check if:

1) The id is unique, i.e., for all $(t, \mathrm{tx}') \in \mathsf{TX}$, $\mathrm{tx}'.\texttt{txid} \neq \texttt{tx.txid}$.
2) All the inputs are unspent and the witness satisfies all the output conditions, i.e., for each $(\texttt{tid}, i) \in \texttt{tx.In}$, there exists $(t, \texttt{tid}, i, \theta) \in \mathsf{UTXO}$ and $\theta.\varphi(\texttt{tx}, t, \tau_1) = 1$.
3) All outputs are valid, i.e., for each $\theta \in \texttt{tx.Out}$ it holds that $\theta.\texttt{cash} > 0$ and $\theta.\varphi \in \mathcal{V}$.
4) The value of the outputs is not larger than the value of the inputs. More formally, let $I := \{\texttt{utxo} := (t, \texttt{tid}, i, \theta) \mid \texttt{utxo} \in \mathsf{UTXO} \wedge (\texttt{tid}, i) \in \texttt{tx.In}\}$, then $\sum_{\theta' \in \texttt{tx.Out}} \theta'.\texttt{cash} \leq \sum_{\texttt{utxo} \in I} \texttt{utxo}.\theta.\texttt{cash}$.
5) The absolute time-lock of the transaction has expired, i.e., $\texttt{tx.TimeLock} \leq \mathsf{now}$.

If all the above checks return true, add $(\tau_1, \texttt{tx})$ to TX, remove the spent outputs from UTXO, i.e., $\mathsf{UTXO} := \mathsf{UTXO} \setminus I$, and add the outputs of $\texttt{tx}$ to UTXO, i.e., $\mathsf{UTXO} := \mathsf{UTXO} \cup \{(\tau_1, \texttt{tx.txid}, i, \theta_i)\}_{i \in [n]}$ for $(\theta_1, \ldots, \theta_n) := \texttt{tx.Out}$. Else, ignore the message.

Read state: Upon $(\texttt{READ}) \xleftarrow{\tau_0} X$, where X is any entity of the system, check that $|\mathsf{PKI}| = |\mathcal{P}|$. If not, drop the message, else $(\mathsf{state}, \mathsf{PKI}, \mathsf{TX}) \xhookrightarrow{\tau_0} X$.

B. Ideal Functionality $\mathcal{F}_{BitVM}$

In our analysis, we abstract BitVM and its variants into a single ideal functionality, denoted by $\mathcal{F}_{\text{BitVM}}$, parameterized by a dispute bound T_{BitVM} representing the maximum number of steps or rounds allowed during dispute resolution. Conceptually, $\mathcal{F}_{\text{BitVM}}$ captures an optimistic proof system between a prover and a verifier for checking the correctness of an off-chain computation.

This abstraction captures the key security properties:

- **Correctness:** honest claims about a computation are always accepted.
- **Soundness:** invalid claims are rejected, and cheating provers are detected within T_{BitVM} rounds, except with negligible probability.
- **Financial enforcement.** Collateral is redistributed on-chain so that honest parties receive the correct payout, while dishonest parties forfeit their deposits as penalty.

Financial enforcement in $\mathcal{F}_{\text{BitVM}}$ is the mechanism that underlies our protocol's financial fairness guarantee (see Section II).

Ideal Functionality $\mathcal{F}_{\text{BitVM}}^{\mathcal{L}(u,\Sigma,\mathcal{V})}$

Let P_A assume the role of the prover and P_B the verifier (in BitPriv*, the Evaluator acts as prover and the Garbler as verifier) and v be the verification function i.e., $v(w, o) = 1$ iff $\mathsf{p}.f(w) = o$*

Setup Phase

Agree on Parameters: Upon $(\texttt{SETUP}, p) \hookleftarrow P_A$ and $(\texttt{SETUP}, p') \hookleftarrow P_B$, or $(\texttt{ABORT}) \hookleftarrow$ from either:

- If $p = p'$ and $(pk_A, P_A), (pk_B, P_B) \in \texttt{PKI}$: send $(\texttt{INIT}(p, \mathsf{setup})) \xhookrightarrow{\tau_0} (P_A, P_B)$.
- Else: $(\texttt{ABORTED}) \xhookrightarrow{\tau_0} (P_A, P_B)$ and halt.

Active Phase

Upon $(\texttt{START}) \xleftarrow{\tau_0} P_A \vee P_B$:

- read-ledger and verify if $tx \in \texttt{TX}$ with $tx.id = \mathsf{setup}.id$, otherwise ignore.
- $(\texttt{INIT}) \xhookrightarrow{\tau_0} (P_A, P_B)$

1) Upon $(\texttt{CLAIM}, \mathsf{w}, \mathsf{o}) \xleftarrow{\tau_1 \leq \tau_0 + \Delta} P_A$:
   - For every round until $\mathsf{p}.T_{BitVM}$ call read-ledger:
     - if $v(w, o) = 1$: Let $(c_A, c_B) \leftarrow p.g(o)$, verify if $tx \in \texttt{TX}$ with output $tx.Out = \big((c_A, \mathsf{One\text{-}Sig}_{Pk_A}), (c_B, \mathsf{One\text{-}Sig}_{Pk_B})\big)$ or $tx.Out = \big((0, \mathsf{One\text{-}Sig}_{Pk_A}), (\mathsf{setup.all}, \mathsf{One\text{-}Sig}_{Pk_B})\big)$; if none exists, output error and halt.
     - if $v(w, o) \neq 1$: verify if $tx \in \texttt{TX}$ with $tx.Out = \big((0, \mathsf{One\text{-}Sig}_{Pk_A}), (\mathsf{setup.all}, \mathsf{One\text{-}Sig}_{Pk_B})\big)$; if none exists, output error and halt
2) Idle A (no $(\texttt{CLAIM}, \texttt{w}, \texttt{o})$ within Δ):
   - $(\texttt{ABORTED-A}) \xhookrightarrow{\tau_2 = \tau_1 + \Delta} P_B$
3) Upon $(\texttt{PUNISH-A}) \xleftarrow{\tau_3 \leq \tau_2 + u}$ from P_B:
   - If $(\texttt{CLAIM}) \xleftarrow{\tau_3' \leq \tau_3}$: handle as above.
   - Else:
     - $(\texttt{PUNISH-ABORT}) \xhookrightarrow{\tau_3} (P_A, P_B)$
     - call read-ledger and verify if $tx \in \texttt{TX}$ with $tx.Out = \big((0, \mathsf{One\text{-}Sig}_{Pk_A}), (\mathsf{setup.all}, \mathsf{One\text{-}Sig}_{Pk_B})\big)$; if none exists, output error and halt.

read-ledger: $(\texttt{READ}) \xhookrightarrow{\tau} \mathcal{L}$; return $(\texttt{state}, \texttt{PKI}, \texttt{TX}) \xhookleftarrow{\tau} \mathcal{L}$.

The functionality is parameterized by a tuple p agreed upon by the parties that encapsulates the logic of the protocol. This tuple specifies: (i) the program f, modeled as a fully unrolled function with an a priori bounded number of execution steps; (ii) a verification function v which, on input (w, o), returns 1 iff $f(w) = o$; and (iii) an allocation policy g mapping outcomes to payouts. Under this model, an honest prover submitting a valid pair (w, o) always receives at least its collateral share c_A: exactly c_A if the verifier is honest, and setup.all otherwise. The same guarantee holds symmetrically for the verifier.

This abstraction omits low-level protocol details (e.g., transcript encoding, step-by-step challenges), while retaining the security-critical guarantees. Error events, such as ledger faults or cryptographic breaks (e.g., signature forgery), occur only with negligible probability in our model.

C. Ideal Functionality $\mathcal{F}_{OT}$

We recall the standard ideal functionality for Oblivious Transfer $\mathcal{F}_{OT}$, used as a building block in our construction (e.g., for input selection in garbled circuits) and assumed in our security proof. In $\mathcal{F}_{OT}$, a sender provides pairs of inputs, and a receiver with choice bits learns exactly one input from each pair, while the sender learns nothing about the receiver's choices.

Ideal Functionality $\mathcal{F}_{\text{OT}}$

- Private inputs: P_A has input $\{(B_i^0, B_i^1)\}_{i=1}^n$ and P_B has input $y \in \{0,1\}^n$.
- Upon receiving $\{(B_i^0, B_i^1)\}_{i=1}^n$ from P_A and y from P_B, send $\{B_i^{y[i]}\}_{i=1}^n$ to P_B.

D. Threat Model

We analyze security in a two-party setting with static corruption. Adversaries are probabilistic polynomial-time bounded but may deviate arbitrarily from the protocol. We distinguish:

- *Malicious adversaries*, who may behave arbitrarily.
- *Covert adversaries*, as introduced in [35], [36], who deviate actively only if they believe the chance of detection is sufficiently low. This captures rational behavior where parties are deterred from cheating by the risk of being exposed and penalized.

This covert model bridges the gap between semi-honest and fully malicious security and reflects practical settings where financial penalties enforce compliance. In our protocol, the Garbler is modeled as covert and the Evaluator as potentially malicious.

E. Security Definitions

We formalize the security of BitPriv via an ideal functionality $\mathcal{F}_{\text{PCDC}}$, which the real protocol aims to realize. We first present the notion of *publicly verifiable covert security*, which captures adversaries that may attempt to cheat but are deterred by the risk of detection. This notion serves as a stepping stone toward proving that BitPriv achieves the security goals outlined in Section II.

Definition 1 (Publicly Verifiable Covert Security)**.** *A two-party protocol satisfies* publicly verifiable covert (PVC) security *with deterrence parameter ϵ if it provides the following guarantees:*

1) ***Covert Security.*** *Following the explicit-cheat formulation [35], any cheating attempt is detected with probability at least $\epsilon = 1 - \frac{1}{n}$, where n is the number of circuits.*
2) ***Public Verifiability.*** *If cheating is detected, the honest party can generate a publicly verifiable certificate of misbehavior (i.e., a witness such that $\Phi_{\text{mis}}(w, o) = 1$), except with negligible probability.*
3) ***Defamation-Freeness.*** *No malicious party can forge such a certificate against an honest party, except with negligible probability.*

Ideal Functionality $\mathcal{F}_{\text{PCDC}}^{\mathcal{L}(u,\Sigma,\mathcal{V})}$

Setup Phase

Register to Ledger: Upon $(\texttt{REGISTER}, pk_{P_k}) \overset{\tau}{\hookleftarrow} P_k$ for $k \in \{A, B\}$, forward to functionality $\mathcal{L}$ to update PKI.

Agree on Parameters: Upon $(\texttt{SETUP}, p, att) \overset{\tau}{\hookleftarrow} P_A$ and $(\texttt{SETUP}, p', [y]) \overset{\tau}{\hookleftarrow} P_B$, or $(\texttt{ABORT}) \overset{\tau}{\hookleftarrow}$ from either:

- If $p = p'$ and $(pk_A, P_A), (pk_B, P_B) \in \texttt{PKI}$: send $\texttt{INIT}(p) \overset{\tau_0}{\hookrightarrow} (P_A, P_B)$.
- Else: send $\texttt{ABORTED} \overset{\tau_0}{\hookrightarrow} (P_A, P_B)$ and halt.

Corruption Setting:

- If att = cheat(z'):
  - $corruptedA := 1$
  - With probability ϵ: $detectedA := 1$
  - With probability $1 - \epsilon$: $undetectedA := 1$
- If att = blatant-cheat: $corruptedA := 1$, $detectedA := 1$
- If $att = \perp$: $corruptedA := 0$

Active Phase

Start: If no (START) received by $\tau = \tau_0 + u$, send $(\texttt{ERROR}) \overset{\tau}{\hookrightarrow} P_A, P_B$ and halt.

Upon $(\texttt{START}) \overset{\tau_1 \le \tau_0 + u}{\longleftarrow}$ from $\{P_A, P_B\}$:

Optimistic Path

Distinguish between the following cases:

- **Both parties honest.**

  Upon $(\texttt{OPT-REQUEST}, m) \overset{\widehat{\tau}}{\hookleftarrow} P_B$ and $(\texttt{OPT-REVEAL}, m, f, x) \overset{\widehat{\tau}'}{\hookleftarrow} P_A$, send $(\texttt{OPT-CLOSE}, m, f(x, y)) \overset{\widehat{\tau}''}{\hookrightarrow} P_B$
- **P_A malicious (undetected) and P_B honest.**

  Upon $(\texttt{OPT-REQUEST}, m) \overset{\widehat{\tau}}{\hookleftarrow} P_B$: If received $(\texttt{OPT-REVEAL-UNDETECTED}, m, f', x') \overset{\widehat{\tau}'}{\hookleftarrow} P_A$ with $z' = f'(x', y) \neq \perp$, send $(\texttt{OPT-CLOSE}, m, z') \overset{\widehat{\tau}''}{\hookrightarrow} P_B$. Otherwise, if $z' = \perp$ or no message is received from P_A, send $(\texttt{OPT-CLOSE-FAIL}) \overset{\widehat{\tau}''}{\hookrightarrow} P_B$.
- **P_A malicious (detected) and P_B honest.**

  Upon $(\texttt{OPT-REQUEST}, m) \overset{\widehat{\tau}}{\hookleftarrow} P_B$: If received $(\texttt{OPT-REVEAL-DETECTED}, m) \overset{\widehat{\tau}'}{\hookleftarrow} P_A$, send $(\texttt{OPT-DETECTED}, m, \texttt{witness}) \overset{\widehat{\tau}''}{\hookrightarrow} P_B$. Otherwise, if no message is received from P_A, send $(\texttt{OPT-CLOSE-FAIL}) \overset{\widehat{\tau}''}{\hookrightarrow} P_B$.
- **P_A honest and P_B malicious.**

  If received $(\texttt{OPT-REQUEST}, m) \overset{\widehat{\tau}}{\hookleftarrow} P_B$ continue as in the case P_B is honest, otherwise send $(\texttt{OPT-UNRESPONSIVE}) \overset{\widehat{\tau}''}{\hookrightarrow} P_A$.

Wait for P_B ($clock = 0$): and distinguish between the following cases:

1) If $(\texttt{OPT-OUT}, j, k) \overset{\tau_2 \le \tau_1 + \Delta}{\longleftarrow} P_B$ for some $j \in [n]$ and $k \in$ Out(j):
   a) **P_B honest and P_A honest or detected.** If previously sent $(\texttt{OPT-CLOSE}, j, k)$ in the optimistic path, Let $(c_A, c_B) \leftarrow g(k)$, wait for Δ to call read-ledger and verify if $tx \in \texttt{TX}$ with output $\big((c_A, \text{One-Sig}_{Pk_A}), (c_B, \text{One-Sig}_{Pk_B})\big)$; if none exists, output error and halt.
   b) **P_B malicious P_A honest.** If previously sent $(\texttt{OPT-CLOSE}, j', k')$ in the optimistic path with $j' \neq j$, wait for u call read-ledger and verify if $tx \in \texttt{TX}$ with $tx.Out = (\text{all}, \text{One-Sig}_{Pk_A})$; if none exists, output error and halt.
2) If $(\texttt{REQUEST}, m) \overset{\tau_2 \le \tau_1 + \Delta}{\longleftarrow}$ from P_B:
   - If $detectedA$: $(\texttt{REVEAL-DETECTED}, m) \overset{\tau_2}{\hookrightarrow} P_A$
   - If $undetectedA$: $(\texttt{REVEAL-UNDETECTED}, m) \overset{\tau_2}{\hookrightarrow} P_A$
   - Else: $(\texttt{REVEAL}, m) \overset{\tau_2}{\hookrightarrow} P_A$
3) Idle B (no $(\texttt{REQUEST}, j)$ within Δ for some $j \in [n]$):
   - $corruptedB := 1$
   - $(\texttt{ABORTED-B}) \overset{\tau_2' = \tau_1 + \Delta}{\longrightarrow} P_A$
4) Upon $(\texttt{PUNISH-B}) \overset{\tau_3 \le \tau_2' + u}{\longleftarrow} P_A$:
   - If $(\texttt{REQUEST}) \overset{\tau_3' \le \tau_3}{\longleftarrow} P_B$: handle as above.
   - Else:
     - $(\texttt{PUNISH-ABORT}) \overset{\tau_3}{\hookrightarrow} (P_A, P_B)$
     - read-ledger and verify if $tx \in \texttt{TX}$ with $tx.Out = \big((\text{all}, \text{One-Sig}_{Pk_A}), (0, \text{One-Sig}_{Pk_B})\big)$; if none exists, $(\texttt{ERROR}) \overset{\tau_3}{\hookrightarrow} (P_A, P_B)$ and halt.

Wait for P_A ($clock = 0$): Let $\tau' := \tau_2 \vee \tau_3'$ be the time the `(REVEAL)` *message was sent to P_A*

1) If $(\texttt{(REVEAL-A)}, z') \xleftarrow{\tau_4 \le \tau' + \Delta} P_A$ and ($detectedA \vee undetectedA$):
   - If $detectedA$: $(\texttt{(DETECTED-A)}, \text{witness}) \xhookrightarrow{\tau_4} P_B$
   - If $undetectedA$: $(\texttt{(OUTPUT)}, z') \xhookrightarrow{\tau_4} P_B$
2) Else if $(\texttt{(REVEAL-A)}, f, x) \xleftarrow{\tau_4 \le \tau' + \Delta}$ and $\neg(detectedA \vee undetectedA)$:
   - $z = f(x, y)$
   - $(\texttt{(OUTPUT)}, z) \xhookrightarrow{\tau_4} P_B$
3) Idle A (no `(REVEAL)` within Δ):
   - `(ABORTED-A)` $\xhookrightarrow{\tau_5 = \tau_4 + \Delta} P_B$
4) Upon `(PUNISH-A)` $\xleftarrow{\tau_6 \le \tau_5 + u}$ from P_B:
   - If `(REVEAL-A)` $\xleftarrow{\tau_6' \le \tau_6}$: handle as above.
   - Else:
     - `(PUNISH-ABORT)` $\xhookrightarrow{\tau_6} (P_A, P_B)$
     - read-ledger and verify if $tx \in$ `TX` with $tx.Out = \big((0, \mathsf{One\text{-}Sig}_{Pk_A}), (\mathsf{all}, \mathsf{One\text{-}Sig}_{Pk_B})\big)$; if none exists, `(ERROR)` $\xhookrightarrow{\tau_6} (P_A, P_B)$ and halt.

Wait for P_B ($clock = 0$:) Let $\tau'' := \tau_4 \vee \tau_6'$ be the time the `OUTPUT` *message was sent to P_B*

1) If `(PUNISH-CHEAT)` $\xleftarrow{\tau_7 \le \tau'' + \Delta} P_B$: for every round until T_{BitVM} call read-ledger:
   - If $detectedA$: verify if $tx \in$ `TX` with $tx.Out = \big((0, \bot), (\mathsf{all}, \mathsf{One\text{-}Sig}_{Pk_B})\big)$; if none exists, `(ERROR)` $\xhookrightarrow{\tau_6}$ (P_A, P_B) and halt. Otherwise: `(FINISH)` $\xhookrightarrow{\tau_6} (P_A, P_B)$ and halt.
   - If $\neg detectedA$: verify if $tx \in$ `TX` with $tx.Out = \big((\mathsf{all}, \mathsf{One\text{-}Sig}_{Pk_A}), (0, \bot)\big)$; if none exists, `(ERROR)` $\xhookrightarrow{\tau_6}$ (P_A, P_B) and halt. Otherwise: `(FINISH)` $\xhookrightarrow{\tau_6} (P_A, P_B)$ and halt.
2) If $(\texttt{(CLOSE,}z')$ $\xleftarrow{\tau_7 \le \tau'' + \Delta} P_B$:
   - If $undetectedA$:
     $(\texttt{(OUTPUT-UNDETECTED,}\ z')$ $\xhookrightarrow{\tau_7} P_A$. Let $(c_A, c_B) \leftarrow g(z')$, wait for Δ rounds to call read-ledger and verify if $tx \in$ `TX` with output $\big((c_A, \mathsf{One\text{-}Sig}_{Pk_A}), (c_B, \mathsf{One\text{-}Sig}_{Pk_B})\big)$; if none exists, output error and halt. Otherwise send `(FINISH)` to both parties and halt.
   - If P_A honest: $z' = f(x, y)$, send $(\texttt{(OUTPUT,}\ f(x, y))$ $\xhookrightarrow{\tau_7} P_A$. Let $(c_A, c_B) \leftarrow g(f(x, y))$, wait for Δ rounds to call read-ledger and verify if $tx \in$ `TX` with output $\big((c_A, \mathsf{One\text{-}Sig}_{Pk_A}), (c_B, \mathsf{One\text{-}Sig}_{Pk_B})\big)$; if none exists, output error and halt. Otherwise send `(FINISH)` to both parties and halt.
3) If $(\texttt{(CLOSE-EQUIV,}z'')$ $\xleftarrow{\tau_7 \le \tau'' + \Delta} P_B$: send `(EQUIV-ATTEMPT)` $\xhookrightarrow{\tau_7} P_A$
   - If `(PUNISH-EQUIV)` $\xleftarrow{\tau_{10} \le \tau_7 + \Delta}$, call read-ledger and verify if $tx \in$ `TX` with $tx.Out = (\mathsf{all}, \mathsf{One\text{-}Sig}_{Pk_A})$; if none exists, output error and halt. Otherwise, send `(EQUIV-DETECTED)` to P_B
   - Otherwise `(IDLE-EQUIV)` $\xhookrightarrow{\tau_{11} = \tau_7 + \Delta}$

   Upon `(CLAIM-EQUIV)` $\xleftarrow{\tau_{12} \le \tau_{11} + u}$:
   - send $(\texttt{(EQUIV-UNDETECTED,}\ z'')$ $\xhookrightarrow{\tau_{12}} P_A$. Let $(c_A, c_B) \leftarrow g(z'')$, call read-ledger and verify if $tx \in$ `TX` with output $\big((c_A, \mathsf{One\text{-}Sig}_{Pk_A}), (c_B, \mathsf{One\text{-}Sig}_{Pk_B})\big)$; if none exists, output error and halt.
4) Idle B (no `(PUNISH-CHEAT)` or $(\texttt{(CLOSE,}z')$ within Δ):
   - `(ABORTED-B)` $\xhookrightarrow{\tau_8 = \tau'' + \Delta} P_A$.

   Upon `(PUNISH-B)` $\xleftarrow{\tau_9 \le \tau_8 + u} P_A$:
   - If `(PUNISH-CHEAT)` $\vee$ `(CLOSE)` $\xleftarrow{\tau_9' \le \tau_9} P_B$: handle as above.
   - Else:
     - `(PUNISH-ABORT)` $\xhookrightarrow{\tau_9} (P_A, P_B)$
     - read-ledger and verify if $tx \in$ `TX` with $tx.Out = \big((\mathsf{all}, \mathsf{One\text{-}Sig}_{Pk_A}), (0, \mathsf{One\text{-}Sig}_{Pk_B})\big)$; if none exists, `(ERROR)` $\xhookrightarrow{\tau_9} (P_A, P_B)$ and halt.

read-ledger: send `(READ)` $\xhookrightarrow{\tau} \mathcal{L}$ and return the reply `(state, PKI, TX)` received from $\mathcal{L}$.

Appendix C
Cryptographic Primitives

A. Succinct Non-Interactive Arguments of Knowledge (SNARKs)

Our definition closely follows [85]. Let $\mathcal{R} \leftarrow \hat{\mathcal{R}}(\lambda)$ denote a relation generator that, on input a security parameter λ, outputs a polynomial-time decidable binary relation $\mathcal{R}$. We write ϕ for the statement and w for the witness, with $(\phi, w) \in \mathcal{R}$.

An efficient publicly verifiable non-interactive argument of knowledge for $\hat{\mathcal{R}}$ consists of three PPT algorithms: Setup, Prove, and Vrfy.

- $crs \leftarrow \mathsf{SNARK.Setup}(\mathcal{R})$ takes the relation $\mathcal{R}$ as input and outputs a common reference string crs.
- $\pi \leftarrow \mathsf{SNARK.Prove}(\mathcal{R}, crs, \phi, w)$ takes as input the common reference string crs and a pair $(\phi, w) \in \mathcal{R}$, and outputs an argument π.
- $\{\mathsf{True}, \mathsf{False}\} \leftarrow \mathsf{SNARK.Vrfy}(\mathcal{R}, crs, \phi, \pi)$ takes as input the common reference string crs, a statement ϕ, and an argument π, and returns True or False depending on whether π is a valid argument.

The tuple (Setup, Prove, Vrfy) is a non-interactive argument of knowledge for $\hat{\mathcal{R}}$ if it satisfies *perfect completeness* and *knowledge soundness*, as defined in [85]. Informally, perfect completeness guarantees that an honest prover convinces an honest verifier of any true statement ϕ with overwhelming probability. Knowledge soundness guarantees that any prover who convinces an honest verifier must know a valid witness w, except with negligible probability.

A non-interactive argument is a preprocessing succinct non-interactive argument of knowledge, or SNARK, if the verifier runs in time polynomial in $\lambda + |\phi|$ and the proof size is polynomial in λ. Our implementation uses [85].

B. Garbled Circuits

A Garbled Circuit (GC) [43], [44] is a cryptographic protocol that allows two parties, a circuit generator (G) and a circuit evaluator (E), to jointly compute a function without revealing their private inputs beyond the output itself. In particular, the generator creates an encrypted version of the Boolean circuit representation of the target function, which the evaluator can process to obtain the final result.

We adopt the definition of a secure garbling scheme $\mathcal{G}$ from [20] as a tuple of efficient, deterministic algorithms

(Gb, Ev, En, De, Ve), which emphasizes the use of seeds for reproducibility and verifiability.

Let f be the function to be computed, κ the security parameter, and x the input. The algorithms are defined as follows:

- $(F, e, d) \leftarrow \mathsf{Gb}(1^\kappa, f, \text{Seed})$: A deterministic *garbler* that takes the function f and a seed to produce a garbled circuit F, encoding information e, and decoding information d.
- $X \leftarrow \mathsf{En}(e, x)$: The *encoder* that transforms a plaintext input x into its corresponding encoded representation, i.e., *labels* X.
- $Y \leftarrow \mathsf{Ev}(F, X)$: The *evaluator* that runs on the garbled circuit F and the encoded input X to produce the encoded output Y.
- $y \leftarrow \mathsf{De}(d, Y)$: The *decoder* that converts the encoded output Y back to the plain text result y.
- $b \leftarrow \mathsf{Ve}(f, \text{Seed}, F, d)$: A *verifier* that outputs a bit indicating whether the circuit F and decoding information d were correctly generated from the function f and the specified seed.

A secure garbling scheme must satisfy several properties. **Correctness** ensures that an honestly generated circuit evaluates to the correct output. **Privacy** guarantees that the garbled circuit and encoded inputs can be simulated knowing only the function's final output, thus protecting the inputs. **Obliviousness** ensures that the circuit and encoded inputs (without decoding information) do not reveal the output value. *Authenticity* guarantees that the output was correctly computed from the specified circuit and inputs, preventing tampering by a malicious evaluator. Finally, **Verifiability** guarantees that the *verifier* is correct except with negligible probability.

In a two-party setting, the generator runs Gb and sends the garbled circuit F to the evaluator. The generator also uses En to encode its own inputs and sends the corresponding wire-labels to the evaluator. To securely transfer the evaluator's input labels without revealing the evaluator's input choice to the generator, the parties engage in an Oblivious Transfer (OT) protocol.

C. Oblivious Transfer

OT [48], [49] is a fundamental cryptographic primitive that allows a receiver to choose one of two messages from a sender without the sender learning which message was chosen. While OT can be built from various public-key hardness assumptions, these constructions are computationally expensive. To improve efficiency, *OT-extension* [86] protocols are used to amortize the cost of a few public-key operations over many OT instances.

Once the evaluator has obtained all input wire-labels (those of the garbler via direct transfer and her own via OT), she runs the Ev algorithm to compute the encoded output. This output is then decoded either locally by the evaluator or, depending on the protocol, returned to the garbler for decoding. To defend against malicious behavior, protocols often employ a *cut-and-choose* [33] approach: the garbler generates multiple circuits, a random subset of which are opened for verification to ensure correctness.

D. Commitment scheme

We instantiate the non-interactive commitment scheme from a hash function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{2\kappa}$ as $\mathsf{Com}(m; r) := \mathcal{H}(m \parallel r)$, with decommitment $m \parallel r$. For messages with high min-entropy (e.g., a garbled label L), we use the saltless form $\mathsf{Com}(L) := \mathcal{H}(L)$. On Bitcoin, both are realized in Script using native opcodes such as `OP_SHA256` or `OP_HASH160`. For a transcript $\tau = (m_1, m_2, \ldots)$ of public messages exchanged during a protocol execution, its *transcript hash sequence* is $\mathcal{H}(\tau) := \big(\mathcal{H}(m_1), \mathcal{H}(m_2), \ldots\big)$.

Appendix D
Algorithm Pseudocode and Transactions

We now present pseudocode for the key scripts and verification routines used in BitPriv, capturing the enforcement logic implemented in Bitcoin transactions and in the BitVM dispute mechanism.

A. *RevealSeeds* Script

The RevealSeeds script (Algorithm 1) enforces that the garbler reveals valid openings for all but one circuit seed and its input labels in the selected evaluation circuit. It ensures that commitments are opened consistently and that the evaluation index m is bound by a Lamport commitment, allowing misbehavior to be detected. By doing so, the script provides the evidence required to prove cheating and enables the dispute mechanism to be resolved on-chain.

B. *Close* script

Algorithm 2 verifies the correctness of the evaluated circuit's output. It reconstructs the output from revealed wire-label preimages and checks that it matches the committed result. If the preimages are inconsistent or the output is incorrect, the script prevents payout, enabling later dispute resolution.

C. *BitVM* pseudocode

This predicate subroutine Φ_{mis} (Algorithm 3) is invoked within BitVM.Enforce to check for protocol violations. It verifies consistency of seeds, circuit encodings, commitments, and OT transcripts. Any mismatch results in a violation bit, which serves as a publicly verifiable certificate of misbehavior.

D. Transaction Specifications

The protocol's logic is embedded in a series of Bitcoin transactions with Taproot scripts. The core transaction chain enforces the protocol flow. Lamport signatures are used to bind a party's choice across transactions, with the public key pk_{m} hardcoded into the scripts during setup. We present a simplified specification of the key transactions and scripts, using shortened names for brevity (e.g., `CommitEvalIndexScript` is denoted `CEIScript`). For the tables, we use the notation from Appendix B.

Setup locks the input and collateral from both parties. Its output is a single UTXO locked by a Taproot script whose leaves encode the spending paths for the CEI transaction, a cooperative Close for each possible output for each possible m (i.e., $n \cdot 2^{\text{outbits}}$ total

Algorithm 1: Redeem script for the RevealSeeds transaction (`RSScript`).

Function $\texttt{RevealSeedsScript}_j\,(\sigma_{GE}, m, \sigma_m, [seedPreimages], [inputPreimages])$ **:**
- $\text{CheckMultiSigVerify}_{pk_{GE}}(\sigma_{GE})$
- $\text{LampSigVerify}_{\mathsf{pk_m}}(m, \sigma_m)$
- ▷ `Verify evaluation index and witness size.`
- **if** $m \neq \mathbf{j}$ ***or*** $|[seedPreimages]| \neq n-1$ ***or*** $|[inputPreimages]| \neq l$ **then**
  - **return** False
- **end**
- ▷ `Verify preimages of seeds with` Com^j `the vector over` $[n] \setminus \{j\}$.
- **for** $k = 0$ **to** $n-2$ **do**
  - **if** $\mathcal{H}(seedPreimages[k]) \neq Com^{\mathbf{j}}(k)$ **then**
    - **return** False
  - **end**
- **end**
- ▷ `Verify preimages of input wire labels.`
- **for** $t = 0$ **to** $l-1$ **do**
  - **if** $\mathcal{H}(inputPreimages[t]) \notin \{com_{X_{j,t}}(0), com_{X_{j,t}}(1)\}$ **then**
    - **return** False
  - **end**
- **end**
- **return** True

leaves in the two-party case, and $t \cdot n \cdot 2^{\text{outbits}}$ once the payout tree is replicated per buyer as in BitMarket), and a $\mathsf{Timeout}_E$.

Setup Transaction	
Inputs	(1) $(*, *, \texttt{CheckSig}_{\mathsf{pk}_G})$ (2) $(*, *, \texttt{CheckSig}_{\mathsf{pk}_E})$
Outputs	(1) $(total, \langle \texttt{CEIScript}, \texttt{CloseScript}, \texttt{Timeout}_E \rangle)$
Witness	(1) σ_G (2) σ_E

CommitEvaluationIndex spends the Setup output and requires a witness containing a valid Lamport signature σ_m to the evaluation index m. Its output is a UTXO locked by a Taproot script with paths for the `RevealSeeds` transaction (via `RSScript`), $\mathsf{Timeout}_G$, and PunishEquiv path (via `PEScript`). The complete unlocking script is $\texttt{CEIScript} \equiv \texttt{CheckMultiSig}_{\mathsf{pk}_{GE}} \wedge \texttt{LampSigVerify}_{\mathsf{pk_m}}$[8].

[8]Note that the witness contains a binary representation of m to "guide" the verification, and as a result shorten the size of the verification script.

Algorithm 2: Redeem script for the Close transaction. Each leaf contains a script for a different combination of result *out* for evaluated circuit j.

Function $\texttt{CloseScript}_{out,j}\,(\sigma_{GE}, m, \sigma_m, [outPreimages])$ **:**
- $\text{CheckMultiSigVerify}_{pk_{GE}}(\sigma_{GE})$
- $\text{LampSigVerify}_{\mathsf{pk_m}}(m, \sigma_m)$
- ▷ `Let` k `be the number of circuit output bits.`
- **if** $m \neq \mathbf{j}$ ***or*** $|[outPreimages]| \neq k$ **then**
  - **return** False
- **end**
- ▷ `Reconstruct circuit output from the revealed output wire preimages.`
- $computed_out \leftarrow 0$
- **for** $i = 0$ **to** $k-1$ **do**
  - **if** $\mathcal{H}(outPreimages[i]) = Output_j^0(i)$ **then**
    - **continue**
  - **end**
  - **else if** $\mathcal{H}(outPreimages[i]) = Output_j^1(i)$ **then**
    - $computed_out \leftarrow computed_out + 2^i$
  - **end**
  - **else**
    - **return** False
  - **end**
- **end**
- **if** $computed_out \neq out$ **then**
  - **return** False
- **end**
- **return** True

CommitEvaluationIndex Transaction	
Inputs	(1) $(\mathsf{Setup}, 0, \texttt{CEIScript})$
Outputs	(1) $(total, \langle \texttt{RSScript}, \texttt{Timeout}_G, \texttt{PEScript} \rangle)$
Witness	(1) $(\sigma_{GE}, \sigma_m, m)$

RevealSeeds spends the CEI output by satisfying the `RSScript` leaf (see Alg. 1) with preimages for the check circuits' seeds and the evaluation circuit's input labels. The transaction creates a new output locked by a Taproot which has spending paths for the final Close transaction (via `CloseScript`), a BitVM dispute initiation (`BVMScript`), and $\mathsf{Timeout}_E$.

RevealSeeds Transaction	
Inputs	(1) $(\mathsf{CEI}, 0, \texttt{RSScript})$
Outputs	(1) $(total, \langle \texttt{CloseScript}, \texttt{BVMScript}, \texttt{Timeout}_E \rangle)$
Witness	(1) $(\sigma_{GE}, \sigma_m, m, [\text{seeds}], [\text{input}])$

Close spends the output of RS via the `CloseScript` leaf for circuit $i \in [n]$ and output $z = f(x, y)$. The script verifies that

Algorithm 3: Predicate subroutine for Φ_{mis} (used within BitVM.Enforce)

Input : κ-bit values $\mathsf{seed}^*, \mathsf{seed}^*_E$, bit b^*, integers $i^* \in [n],\ \lambda^* \in [l]$
Output: Violation bit β
$C \leftarrow \mathsf{Com}_{\mathsf{seed}_{i^*}}$;
$C_E \leftarrow \mathsf{Com}_{\mathsf{seed}_{E,i^*}}$;
if $\pi_{Vc}(\mathsf{seed}^*, C) = 0 \ \vee\ \pi_{Vc}(\mathsf{seed}^*_E, C_E) = 0$ **then**
  return 1;
$F, e, d, O, I_G, I_E \leftarrow \pi_{Gb}(1^\kappa, f, \mathsf{seed}^*)$;
if $\pi_{Vc}(F, \mathsf{Com}_{F_{i^*}}) = 0 \ \vee\ \pi_{Vc}(d, \mathsf{Com}_{d_{i^*}}) = 0$ **then**
  return 1;
for $j \leftarrow 1$ **to** k **do**
  $(L^0_{out,j}, L^1_{out,j}) \leftarrow O_j$;
  if $\pi_{Vc}(L^0_{out,j}, \mathsf{Output}^0_j[i^*]) =$
   $0 \ \vee\ \pi_{Vc}(L^1_{out,j}, \mathsf{Output}^1_j[i^*]) = 0$ **then**
    return 1;
for $j \leftarrow 1$ **to** l **do**
  $(L^0_{G,j}, L^1_{G,j}) \leftarrow I_{G,j}$;
  if $\neg\big(\pi_{Vc}(L^0_{G,j}, \mathsf{Com}_{X_{i^*,j}}(0)) \wedge$
   $\pi_{Vc}(L^1_{G,j}, \mathsf{Com}_{X_{i^*,j}}(1))\big) \ \wedge$
   $\neg\big(\pi_{Vc}(L^1_{G,j}, \mathsf{Com}_{X_{i^*,j}}(0)) \wedge$
   $\pi_{Vc}(L^0_{G,j}, \mathsf{Com}_{X_{i^*,j}}(1))\big)$ **then**
    return 1;
transcript $\leftarrow \pi_{OT}(\mathsf{seed}^*, \mathsf{seed}^*_E, i^*, \lambda^*, b^*, I_{E,\lambda^*})$;
for $j \leftarrow 1$ **to** $|transcript|$ **do**
  if $\pi_{Vc}(transcript(j), \mathsf{Com}_{OT_{i^*,\lambda^*,j}}) = 0$ **then**
    $r \leftarrow \pi_\phi(i^*, \lambda^*, j)$;
    return $1 - r$;
return 0;

E provides valid output labels $\{Z\}$ for which $\mathsf{De}(Z) = z$ for the chosen circuit F_m. This transaction creates two new outputs according to $g(z)$: $G's$ output (value_G which is immediately spendable and $E's$ output (value_E, which however, is locked behind a new script. This script allows E to claim their funds after a timelock (Δ), but it also includes a `PEScript` path that allows G to claim the funds if E equivocated on the Lamport signature.

Close Transaction	
Inputs	(1) $(\mathsf{RS}, 0, \texttt{CloseScript})$
Outputs	(1) $(\text{value}_G, \texttt{CheckSig}_{\mathsf{pk}_G})$ (2) $(\text{value}_E, \langle\, \texttt{CheckSig}_{\mathsf{pk}_E} \wedge \text{TL}(\Delta), \texttt{PEScript}\,\rangle)$
Witness	(1) $(\sigma_{GE}, m, \sigma_m, [\text{outPreimages}])$

From pseudocode to Bitcoin Script. The algorithms we present in D describe the verification logic for Bitcoin scripts but are presented as pseudocode for clarity. In practice, these serve as *script factories* that generate the actual Bitcoin Script in the following way: since Bitcoin Script does not support loops or complex control flow, each Taproot leaf corresponds to a single hardcoded verification check for a specific evaluation index m and a specific output value z.

Appendix E
BitPriv in the $(\mathcal{F}_{\text{BitVM}}, \mathcal{L}, \mathcal{F}_{\text{OT}})$-hybrid model

In this section, we specify BitPriv as a protocol in the $(\mathcal{F}_{\text{BitVM}}, \mathcal{L}, \mathcal{F}_{\text{OT}})$-hybrid model. The description proceeds in two phases: a *setup phase*, where parties exchange commitments and prepare transactions, and an *active phase*, where transactions are posted and disputes, if any, are resolved via $\mathcal{F}_{\text{BitVM}}$.

BitPriv Description

1) **Setup Phase.**
   a) **(Circuit Generation and Commitment.)** For each $i \in [n]$, G generates a garbled circuit $(F_i, e_i, d_i) \leftarrow \text{Gb}(1^\kappa, f, \text{seed}_i)$ using an independent random seed seed_i. G sends commitments $\mathsf{Com}_{\mathbf{seed_i}}$, $\mathsf{Com}_{\mathbf{F_i}}$, and $\mathsf{Com}_{\mathbf{d_i}}$ to E, along with the plaintext garbled circuits F_i and the opening to commitments Com_{seed_i}. E verifies the structural integrity of each F_i and its corresponding commitment. Otherwise E aborts.
   b) **(Output Label Commitment.)** For each output wire $j \in [k]$ (corresponding to one plain-text output bit), G constructs two vectors: $\mathsf{Output}^0_j = [\mathsf{Com}(L^0_{\mathsf{out},i,j})]_{i\in[n]}$ and $\mathsf{Output}^1_j = [\mathsf{Com}(L^1_{\mathsf{out},i,j})]_{i\in[n]}$, where $L^b_{\mathsf{out},i,j}$ denotes the wire label corresponding to output bit $b \in \{0,1\}$ on wire j in circuit F_i. These vectors are then sent to E. This step ensures that the output decoding (and thus payout conditions) is fixed across all circuits.
   c) **(Garbler's Input Commitment.)** To commit to his input x for each circuit F_i while preserving privacy, G commits to his input labels in a randomly permuted order. For each input bit $j \in [l]$ of circuit F_i, G commits to the pair $(L^b_{G,i,j}, L^{1-b}_{G,i,j})$ for a random bit b, and sends the commitment $\mathsf{Com}_{\mathbf{X_{i,j}}}$ to E.
   d) **(Oblivious Transfer (OT) and Transcript Commitment.)** For each circuit $i \in [n]$ and each of her input bits $\lambda \in [l]$, E engages in an OT protocol with G to receive her corresponding input label $L^y_{E,i} := L^{y_\lambda}_{E,i,\lambda} \forall \lambda \in l$. To ensure privacy in the case where the OT transcript is part of the fraud proof, E uses her real input y only for a single secret evaluation index m, and dummy inputs (e.g., 0^λ) for all other $i \neq m$. To make the protocol reproducible, E also commits to a random seed $\text{seed}_{E,i}$ from which her randomness is drawn throughout the execution for $i \in [n]$ and sends $\mathsf{Com}_{\mathbf{seed_{E,i}}}$ to G. The parties commit to the full public transcripts of all OT instances, enabling post-facto verification by the BitVM instance.
   e) **(Transaction Pre-signing/ BitVM preparation.)** Using the commitments generated above, G and E collaboratively construct and mutually pre-sign a tree of Bitcoin transactions. This transaction set, detailed in Section D-D, embeds the protocol logic into on-chain spendable scripts. The commitments generated in the previous steps are also hard-coded in the logic of the BitVM instance to allow for later verification. If they agree on the transaction set, they are ready to proceed to the on-chain phase. If at any point there is disagreement, either party can abort and the protocol ends.

2) **Active Phase.** *The description of transactions* {Setup, CommitEvaluationIndex, RevealSeeds, Close} *is given in section V*
   a) **(Post Setup)** E or G sends $(\texttt{POST}, tx) \overset{\tau_0}{\hookrightarrow} \mathcal{L}$ with $tx.id$ = Setup.id. Let $\tau_1 \leq \tau_0 + u$ be when Setup appears on $\mathcal{L}$
   b) **(upon** Setup $\in \mathcal{L}$.TX**)**
   Distinguish between the following cases:

Optimistic Path Through an authenticated off-chain communication channel, E sends a Lamport commitment on the evaluation index m using the public key hardcoded in the prepared transactions. If G replies with the corresponding pre-images as in step 2(c)i of the pessimistic path and verifies the correctness of all operations in circuits $F_j \; \forall j \neq m$. If the verification succeeds, E proceeds to close the protocol immediately by evaluating F_m and spending the Setup transaction with a Close transaction. Otherwise, E follows the pessimistic path (second case).

---

   i) E commits to the evaluation index $m \in [n]$ via a Lamport commitment with $(\texttt{POST},$ CommitEvaluationIndex$) \overset{\tau_2}{\hookrightarrow} \mathcal{L}$
   ii) If Setup remains unspent on $\mathcal{L}$ for Δ rounds, G sends $(\texttt{POST}, tx_G) \overset{\tau_2'}{\hookrightarrow} \mathcal{L}$ with $tx_G.Out$ = (all, One-Sig$_{Pk_G}$)
   Wait (up to maximum $\tau_2' + u$) for a transaction spending Setup to appear on $\mathcal{L}$: If $tx_G \in \mathcal{L}$.TX halt, otherwise let τ' be the time that CommitEvaluationIndex appears on $\mathcal{L}$
   c) **(upon** CommitEvaluationIndex $\in \mathcal{L}$.TX**)**
   Distinguish between the following cases:
   i) G spends CommitEvaluationIndex by posting pre-signed $(\texttt{POST},$ RevealSeeds$) \overset{\tau'}{\hookrightarrow}$, revealing the following information as part of the witness for unlocking the corresponding Tapleaf script: i) preimages for $seed$ commitments in combination $m := [seed_i : \forall i \in [n] \setminus \{m\}]$ and ii) the input label for each bit of his input in circuit F_m, that is $L^{x_\lambda}_{G,m,\lambda} \; \forall \lambda \in [l]$ where x_λ is the λ-th bit of $G's$ input. For each λ the label has to be a valid decommitment to one of the two commitments in tuple Com$_{X_{m,\lambda}}$.
   ii) If E commits to a different evaluation index than the one she previously committed to through their off-chain communication channel, G sends $(\texttt{POST}, tx_G \overset{\tau'}{\hookrightarrow} \mathcal{L}$ with $tx_G.Out$ = (all, One-Sig$_{Pk_G}$)
   iii) If CommitEvaluationIndex remains unspent on $\mathcal{L}$ for Δ rounds, E sends $(\texttt{POST}, tx_E) \overset{\tau_5}{\hookrightarrow} \mathcal{L}$ with $tx_E.Out$ = (all, One-Sig$_{Pk_E}$)
   Wait (up to maximum $\tau_5 + u$) for a transaction spending CommitEvaluationIndex to appear on $\mathcal{L}$: If $(tx_E \vee tx_G) \in \mathcal{L}$.TX halt, otherwise let τ'' be the time that RevealSeeds appears on $\mathcal{L}$
   d) **(upon** RevealSeeds $\in \mathcal{L}$.TX**)**
   Distinguish between the following cases:
   i) E sends valid $(\texttt{POST},$ Close$_{out,i}) \overset{\tau_7}{\hookrightarrow} \mathcal{L}$ with $i = m$, spending the outputs of the RevealSeeds transaction. G cannot produce a valid witness for the PunishEquivScript so after Δ rounds, let $(c_A, c_B) \leftarrow g(out)$, then the output of Close$_{out,i}$) is equivalent to $((c_A,$ One-Sig$_{Pk_A}), (c_B,$ One-Sig$_{Pk_B}))$
   ii) E sends valid $(\texttt{POST},$ Close$_{out',i}) \overset{\tau_7}{\hookrightarrow} \mathcal{L}$ with $i \neq m$, spending the outputs of the RevealSeeds transaction. G can then spend both outputs of the transaction by utilizing the PunishEquivScript within Δ rounds.
   iii) E sends $(\texttt{POST},$ setup$) \overset{\tau_7}{\hookrightarrow} \mathcal{L}$ where setup is the transaction received by F_{BitVM} and sends $(\texttt{START}) \overset{\tau_7}{\hookrightarrow} F_{BitVM}$. The parties then interact with F_{BitVM} which handles the allocation of funds in $\mathcal{L}$.
   iv) If RevealSeeds remains unspent on $\mathcal{L}$ for Δ rounds, G sends $(\texttt{POST}, tx_G) \overset{\tau_8}{\hookrightarrow} \mathcal{L}$ with $tx_G.Out$ = (all, One-Sig$_{Pk_G}$)

Appendix F
BitPriv Proof of Security

We first show that BitPriv achieves covert security, analyzing separately the cases of a malicious garbler and a malicious evaluator. We then prove publicly verifiable covert (PVC) security by establishing public verifiability and defamation-freeness. Finally, we use PVC to demonstrate that BitPriv satisfies the protocol goals defined in Section II.

Theorem 1. *Assume a straight-line extractable commitment scheme in the global random oracle model (ROM), the garbling scheme is secure, Π_{OT} UC-realizes $\mathcal{F}_{OT}$, and the signature scheme is existentially unforgeable under a chosen-message attack (EUF-CMA). Then* BitPriv *securely realizes $\mathcal{F}_{PCDC}$ in the $(\mathcal{F}_{BitVM}, \mathcal{L}, \mathcal{F}_{OT})$-hybrid model.*

Proof. Covert Security - Malicious G

Let $\mathcal{A}$ be an adversary that corrupts G. We construct the following simulator S[9] that runs $\mathcal{A}$ as a subroutine, while playing the role of G in the ideal world interacting with $\mathcal{F}$:

Simulator for Malicious G

(Signature Forgery) If the simulator S detects a valid signature with the secret key of E on a transaction it did not sign during the simulation, S outputs error and aborts. If the simulator S detects a valid one-time (Lamport) signature with the secret key of E on a value it did not sign during the simulation, S outputs error and aborts.

Setup Phase
1) choose n uniform κ-bit strings $seed_{E_i} \forall i \in [n]$ and send $\{\text{com}_{seed_{E_i}}\}_{i=1}^n$ to $\mathcal{A}$
2) Upon receiving the non-hiding commitments $\{\text{com}_{F_i}\}_{i=1}^n$ and straight-line extractable computationally hiding/binding $\{\text{com}_{seed_i}\}_{i=1}^n$ from the adversary $\mathcal{A}$:
   - The simulator S extracts the underlying circuits $\{F_i\}$ and $\{seed_i\}$ from the extractable commitments.
   - If any F_i does not match com_{F_i}, send $(\texttt{ABORT}) \overset{\tau}{\hookrightarrow} \mathcal{F}$ and halt.
3) Upon receiving the commitments to the output wire labels $\text{Output}_j^0 = [\textbf{com}(L^0_{\text{out},i,j})]_{i\in[n]}$ and $\text{Output}_j^1 = [\textbf{com}(L^1_{\text{out},i,j})]_{i\in[n]}$:
   - S extracts the values and stores them for each circuit $i \in [n]$.
4) Upon receiving commitments to the garbler's input wire labels for each circuit $i \in [n]$:
   - S extracts the committed values $X_{i,j} := (X^b_{i,j}, X^{1-b}_{i,j})$, a randomly permuted pair of the j-th input label of G.
5) S runs n Π_{OT} instances with $\mathcal{A}$, where S plays the receiver using

[9] We only allow the simulator to rewind $\mathcal{A}$ in the Setup Phase, before any interaction with the ledger functionality $\mathcal{L}$

fixed selection bits 0^λ and randomness derived from $seed_{E_i} \forall i \in [n]$. S saves the transcript of messages exchanged during each (i,λ)-th OT instance $transcript_{i,\lambda} := \{msg_{i,\lambda,j}\}\ \forall j \in |transcript_{i,\lambda}|$ and their non-hiding commitments $com_{OT_{i,\lambda}} := [com_{\text{msg}_{i,\lambda,j}}]\ \forall i \in [n],\ \forall \lambda \in [l],\ \forall j \in |transcript_{i,\lambda}|$. S verifies that the commitments received from $\mathcal{A}$ match the transcript and otherwise send (ABORT) $\overset{\tau}{\hookrightarrow} \mathcal{F}$ and halt.

6) For all $i \in [n]$ use the extracted $seed_i$ to reproduce the steps of an honest G during protocol steps 1.{a,b,c,d} and in particular compute $F'_i, \text{Output}'_{zero}, \text{Output}'_{one}, com'_{OT_{i,\lambda}}, com'_{X_{i,j}} \forall i \in [n]$. Let J be the set of indices where $\rho \in J$ if: $F'_\rho \neq F_\rho \vee \text{Output}^0_\rho \neq \text{Output}'^0_\rho \vee \text{Output}^0_\rho \neq \text{Output}'^1_\rho \vee \exists \lambda : com'_{OT_{\rho,\lambda}} \neq com_{OT_{\rho,\lambda}} \vee \exists j : com'_{X_{\rho,j}} \neq com_{X_{\rho,j}}$.
   a) If $|J| = 0$ set $att = \perp$
   b) If $|J| = 1$ set att = cheat(z')
   c) if $|J| \geq 2$ set att = blatant-cheat
7) S goes through the presigning phase of transactions and the initialisation of F_{BitVM} with $\mathcal{A}$ as an honest E would using the messages received above. If at any point $\mathcal{A}$ does not follow the protocol, send abort message to $\mathcal{F}$ and stop. Otherwise, if presigning of messages is done correctly, send (SETUP,p,att) to $\mathcal{F}$ on behalf of $\mathcal{A}$ and (SETUP,p,y) as an honest E would. Receive the message from $\mathcal{F}$ and set the flag detectedA accordingly.
8) S rewinds the adversary $\mathcal{A}$ until $|J'| = |J|$ and detectedA' = detectedA by repeating steps 1-5 as above with the following differences:
   Choose random $m \in [n]$ at the beginning of the simulation.
   a) In step 4: Run Π_{OT} for $i \neq m$ as above with input 0^λ randomness drawn from uniform $seed_{E_i}$, and for m use the simulator S_{OT} for Π_{OT} extracting all input labels $L^b_{E,m}$ for $b \in \{0,1\}$. Use the extracted values and the messages received to reconstruct the actions of an honest G and in particular: Let J' be the set of indices where $\rho' \in J'$ if: $F'_{\rho'} \neq F_{\rho'} \vee \text{Output}^0_{\rho'} \neq \text{Output}'^0_{\rho'} \vee \text{Output}^0_{\rho'} \neq \text{Output}'^1_{\rho'} \vee \exists \lambda : com'_{OT_{\rho',\lambda}} \neq com_{OT_{\rho',\lambda}} \vee \exists j : com'_{X_{\rho',j}} \neq com_{X_{\rho',j}}$.
      i) if $|J'| = 1 \wedge m \notin J'$ set detectedA=1
      ii) if $|J'| = 1 \wedge m \in J'$ set detectedA=0
      iii) if $|J'| \geq 2$ set detectedA=1
   b) S goes through the presigning phase of transactions exactly as an honest E would. If at any point $\mathcal{A}$ gives an invalid signature or aborts, rewind to step 8

Active Phase

1) S sends (POST, Setup) to $\mathcal{L}$ exactly as an honest E would and monitors $\mathcal{L}$. As soon as Setup $\in \mathcal{L}$.TX S sends (START) to $\mathcal{F}$ and receives back the reply message. If received (ERROR) output error and halt. Otherwise continue to the next step.
2) S creates a valid Lamport signature for the chosen evaluation index m (with the secret key whose corresponding public key is hardcoded in the locking script of CommitEvaluationIndex) and sends (POST, CommitEvaluationIndex) to $\mathcal{L}$. As soon as CommitEvaluationIndex $\in \mathcal{L}$.TX S sends (REQUEST,m) to $\mathcal{F}$ and receives back the reply message based on the *detectedA* flag which S forwards to $\mathcal{A}$.
3) S monitors $\mathcal{L}$ for the RevealSeeds transaction from $\mathcal{A}$. If the transaction appears within Δ distinguish according to the *detectedA* flag:
   - if *detectedA*, send (REVEAL-A,z') to $\mathcal{F}$ and receive back (Detected-A, witness) where the witness contains values from F_j, $j \in J$ such that v(witness, 1) where v is the verification function of F_{BitVM}.
   - if $\neg$*detectedA*: (REVEAL-A,z') to $\mathcal{F}$ and receive back (OUTPUT),z'. Use the witness of the RevealSeeds transactions to extract the input labels $L^{x'}_{G,m}$ corresponding to the effective input x' of the adversary $\mathcal{A}$ in F_m. S uses the extracted input labels $L^b_{E,m}$ for $b \in \{0,1\}$ together with y to evaluate circuit F_m as an honest E would. If the result is an output label which has effective output z' when reconstructed with the semantics of values extracted from $\text{Output}^0_j(m)$ and $\text{Output}^1_j(m)\ \forall j \in [l]$ proceed as an honest E would. Otherwise, set an *uneval* flag.
   - If $|J| = 0$ (G did not make a cheat attempt), then S uses the witness of the RevealSeeds transactions to extract the input labels $L^x_{G,m}$ corresponding to the input x of the adversary $\mathcal{A}$ in F_m. S uses the extracted input labels $L^b_{E,m}$ for $b \in \{0,1\}$ together with y to calculate the output $z = f(x,y)$. Send (REVEAL-A, f,x) to $\mathcal{F}$ and receive back the (OUTPUT, z) message.

   If $\mathcal{A}$ fails to post RevealSeeds within the timeout Δ, receive (ABORTED-A) from $\mathcal{F}$ S sends a punishment transaction tx to $\mathcal{L}$. Monitor $\mathcal{L}$ for the next u rounds and if $tx \in \mathcal{L}$.TX, S sends (PUNISH-A) to $\mathcal{F}$ and receives a (PUNISH-ABORT) message in return, which S shares with $\mathcal{A}$. If (ERROR) is received from $\mathcal{F}$, S outputs error and halts. Otherwise, if the RevealSeeds transaction appears instead, then continue as above.
4) Upon receiving the response from $\mathcal{F}$ after the reveal step, S proceeds to the final action:
   a) If *detectedA*, S simulates E initiating a dispute. S sends (START) and (CLAIM, witness, 1) to $\mathcal{F}_{\text{BitVM}}$. If S receives (INIT) from $\mathcal{F}_{\text{BitVM}}$, S then sends (PUNISH-CHEAT) to $\mathcal{F}$ and outputs whatever $\mathcal{F}$ outputs.
   b) If *undetectedA* $\wedge$ $\neg$*uneval* or $|J| = 0$, S sends (POST, $\text{Close}_{z',m}$) to $\mathcal{L}$ and when the transaction appears on $\mathcal{L}$ informs $\mathcal{F}$ by sending (CLOSE, z'). S outputs whatever $\mathcal{F}$ outputs.
   c) If *undetectedA* $\wedge$ *uneval*, S wait for Δ rounds, receives (ABORTED-B) from $\mathcal{F}$m forwards it to $\mathcal{A}$ and halts.

Hybrid 0_a*:* We assume an EUF-CMA signature scheme and a secure one-time signature scheme so the error events (in the light orange frame) of the description of S do not occur, except with negligible probability. Therefore, the distribution of the output of S and Hybrid 0_a is computationally indistinguishable.

Hybrid 0_b*:* This is the ideal-world execution between S and the honest E holding input $y(m)$, interacting with functionality $\mathcal{F}$. By inlining the actions of S, $\mathcal{F}$ and E we may rewrite the experiment as follows:

Hybrid 0_b

Setup Phase

1) choose n uniform κ-bit strings $seed_{E_i} \forall i \in [n]$ and send $\{\text{com}_{seed_{E_i}}\}_{i=1}^n$ to $\mathcal{A}$
2) Upon receiving the non-hiding commitments $\{\text{com}_{F_i}\}_{i=1}^n$ and straight-line extractable computationally hiding/binding $\{\text{com}_{seed_i}\}_{i=1}^n$ from the adversary $\mathcal{A}$:
   - extract the underlying circuits $\{F_i\}$ and $\{seed_i\}$ from the extractable commitments.
   - If any F_i does not match com_{F_i}, E aborts and outputs $\perp$
3) Upon receiving the commitments to the output wire labels $\text{Output}^0_j = [\mathbf{com}(L^0_{\text{out},i,j})]_{i\in[n]}$ and $\text{Output}^1_j = [\mathbf{com}(L^1_{\text{out},i,j})]_{i\in[n]}$:
   - extract the values and store them for each circuit $i \in [n]$.
4) Upon receiving commitments to G's input wire labels for each circuit $i \in [n]$:

- extract the committed values $X_{i,j} := (X^b_{i,j}, X^{1-b}_{i,j})$, a randomly permuted pair of the j-th input label of G.

5) Run n Π_{OT} instances with $\mathcal{A}$, as the receiver using fixed selection bits 0^λ and randomness derived from $seed_{E_i} \forall i \in [n]$. Save the transcript of messages exchanged during each (i,λ)-th OT instance $transcript_{i,\lambda} := \{msg_{i,\lambda,j}\}\ \forall j \in |transcript_{i,\lambda}|$ and their non-hiding commitments $com_{OT_{i,\lambda}} := [com_{\text{msg}_{i,\lambda,j}}]\ \forall i \in [n],\ \forall \lambda \in [l],\ \forall j \in |transcript_{i,\lambda}|$. Verify that the commitments received from $\mathcal{A}$ match the transcript and otherwise E aborts and outputs $\perp$
6) For all $i \in [n]$ use the extracted $seed_i$ to reproduce the steps of an honest G during protocol steps 1.{a,b,c,d} and in particular compute $F'_i, \text{Output}'_{zero}, \text{Output}'_{one}, com'_{OT_{i,\lambda}}, com'_{X_{i,j}} \forall i \in [n]$. Let J be the set of indices where $\rho \in J$ if: $F'_\rho \neq F_\rho \vee \text{Output}^0_\rho \neq \text{Output}'^0_\rho \vee \text{Output}^0_\rho \neq \text{Output}'^1_\rho \vee \exists \lambda : com'_{OT_{\rho,\lambda}} \neq com_{OT_{\rho,\lambda}} \vee \exists j : com'_{X_{\rho,j}} \neq com_{X_{\rho,j}}$.
   a) If $|J| = 0$ set $att = \perp$
   b) If $|J| = 1$ set $att =$ cheat(z')
   c) if $|J| \geq 2$ set $att =$ blatant-cheat
7) Go through the presigning phase of transactions and the initialisation of F_{BitVM} with $\mathcal{A}$ as an honest E would using the messages received above. If at any point $\mathcal{A}$ reply or sends invalid messages, E aborts. Otherwise, if presigning of messages is done correctly, if $|J| \geq 2$ set detectedA, if $|J| = 1$ with probabilty ϵ set $detectedA$ and with the remaining probability $1-\epsilon$ set $undetectedA$. If $|J| = 0$, do nothing and continue below
8) Rewind $\mathcal{A}$ until $|J'| = |J|$ and detectedA' = detectedA by repeating steps 1-5 as above with the following differences:
   Choose random $m \in [n]$ at the beginning of the simulation.
   a) In step 4: Run Π_{OT} for $i \neq m$ as above with input 0^λ randomness drawn from uniform $seed_{E_i}$, and for m use the simulator S_{OT} for Π_{OT} extracting all input labels $L^b_{E,m}$ for $b \in \{0,1\}$. Use the extracted values and the messages received to reconstruct the actions of an honest G and in particular: Let J' be the set of indices where $\rho' \in J'$ if: $F'_{\rho'} \neq F_{\rho'} \vee \text{Output}^0_{\rho'} \neq \text{Output}'^0_{\rho'} \vee \text{Output}^0_{\rho'} \neq \text{Output}'^1_{\rho'} \vee \exists \lambda : com'_{OT_{\rho',\lambda}} \neq com_{OT_{\rho',\lambda}} \vee \exists j : com'_{X_{\rho',j}} \neq com_{X_{\rho',j}}$.
      i) if $|J'| = 1 \wedge m \notin J'$ set detectedA=1
      ii) if $|J'| = 1 \wedge m \in J'$ set detectedA=0
      iii) if $|J'| \geq 2$ set detectedA=1
   b) Go through the presigning phase of transactions exactly as an honest E would. If at any point $\mathcal{A}$ gives an invalid signature or aborts, rewind to step 8

Active Phase

1) Send (`POST, Setup`) to $\mathcal{L}$ exactly as an honest E would and monitor $\mathcal{L}$. As soon as Setup $\in \mathcal{L}$.TX continue to the next step. If the transaction is not in $\mathcal{L}$ after u rounds, E outputs error and halts.
2) Create a valid Lamport signature for the chosen evaluation index m (with the secret key whose corresponding public key is hardcoded in the locking script of CommitEvaluationIndex) and send (`POST, CommitEvaluationIndex`) to $\mathcal{L}$. As soon as CommitEvaluationIndex $\in \mathcal{L}$.TX continue to the next step. If the transaction is not in $\mathcal{L}$ after u rounds, E outputs error and halts.
3) Monitor $\mathcal{L}$ for the RevealSeeds transaction from $\mathcal{A}$. If the transaction appears within Δ distinguish according to the $detectedA$ flag:
   - if $detectedA$, calculate a witness where the witness contains values from F_j, for uniform $j \in J$ such that $v(\text{witness}, 1)$ where v is the verification function of F_{BitVM}.
   - if $\neg detectedA$: Use the witness of the RevealSeeds transactions to extract the input labels $L^{x'}_{G,m}$ corresponding to the effective input x' of the adversary $\mathcal{A}$ in F_m. Use the extracted input labels $L^b_{E,m}$ for $b \in \{0,1\}$ together with y to evaluate circuit F_m as an honest E would. If the result is an output label which has effective output z' when reconstructed with the semantics of values extracted from $\text{Output}^0_j(m)$ and $\text{Output}^1_j(m)\ \forall j \in [l]$ proceed as an honest E would. Otherwise, set an $uneval$ flag.
   - If $|J| = 0$ (G did not make a cheat attempt), then use the witness of the RevealSeeds transactions to extract the input labels $L^x_{G,m}$ corresponding to the input x of the adversary $\mathcal{A}$ in F_m. Use the extracted input labels $L^b_{E,m}$ for $b \in \{0,1\}$ together with y to calculate the output $z = f(x,y)$.

   If $\mathcal{A}$ fails to post RevealSeeds within the timeout Δ, send a punishment transaction tx to $\mathcal{L}$. Monitor $\mathcal{L}$ for the next u rounds and if $tx \in \mathcal{L}$.TX, send (`PUNISH-ABORT`) to $\mathcal{A}$. Otherwise, if the RevealSeeds transaction appears instead, then continue as above. If neither transaction is not in $\mathcal{L}$ after u rounds, E outputs error and halts.
4) Proceed to the final step:
   a) If $detectedA$, E initiates a dispute by sending (`START`) and (`CLAIM, witness, 1`) to $\mathcal{F}_{\text{BitVM}}$. If received (`INIT`) from $\mathcal{F}_{\text{BitVM}}$, monitor $\mathcal{L}$ for T_{BitVM} rounds and if $tx \in$ TX with $tx.Out = \big((0,\perp), (\text{all}, \text{One-Sig}_{Pk_B})\big)$ send (`FINISH`) to $\mathcal{A}$. Otherwise E outputs error and halts.
   b) If $undetectedA \wedge \neg uneval$ or $|J| = 0$, send (`POST, Close`$_{z',m}$) to $\mathcal{L}$ and when the transaction appears on $\mathcal{L}$ E outputs z' if $undetected$ and $f(x,y)$ otherwise.
   c) If $undetectedA \wedge uneval$, wait for Δ rounds and E sends (`ABORTED-B`) to $\mathcal{A}$ and halts.

Hybrid 1: According to the ledger functionality $\mathcal{L}$, a transaction is delayed by at most u rounds. Additionally, as stated before, the transactions that are pre-signed at the Setup phase uniquely define the eligible set of spending transactions (except with negligible probability). Spending a transaction output with a transaction outside the presigned set would require forging a signature. Therefore, the red events of the previous experiment never occur, except with negligible probability, and the distribution of the outputs of the two experiments is identically distributed.

Hybrid 2: We modify the previous hybrid experiment by choosing a uniform $m \in [n]$ at the beginning of the experiment. Then in step 7 of the Setup Phase: If $|J| \geq 2$ then set $detectedA$, If $|J| = 1$ set $detectedA$ if $m \notin J$ and $undetected$ if $m \in J$. In step 3 of the Active Phase, in the case that $|J| \geq 2$, when calculating a witness, select $j \in J$ such that $j \neq m$. We notice that $m \notin J$ with probability ϵ in the case that $|J| = 1$ so the outputs of Hybrid 1 and Hybrid 2 are identically distributed.

Hybrid 3: The previous hybrid experiment is modified as follows: In step 5 of the Setup Phase, use S_{OT} to run the m'th instance of Π_{OT} and extract all of $\mathcal{A}$'s inputs in those executions. Following the security of Π_{OT} the distribution of the outputs of Hybrid 2 and Hybrid 3 are computationally indistinguishable.

Hybrid 4: Since steps 1-5 have become identical to the modifications applied to the rewinding. We can "collapse" the rewinding t, obtaining the following experiment Hybrid 4 which is statistically indistinguishable from Hybrid 3 (with the only difference occurring in case of an aborted rewinding in the

latter):

Hybrid 4

Setup Phase Choose uniform index $m \in [n]$

1) choose n uniform κ-bit strings $seed_{E_i} \forall i \in [n]$ and send $\{\mathsf{com}_{seed_{E_i}}\}_{i=1}^{n}$ to $\mathcal{A}$
2) Upon receiving the non-hiding commitments $\{\mathsf{com}_{F_i}\}_{i=1}^{n}$ and straight-line extractable computationally hiding/binding $\{\mathsf{com}_{seed_i}\}_{i=1}^{n}$ from the adversary $\mathcal{A}$:
   - extract the underlying circuits $\{F_i\}$ and $\{seed_i\}$ from the extractable commitments.
   - If any F_i does not match com_{F_i}, E aborts and outputs $\perp$
3) Upon receiving the commitments to the output wire labels $\mathsf{Output}_j^0 = [\mathbf{com}(L^0_{\mathsf{out},i,j})]_{i\in[n]}$ and $\mathsf{Output}_j^1 = [\mathbf{com}(L^1_{\mathsf{out},i,j})]_{i\in[n]}$ extract the values and store them for each circuit $i \in [n]$.
4) Upon receiving commitments to G's input wire labels for each circuit $i \in [n]$ extract the committed values $X_{i,j} := (X^b_{i,j}, X^{1-b}_{i,j})$, a randomly permuted pair of the j-th input label of G.
5) Run Π_{OT} for $i \neq m$ with input 0^λ and randomness drawn from $seed_{E_i}$, and for m use the simulator S_{OT} for Π_{OT} extracting all input labels $L^b_{E,m}$ for $b \in \{0,1\}$. Save the transcript of messages exchanged during each (i,λ)-th OT instance $transcript_{i,\lambda} := \{msg_{i,\lambda,j}\}\ \forall j \in |transcript_{i,\lambda}|$ and their non-hiding commitments $com_{OT_{i,\lambda}} := [com_{\mathrm{msg}_{i,\lambda,j}}]\ \forall i \in [n],\ \forall \lambda \in [l],\ \forall j \in |transcript_{i,\lambda}|$. Verify that the commitments received from $\mathcal{A}$ match the transcript and otherwise E aborts and outputs $\perp$. Use the extracted values and the messages received to reconstruct the actions of an honest G and in particular: Let J be the set of indices where $\rho \in J$ if: $F'_{\rho'} \neq F_\rho \vee \mathrm{Output}^0_\rho \neq \mathrm{Output}'^0_\rho \vee \mathrm{Output}^1_\rho \neq \mathrm{Output}'^1_\rho \vee \exists \lambda : com'_{OT_{\rho,\lambda}} \neq com_{OT_{\rho,\lambda}} \vee \exists j : com'_{X_{\rho,j}} \neq com_{X_{\rho,j}}$.
   a) if $|J| = 1 \wedge m \notin J$ set detectedA=1
   b) if $|J| = 1 \wedge m \in J$ set detectedA=0
   c) if $|J| \geq 2$ set detectedA=1
6) Go through the presigning phase of transactions and the initialisation of $F_{\mathrm{Bit_{VM}}}$ with $\mathcal{A}$ as an honest E would using the messages received above. If at any point $\mathcal{A}$ reply or sends invalid messages, E aborts. Otherwise, if presigning of messages is done correctly, if $|J| \geq 2$ set detectedA, if $|J| = 1$ with probabilty ϵ set *detectedA* and with the remaining probability $1-\epsilon$ set *undetectedA*. If $|J| = 0$, do nothing and continue below

Active Phase

1) Send (`POST, Setup`) to $\mathcal{L}$ exactly as an honest E would and monitor $\mathcal{L}$. As soon as Setup $\in \mathcal{L}$.TX continue to the next step.
2) Create a valid Lamport signature for the chosen evaluation index m (with the secret key whose corresponding public key is hardcoded in the locking script of CommitEvaluationIndex) and send (`POST, CommitEvaluationIndex`) to $\mathcal{L}$. As soon as CommitEvaluationIndex $\in \mathcal{L}$.TX continue to the next step.
3) Monitor $\mathcal{L}$ for the RevealSeeds transaction from $\mathcal{A}$. If the transaction appears within Δ distinguish according to the *detectedA* flag:
   - if *detectedA*, calculate a witness where the witness contains values from F_j, for uniform $j \in J$ such that $v(\text{witness}, 1)$ where v is the verification function of F_{BitVM}.
   - if $\neg$*detectedA*: Use the witness of the RevealSeeds transactions to extract the input labels $L^{x'}_{G,m}$ corresponding to the effective input x' of the adversary $\mathcal{A}$ in F_m. Use the extracted input labels $L^b_{E,m}$ for $b \in \{0,1\}$ together with y to evaluate circuit F_m as an honest E would. If the result is an output label which has effective output z' when reconstructed with the semantics of values extracted from $\mathsf{Output}^0_j(m)$ and $\mathsf{Output}^1_j(m)\ \forall j \in [l]$ proceed exactly as an honest E would when executing the protocol. Otherwise, set an *uneval* flag.
   - If $|J| = 0$ (G did not make a cheat attempt), then use the witness of the RevealSeeds transactions to extract the input labels $L^x_{G,m}$ corresponding to the input x of the adversary $\mathcal{A}$ in F_m. Use the extracted input labels $L^b_{E,m}$ for $b \in \{0,1\}$ together with y to calculate the output $z = f(x,y)$.

   If $\mathcal{A}$ fails to post RevealSeeds within Δ, send a punishment transaction tx to $\mathcal{L}$. Monitor $\mathcal{L}$ for the next u rounds and if $tx \in \mathcal{L}$.TX, send (`PUNISH-ABORT`) to $\mathcal{A}$. Otherwise, if the RevealSeeds transaction appears instead, then continue as above.
4) Proceed to the final step:
   a) If *detectedA*, E initiates a dispute by sending (`START`) and (`CLAIM, witness, 1`) to $\mathcal{F}_{\mathrm{BitVM}}$. If received (`INIT`) from $\mathcal{F}_{\mathrm{BitVM}}$, monitor $\mathcal{L}$ for T_{BitVM} rounds and if $tx \in$ TX with $tx.Out = \big((0,\perp), (\mathsf{all}, \mathsf{One\text{-}Sig}_{Pk_B})\big)$ send (`FINISH`) to $\mathcal{A}$.
   b) If *undetectedA* $\wedge\ \neg$*uneval* or $|J| = 0$, send (`POST, Close`$_{z',m}$) to $\mathcal{L}$ and when the transaction appears on $\mathcal{L}$ E outputs z' if *undetected* and $f(x,y)$ otherwise.
   c) If *undetectedA* $\wedge$ *uneval*, wait for Δ rounds and E sends (`ABORTED-B`) to $\mathcal{A}$ and halts.

Hybrid 5: This hybrid is modified from the previous experiment in how the witness is generated in the *detectedA* case. Specifically, in step 3 of the Active Phase, the witness is now calculated from the values revealed by $\mathcal{A}$ on the ledger $\mathcal{L}$ for the check circuits $j \neq m$, precisely as an honest E would in the real protocol. This same witness is then used in the (`CLAIM, witness, 1`) message sent to the $\mathcal{F}_{\mathrm{BitVM}}$ functionality.

To formalize this, let $\hat{J} := J \setminus \{m\}$ be the set of check-circuit indices where an inconsistency is found. An index $\rho \in [n] \setminus \{m\}$ is in $\hat{J}$ if the reconstructed honest execution using seed_ρ does not match $\mathcal{A}$'s revealed data; i.e., if $F'_\rho \neq F_\rho$, if the output label commitments do not match, or if the OT or input-label commitments are inconsistent. With J defined as in Hybrid 4, we observe the following equivalences:

$$|J| \geq 2 \vee (|J| = 1 \wedge m \notin J) \iff |\hat{J}| \geq 1$$

$$|J| = 0 \vee (|J| = 1 \wedge m \in J) \iff \hat{J} = \emptyset$$

Therefore, the only way for the output distribution of this hybrid to differ from the previous one is if $\mathcal{A}$ violates the binding property of the commitment scheme by revealing decommitments different from those extracted by the simulator. In the random-oracle and ideal-ledger models we assume, this is not possible. Thus, the output distribution of Hybrid 5 is identical to that of Hybrid 4.

Hybrid 6: This hybrid alters the previous experiment in the case where no cheating was attempted ($|J| = 0$). In step 3 of the Active Phase, the output z is now calculated by evaluating the garbled circuit F_m exactly as an honest E would in the protocol

execution, rather than being assigned the value $f(x, y)$ by the experiment's logic.

Since $|J| = 0$, it is established that $\mathcal{A}$ has correctly garbled the circuit for function f into F_m, has committed to valid input labels $L^x_{G,m}$ corresponding to its input x, and has correctly committed the circuit's output labels. By the correctness property of the garbling scheme, the output resulting from an evaluation of F_m will be $f(x, y)$, unless $\mathcal{A}$ violates the binding property of the commitment scheme during the reveal. As this is not possible in the assumed model, the output distributions of Hybrid 6 and Hybrid 5 are identical.

Hybrid 7: We modify the previous experiment by altering the execution of the Oblivious Transfer for the evaluation circuit $i = m$. In step 5 of the setup phase, we now run the real OT protocol, Π_{OT}, with E using her true input y, instead of using the OT simulator S_{OT}. It follows from the security of Π_{OT} that the output distribution of Hybrid 7 is computationally indistinguishable from that of Hybrid 6.

Hybrid 8: In the final modification, the execution of Π_{OT} for the evaluation circuit $i = m$ now proceeds using pseudorandomness derived from the seed $\text{seed}_{E,m}$, rather than true randomness. It is immediate that the output distribution of Hybrid 8 is computationally indistinguishable from that of Hybrid 7.

As Hybrid 8 corresponds to a real-world protocol execution between the adversary $\mathcal{A}$ and an honest party E, this concludes the proof.

Covert Security - Malicious E

Let $\mathcal{A}$ be an adversary that corrupts E. We construct the following simulator S that runs $\mathcal{A}$ as a subroutine, while playing the role of E in the ideal world interacting with $\mathcal{F}$. To avoid lengthy repetition, we omit error events that occur due to signature forgery and $\mathcal{L}$ malfunction from the description of the simulator with the same arguments that were used above.

Simulator for Malicious E

Setup Phase At any point in the setup phase, if $\mathcal{A}$ does not reply within a predetermined number of rounds t_a since the last message received at τ, send $(\texttt{ABORT}) \xrightarrow{\tau+t_a} \mathcal{F}$

1) Receive n commitments $\{\mathsf{com}_{seed_{E_i}}\}_{i=1}^n$ from $\mathcal{A}$ and extract their values.
2) Choose uniform κ-bit strings seed_i $\forall i \in [n]$ and honestly calculate $(F_i, e_i, d_i) \leftarrow \text{Gb}(1^\kappa, f, \text{seed}_i)$. Send the values $\{\mathsf{com}_{seed_i}\}_{i=1}^n$ and $\{\mathsf{com}_{F_i}\}_{i=1}^n$ together with F_i to $\mathcal{A}$.
3) Honestly generate $\mathsf{Output}^0_j = [\mathbf{com}(L^0_{\text{out},i,j})]_{i\in[n]}$ and $\mathsf{Output}^1_j = [\mathbf{com}(L^1_{\text{out},i,j})]_{i\in[n]}$, where $L^b_{\text{out},i,j}$ denotes the wire label corresponding to output bit $b \in \{0, 1\}$ on wire j in circuit F_i and send them to $\mathcal{A}$.
4) Generate the pair $(L^x_{G,i,j}, 0^\lambda)$ where $L^x_{G,i,j}$ is the j-th input label in circuit i which corresponds to the j-th bit of x and send the randomly permuted commitment $\text{com}_{X_{i,j}}$ to $\mathcal{A}$.
5) Use the simulator S_{OT} to interact with $\mathcal{A}$ in all executions of Π_{OT} thus extracting all the values y^*_i. (Note that if $\mathcal{A}$ executes Π_{OT} honestly there is a single $m \in [n]$ such that $y_m = y$ and $y_i = 0$ $\forall i \neq m$.) Save all the transcripts for the (i, λ)-th OT instance $transcript_{i,\lambda} := \{msg_{i,\lambda,j}\}$ $\forall j \in |transcript_{i,\lambda}|$ and their non-hiding commitments $com_{OT_{i,\lambda}} := [com_{\text{msg}_{i,\lambda,j}}]$ $\forall i \in [n]$, $\forall \lambda \in [l]$, $\forall j \in |transcript_{i,\lambda}|$. Send the commitments to $\mathcal{A}$ and if the adversary aborts send $(\texttt{ABORT}) \overset{\tau}{\hookrightarrow} \mathcal{F}$ and halt.
6) Go through the presigning phase of transactions and the initialization of F_{BitVM} as an honest G would verifying that transactions are prepared correctly according to the above commitments sent to and received from $\mathcal{A}$. If at any points $\mathcal{A}$ does not sign a transaction or attempts to sign a maliciously crafted one, send $(\texttt{ABORT}) \overset{\tau}{\hookrightarrow} \mathcal{F}$ and halt. Otherwise, send $(\texttt{SETUP}, \texttt{p}, \perp)$ and $(\texttt{SETUP}, \texttt{p}, [y])$ on behalf of $\mathcal{A}$.

Active Phase

1) Send $(\texttt{POST}, \texttt{Setup})$ to $\mathcal{L}$ exactly as an honest G would and monitors $\mathcal{L}$. As soon as $\mathsf{Setup} \in \mathcal{L}.\mathsf{TX}$ S sends $(\texttt{START})$ to $\mathcal{F}$ and forwards the reply message to $\mathcal{A}$.
2) Monitor $\mathcal{L}$ for the next Δ rounds. Once $\mathsf{CommitEvaluationIndex}$ appears on $\mathcal{L}$ extract the value m from the witness of the transaction together with the Lamport signature on that value which is valid with respect to the public key hardcoded in the transaction and send $(\texttt{REQUEST}, m)$ on behalf of $\mathcal{A}$ and receive $(\texttt{REVEAL}, m)$ from $\mathcal{F}$ .If $\mathsf{CommitEvaluationIndex}$ transaction $\notin \mathcal{L}.\mathsf{TX}$ after Δ rounds, receive $(\texttt{ABORTED-B})$ from $\mathcal{F}$ and send punishment transaction tx (spending condition is expired timelock) to $\mathcal{L}$. Monitor $\mathcal{L}$ for the next u rounds: if $tx \in \mathcal{L}.\mathsf{TX}$, send $(\texttt{PUNISH-ABORT})$ to $\mathcal{A}$. Otherwise, if the $\mathsf{CommitEvaluationIndex}$ transaction appears instead, then continue as above.
3) Post $\mathsf{RevealSeeds}$ transactions, including the Lamport signature received above in the witness, revealing seed_i $\forall i \neq m$ and $L^x_{G,m}$ corresponding to input x. Once $\mathsf{RevealSeeds} \in \mathcal{L}.\mathsf{TX}$, send $(\texttt{REVEAL-A}, f, x)$ to $\mathcal{F}$, and forward the reply $(\texttt{OUTPUT}, f(x, y))$ to $\mathcal{A}$ where y is the value y_m used by $\mathcal{A}$ in the executions of Π_{OT} of input labels for F_m.
4) Distinguish between the possible actions of $\mathcal{A}$:
   a) If $\mathcal{A}$ posts Close using one-time signature for m which is validated by $\mathcal{L}$. Output $f(x, y)$, wait for Δ rounds and send $(\texttt{FINISH})$ to $\mathcal{A}$ on behalf of $\mathcal{F}$.
   b) If $\mathcal{A}$ posts Close signing a different index $j \neq m$ (equivocating). Spend the output on $\mathcal{L}$ before the timelock Δ expires, and send $(\texttt{EQUIV-DETECTED})$ to $\mathcal{A}$ on behalf of $\mathcal{F}$. Output $\perp$ and halt.
   c) If $\mathcal{A}$ spends $\mathsf{RevealSeeds}$ with the setup transaction received by $\mathcal{F}_{\text{BitVM}}$, extract the values w, o from $(\texttt{CLAIM}, w, o)$, verify that $v(w, o) \neq 1$ and output $\perp$. Otherwise output error and halt
   d) If $\mathcal{A}$ does not send a valid transaction to $\mathcal{L}$ spending the output of $\mathsf{RevealSeeds}$. Send tx to $\mathcal{L}$ spending the output through the expired timelock. Monitor $\mathcal{L}$ for the next u rounds, and if $tx \in \mathcal{L}.\mathsf{TX}$, send $(\texttt{PUNISH-ABORT})$ to $\mathcal{A}$ on behalf of $\mathcal{F}$, output $\perp$ and halt. Otherwise, if a Close or setup transaction appears on $\mathcal{L}$ instead, continue as above.

Hybrid 1: We argue that the event in red occurs with negligible probability. Since the garbling scheme $\mathcal{G}$ and the transcript of Π_{OT} can be reconstructed deterministically, for an honest G, the only scenario where $v(w, o) \neq 1$ is if:

1) $\mathcal{A}$ uses a different decommitment $seed'$ than the ones revealed by S in the witness of $\mathsf{RevealSeeds}$ to forge $\text{com}_{seed_j} = \text{com}_{seed'}$ for some $j \in [n] \setminus m$
2) $\mathcal{A}$ uses a different decommitment in w to a value $seed_{E_i}$ than the ones extracted by S at step 1 of the Setup Phase.
3) $\mathcal{A}$ uses a different decommitment in w for a commitment $com_{OT_{i,\lambda}} := [com_{\text{msg}_{i,\lambda,j}}]$ $\forall i \in [n]$, $\forall \lambda \in [l]$, $\forall j \in |transcript_{i,\lambda}|$

However, due to the binding property of the commitment in

the random oracle model, these events occur with negligible probability. Therefore, the distributions of the outputs of the two experiments are computationally indistinguishable.

Hybrid 2: We modify the previous experiment in the following way: In Step 4 of the Active Phase, in the case that $\mathcal{A}$ does not attempt to equivocate: reconstruct the output value z with the semantics of values of $\mathsf{Output}_j^0(m)$ and $\mathsf{Output}_j^1(m)$ $\forall j \in [l]$ and output z. Due to the security of Π_{OT}, $\mathcal{A}$ receives only the labels corresponding to y_m, and receives the input labels corresponding to x at step 3 as decommitments. Due to the security properties of $\mathcal{G}$, the only output labels that $\mathcal{A}$ learns that function as decommitments to values of $\mathsf{Output}_j^0(m)$ and $\mathsf{Output}_j^1(m)$, are $L_{\mathrm{out},j,m}^{z[j]}$ which unlock commitments $\mathsf{Output}_j^{z[j]}(m)$ for each output bit $j \in [l]$, where $z = f(x, y)$. It follows that the distribution of Hybrid 2 is computationally indistinguishable from the distribution of Hybrid 1.

Hybrid 3: We modify the previous experiment in the following way: In step 3 of the Active Phase, generate the permuted commitment tuple exactly as an honest G would. It follows from the hiding property of the commitment scheme that the two distributions are computationally indistinguishable.

Hybrid 4: Finally, change the last experiment by executing protocol Π_{OT} in step 5 of the Setup Phase. It is immediate from the security of Π_{OT} that the two distributions are computationally indistinguishable.

Since Hybrid 4 corresponds to a real-world execution of the protocol by an honest G, this completes the proof. □

Publicly Verifiable Covert Security

We use the notion of *publicly verifiable covert (PVC) security* from Definition 1, comprising (i) covert security with deterrence parameter ϵ, (ii) public verifiability via a certificate (witness) w with $\Phi_{\mathsf{mis}}(w, o) = 1$ when cheating is detected, and (iii) defamation-freeness (no certificate can be forged against an honest party, except with negligible probability).

Corollary 1 (PVC Security)**.** *In the $(\mathcal{F}_{BitVM}, \mathcal{L}, \mathcal{F}_{OT})$-hybrid model, and under the assumptions in Section II, the protocol Π_{BitPriv} satisfies PVC security with deterrence parameter $\epsilon = 1 - \frac{1}{n}$, where n is the number of circuits in the cut-and-choose.*

Proof. It remains to prove public verifiability and defamation-freeness. These properties are captured directly in our ideal functionality $\mathcal{F}_{\mathsf{PCDC}}$. In particular, when an honest evaluator E initiates the dispute mechanism through $\mathcal{F}_{\mathsf{BitVM}}$, the resulting witness w constitutes a valid certificate, while for an honest garbler G, we showed that it is impossible for E to forge w such that $v(w, o) = 1$.

Corollary 2. BitPriv *achieves* privacy*,* financial fairness*,* correctness *and* guaranteed output delivery *with probability at least* ϵ.

Proof. Implied by Corollary 1, the financial enforcement of $\mathcal{F}_{\mathrm{BitVM}}$, and the security of $\mathcal{L}$.

Guaranteed output delivery uses the financially compensated notion from §VI. Cleve's impossibility [87] rules out fair general two-party computation with a dishonest majority, so no protocol in this setting can guarantee that an aborting party's counterparty receives z. $\mathcal{F}_{\mathsf{PCDC}}$ and the timelocked graph guarantee instead that a party denied z is compensated from the aborting party's collateral. This guarantee holds with probability ϵ, and in the complementary $1-\epsilon$ fraction of undetected Garbler deviations the corollary makes no claim about the Evaluator's output.

Appendix G
Compliance of Rational Parties

Our security proof guarantees that cheating is detected with high probability and financially penalized on-chain. For the protocol to be effective in practice, however, honesty must be the most profitable strategy for rational, utility-maximizing parties. This is achieved by requiring parties to lock financial collateral high enough to ensure that the expected outcome of any cheating attempt is a net loss.

G's potential cheating actions are constrained in two key ways: (i) all potential fund allocations are defined by pre-signed transactions, limiting them to biasing the outcome, not inventing a new one; and (ii) the cut-and-choose mechanism ensures any deviation from the protocol is detected with probability $\epsilon = 1 - 1/n$. These constraints leave two primary avenues for attack:

1) **Function Manipulation**: G may alter the circuit's logic to prioritize either financial gain or information leakage. For financial gain, the function can be modified to deterministically yield the most profitable pre-signed allocation, irrespective of the inputs. For information leakage, the function can map specific bits of E's input to distinct outputs, thereby encoding information into the final payout.
2) **Selective or Full Denial-of-Service (DoS)**: G can engineer the circuit to produce output wire labels that do not correspond to any of the pre-signed payout preimages, again choosing between a financial or informational objective. A *full DoS* forces failure for all inputs, guaranteeing G receives the fairness/liveness punishment γ. In contrast, a *selective DoS* forces failure only for a subset of inputs. This partitions the input space, allowing G to learn a single bit of information based on whether the protocol aborts or completes successfully.

A. Garbler Compliance

A rational G deviates only if the expected gain is positive.
Model parameters:

- $\alpha(\sigma)$: utility from learning $\sigma \in \{0, \ldots, k\}$ bits about E's input y.
- R: reputational cost if publicly caught cheating.
- r: reputational cost if G notices a non-evaluable circuit without public evidence.
- col_G: G collateral; in_G: G input coins; in_E: E input coins.
- $u_G(z)$: utility from fund allocation $g(z)$; $u_{G,\max}$ is its maximum.
- ϵ: detection probability, where $\epsilon = 1 - 1/n$.
- γ: penalty for E not revealing output.

Strategy 1: *Function Manipulation.* Since payouts are fixed in advance, the only way to extract information about y is to modify

the function so that the selected presigned allocation correlates with bits of y. This may force the run into allocations that are less favorable for G. Therefore, G finds the optimal balance between maximizing financial payout and gaining information. According to their personal evaluation of the utility from learning specific bits, G might map multiple outputs of $\hat{y}$ to the same allocation to avoid payouts that are not worth the extra information gain.

Let $\hat{f}(x, y) = \hat{y}$ be the modified function and let $Q(\sigma)$ be the best allocation achievable while leaking σ bits. G chooses:

$$\sigma^* \in \arg \max_{\sigma \in \{0,\dots,k\}} \Big(\alpha(\sigma) + \mathbb{E}[u_G(Q(\sigma))]\Big).$$

If not detected, the expected utility is $\mathcal{W}_1 = \alpha(\sigma^*) + \mathbb{E}[u_G(Q(\sigma^*))]$, and the expected gain is:

$$\mathbb{E}[\text{Gain}_1] = (1-\epsilon)\,\mathcal{W}_1 - \epsilon\,(col_G + in_G + R). \tag{1}$$

As σ grows, $\alpha(\sigma)$ increases while $\mathbb{E}[u_G(Q(\sigma))]$ typically decreases, capturing the money-for-information trade-off.

Strategy 2: *DoS*. In a Denial-of-Service attack, G engineers the circuit to fail on a chosen subset of E's inputs, $S \subseteq \mathcal{Y}$. This partitions the input space, creating a binary outcome: the protocol either completes successfully or it aborts. By observing this outcome, G learns whether E's input y was in the failure set S.

The amount of information gained from this binary observation is fundamentally limited by its entropy. For a binary event with success probability p, the entropy $H_2(p)$ is at most one bit, a maximum reached when the outcome is most uncertain ($p = 0.5$). Formally, the mutual information $I(Y; F(Y))$ between the secret input and the binary outcome is bounded by this value:

$$I(Y; F(Y)) \le H(F(Y)) = H_2(p), \text{ where } p = \Pr[Y \notin S].$$

G's optimal choice of p balances the utility from the information leak against the competing financial outcomes of success versus failure. This trade-off is formalized as the following utility-maximization problem:

$$p^* \in \arg \max_{p \in [0,1]} \Big[\alpha_{\text{DoS}}(p) + p \cdot \mathbb{E}\big[u_G(Q_{\text{DoS}}(p))\big] + (1-p)\cdot(\gamma + in_G - r)\Big] \tag{2}$$

Here, $\alpha_{\text{DoS}}(p)$ represents G's subjective utility from the information gained when the circuit fails with probability p. G's optimal strategy, p^*, is the probability that maximizes their potential gain from this cheat. This maximized utility is $\mathcal{W}_2$:

$$\mathcal{W}_2 = \alpha_{\text{DoS}}(p^*) + \mathbb{E}\Big[p^* \cdot \mathbb{E}\big[u_G(Q_{\text{DoS}}(p^*))\big] + (1-p^*)\cdot(\gamma + in_G - r)\Big] \tag{3}$$

The final expected gain, $\mathbb{E}[\text{Gain}_2]$, determines if the strategy is rational by weighing $\mathcal{W}_2$ against the penalty of being caught:

$$\mathbb{E}[\text{Gain}_2] = (1-\epsilon)\,\mathcal{W}_2 - \epsilon\,(col_G + in_G + R). \tag{4}$$

Theorem 2 (Garbler Compliance)**.** *The protocol achieves rational garbler compliance for parameters* $(\alpha, \epsilon, in_G, in_E, g, col_G, R, r)$ *if*

$$\max\big(\mathbb{E}[Gain_1],\ \mathbb{E}[Gain_2]\big) < 0$$

B. Evaluator Compliance

E's only profitable deviation is to learn the output $z = f(x, y)$ but refuse to complete the protocol, thus keeping the result secret from G. Let β be E's utility from this secrecy, and let r_E be the reputation cost to E from this deviation. To ensure compliance, the financial fairness penalty for being unresponsive, denoted γ and enforced by timelocks, must outweigh any utility gained from secrecy.

Theorem 3 (Evaluator Compliance)**.** *The protocol achieves rational evaluator compliance for a set of parameters* (β, γ, r_E) *if the fairness penalty an unresponsive E incurs,* γ*, satisfies:*

$$\gamma + r_E > \beta$$

C. Extended collateral parameterization

We derive the parameterization shown in Figure 3 by instantiating Eq. (5). Each party locks $\text{IN}_G = \text{IN}_E = V = \$1{,}000$, and the pre-signed graph redistributes the two amounts according to the outcome of a function. The largest pre-signed payout to G awards it the total locked input amounts by both parties: $u_{G,\max} = \text{IN}_G + \text{IN}_E = 2V$.

A detected deviation exposes G's collateral, locked input, and reputational cost. We take no reputation credit on either side, setting $R = r_E = 0$. Function manipulation gives G utility at most $\alpha(\sigma^*) + u_{G,\max}$, whereas denial of service gives at most $\alpha_{\text{DoS}}(p^*) + \gamma + \text{IN}_G$. Rather than estimating the optimal σ^* and p^*, we conservatively set $\alpha(\sigma^*) = \alpha_{\text{DoS}}(p^*) = \alpha_{\text{full}}$, where α_{full} is the value G assigns to learning E's complete input. This upper-bounds both partial leakage and the one-bit signal from selective denial of service.

Under these parameters, the minimum total slashable Garbler stake is

$$C_G^{\min}(n, \gamma, \alpha_{\text{full}}) = D + \frac{\alpha_{\text{full}} + \text{IN}_G + \max\{\text{IN}_E, \gamma\}}{n-1}. \tag{5}$$

Because IN_G is already slashable, the separate collateral deposit is $\text{COL}_G^{\min} = \max\{0, C_G^{\min} - \text{IN}_G\}$. This deposit is returned after an honest execution, so its honest-path cost is therefore the time-value of locked capital.

Evaluator compliance (Theorem 3) requires $\gamma + r_E > \beta$. With $r_E = 0$, the values $\gamma \in \{\$500, \$1{,}000, \$2{,}000\}$ deter withholding utilities below one half, all, and twice the locked input, respectively. A larger γ strengthens Evaluator compliance but also increases the Garbler's required collateral whenever $\gamma > \text{IN}_E$.

Figure 3 exposes the trade-off in n. At a fixed output width, smaller n reduces circuit material and the Close graph linearly but increases the Garbler stake through the $1/(n-1)$ factor. It also increases the honest Evaluator's expected timeout loss in an undetected denial of service, γ/n. For $\gamma = V$, this loss is \$500, \$250, \$125, and approximately \$63 at $n = 2, 4, 8, 16,$

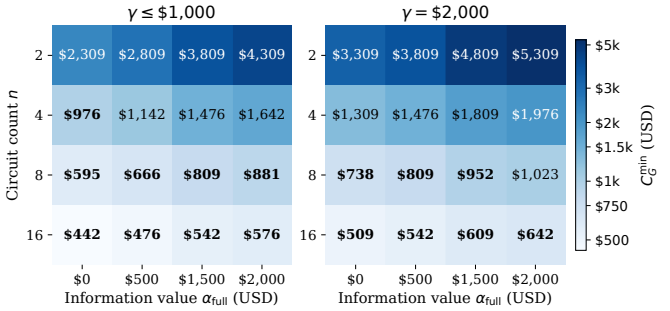

Fig. 7: Minimum total slashable Garbler stake $C_G^{\min}$ under the BitVMX dispute backend (D = \$308.63). Parameters match Figure 3: each party locks \$1,000, $u_{G,\max}$ = 2,000, $R = r_E = 0$, and $\alpha(\sigma^*) = \alpha_{\text{DoS}}(p^*) = \alpha_{\text{full}}$. Bold entries satisfy $C_G^{\min} \le \text{IN}_G$.

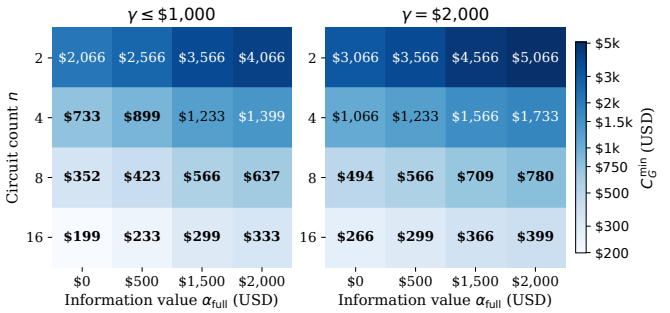

Fig. 8: Minimum total slashable Garbler stake $C_G^{\min}$ under the BABE dispute backend (D = \$65.40), using the parameters of Figure 3. Bold entries satisfy $C_G^{\min} \le \text{IN}_G$.

respectively. Thus $n = 4$ offers lower setup with weaker deterrence, $n = 8$ provides an intermediate point, and $n = 16$ provides stronger deterrence at greater setup cost. Table X reports the corresponding setup costs.

Other dispute backends. The main parameterization fixes BitVM3 as the dispute backend. Because D enters Eq. (5) additively, switching backends shifts every cell of Figure 3 by a constant. At 3 sat/vB, the BitVMX and BABE dispute paths cost \$308.63 and \$65.40, respectively, compared with \$43.75 for BitVM3 (Table II). Figures 7 and 8 show the corresponding grids.

BABE adds only \$21.65 relative to BitVM3 and therefore leaves the region in which the locked input covers the requirement unchanged. BitVMX adds \$264.88: for $\gamma \le \text{IN}_E$, the $n = 4$, $\alpha_{\text{full}} = 0$ configuration remains covered by the locked input, but the other displayed $n = 4$ configurations require a separate collateral deposit. All displayed configurations with $n \ge 8$ remain covered except (γ = \$2,000, α_{full} = \$2,000) at $n = 8$, which exceeds the locked input by \$23.

BitBridge example collateralization. BitBridge composes an auction instance with a settlement instance. The client and the winning solver (market maker MM_w) exchange the Garbler and Evaluator roles between the two instances, while D_{auc} and D_{set} denote their backend-specific dispute costs.

Auction. The client is the Garbler and locks the bridge input IN_C. It may deviate either to learn the solvers' private quotes or to manipulate the circuit so that the winner settles at a price different from its quote. We set α_{MEV} = \$250 as an illustrative value for learning the quotes, approximately the mean realized profit per sandwich or arbitrage instance reported by Qin et al. [88]. This is a calibration point rather than an upper bound on the application-specific information value.

We assume that the winning solver observes the settlement price before delivery and can abort at cost γ. The client can therefore improve its settlement price by at most γ before the solver prefers to abort. Conservatively, we charge the client for both learning the quotes and obtaining this maximum price improvement, even though these gains need not occur together. Applying the Garbler-compliance bound gives the client a required total slashable auction stake of

$$C_C^{\text{auc}} = D_{\text{auc}} + \frac{\alpha_{\text{MEV}} + \gamma + \text{IN}_C}{n-1}.$$

Because IN_C is already slashable, the client's additional auction collateral is $\text{COL}_C^{\text{auc}} = \max\{0, C_C^{\text{auc}} - \text{IN}_C\}$.

Each solver is an Evaluator during the auction. Its input remains on the destination chain and is neither locked nor slashable on Bitcoin. Every solver therefore posts enough collateral to cover either an auction dispute or the timeout penalty for failing to respond:

$$\text{COL}_S^{\text{auc}} = \max\{D_{\text{auc}}, \gamma\}.$$

The maximum suffices because the dispute and timeout branches are mutually exclusive.

Settlement. After a successful auction, the winning solver (MM_w) becomes the Garbler and the client becomes the Evaluator. Settlement has no private input, so the solver gains nothing from learning intermediate values. Its relevant deviation is to bypass verification and obtain the client's locked BTC without completing the required cross-chain delivery. This gain is at most IN_C.

The solver has no Bitcoin input that can be credited toward its slashable stake. It therefore locks the full settlement Garbler requirement as collateral:

$$\text{COL}_S^{\text{set}} = D_{\text{set}} + \frac{\text{IN}_C}{n-1}.$$

As the settlement Evaluator, the client requires total slashable stake covering either a settlement dispute or the timeout penalty:

$$C_C^{\text{set}} = \max\{D_{\text{set}}, \gamma\}.$$

Its additional settlement collateral is $\text{COL}_C^{\text{set}} = \max\{0, C_C^{\text{set}} - \text{IN}_C\}$ because the bridge input is already slashable.

Collateral reuse. All parties post collateral before the auction because the winning solver is not yet known. Settlement begins only after the auction has completed successfully and can no longer be disputed. The same collateral can therefore secure a party's auction and settlement roles, so each party locks the larger of its two requirements.

The client locks its bridge input together with the larger additional collateral requirement:

$$\text{IN}_C + \max\left\{\text{COL}_C^{\text{auc}}, \text{COL}_C^{\text{set}}\right\}.$$

Equivalently, its total locked amount is $\max\{\text{IN}_C, C_C^{\text{auc}}, C_C^{\text{set}}\}$: the slashable bridge input secures both roles up to its value.

Every solver locks the larger of its auction-Evaluator requirement and the collateral it would need if selected as the settlement Garbler:

$$\max\left\{\text{COL}_S^{\text{auc}}, \text{COL}_S^{\text{set}}\right\}.$$

During the auction, this collateral secures the solver's Evaluator role. After the auction, losing solvers recover their collateral once the result can no longer be disputed. The winning solver keeps the same collateral locked until settlement completes.

For a concrete example, let both instances use $n = 16$, with $\text{IN}_C = \$1{,}000$ and $\alpha_{\text{MEV}} = \$250$. We conservatively assume that the Evaluator's withholding utility is twice the calibrated MEV information value and set $\gamma = 2\alpha_{\text{MEV}} = \$500$. With $r_E = 0$, this satisfies Theorem 3 for $\beta < \$500$. The resulting total locked amounts for the client and each solver are respectively \$1,000 and \$500 under BitVM3 or BABE, and \$1,000 and \$660.97 under BitVMX. The client figures include its locked input IN_C.

D. Coalition Resistance for BitBridge

Alice can garble a circuit maliciously, a complicit solver can withhold its input labels after honest solvers reveal theirs, and a complicit solver can falsely challenge a correct circuit. These attacks can always be mounted by a coalition and are independent of the garbling detection probability. The garbling dispute pays the disputer's backend cost D and shares the remaining stake among all t solvers. A withholding timeout shares the deposit among Alice and the other $t-1$ solvers. A failed challenge reimburses Alice's dispute cost and shares the rest among the $t-1$ solvers that did not challenge. A coalition of Alice and complicit solvers could recover its penalty through these shares, so we burn a fraction ω chosen for a tolerated coalition size $k_{\max}$.

We treat the coalition as one adversary with a combined utility, so only flows into or out of it matter. Under every rule the coalition's share of the redistributed amount is at most $\rho_{\text{coal}} = k/t$ for k complicit solvers. A deviation gains at most $\alpha_{\text{coal}}(\sigma)$, which covers learning the $t-k$ honest quotes and blocking an unfavorable auction outcome. The penalty $P_{\mathsf{slash}} > 0$ is what the deviator loses after reimbursing its dispute costs to the other party, and it is positive because every collateral requirement includes at least the dispute costs (Appendix G-C). The coalition recovers at most $(k/t)(1-\omega)P_{\mathsf{slash}}$ and forfeits P_{slash}, so its joint utility is negative whenever

$$\omega > 1 - \frac{t}{k}\left(1 - \frac{\alpha_{\text{coal}}(\sigma)}{P_{\mathsf{slash}}}\right). \tag{6}$$

Setting $k{=}k_{\max}$ deters every coalition up to the tolerated size, so the burned fraction grows with $k_{\max}$.

When the required penalty is infeasible or the information value is unbounded, a deployment caps the notional or uses the malicious cut-and-choose setup of §VIII-D. As an example, take $t{=}4$ solvers with the \$500 timeout deposit of Appendix G-C and coalition gain $\alpha_{\text{coal}}{=}\$300$. A coalition of $k{=}2$ recovers half of the redistributed amount, so Equation (6) requires $\omega > 20\%$.

Appendix H
BitBridge Protocol Description

We give a protocol description of BitBridge in Fig. 10, Fig. 11 and Fig. 12, Fig. 13, collected at the end of the paper. The description treats BitBridge as a composition of two BitPriv instances: a BitMarket instance and a privacy-free settlement instance. For readability, we write the settlement verifier as $\Phi_{\mathsf{settle}}(w, a_{\mathsf{eth}}; \pi)$, where the public statement is $\phi = (w, a_{\mathsf{eth}})$ and π is the SNARK proof of destination-chain delivery. The settlement transaction graph is the standard BitPriv graph (Setup, CEI, RS, Close) of Section V, with the OT phase and input-label permutation removed (the settlement instance has no private inputs).

a) Helper Functions.: We use the following auxiliary functions in the protocol description, in addition to the garbling-scheme algorithms Gb, En, Ev, De defined in Section III.

- $\mathsf{AuctionCircuit}(p_{\mathsf{reserve}}, b_1, \ldots, b_t)$: Returns (w, a_{eth}) if $\max_i b_i \geq p_{\mathsf{reserve}}$, and returns $\perp$ otherwise. The value a_{eth} is computed according to the clearing rule fixed during setup.
- $\mathsf{SettleCircuit}(w, a_{\mathsf{eth}}; \pi)$: Returns True if and only if π proves that a finalized destination-chain transaction transfers at least a_{eth} from MM_w to Alice's destination address a_{addr}.
- $\mathsf{LamportVerify}(\ell, x, \sigma)$: Returns 1 if and only if σ is a valid Lamport signature on message x under Lamport public key ℓ.
- $\mathsf{Equal}(x, x')$: Returns 1 if and only if the two bit strings are equal.
- $\mathsf{LamportGen}(1^\kappa)$: Generates a fresh Lamport keypair (sk_ℓ, ℓ).

A. Atomic Settlement for Auctioned Assets

BitMarket can be composed with various settlement and fulfillment mechanisms to handle a range of assets. We outline two primary applications: the trade of (a) general digital goods and (b) assets on blockchain systems.

General Digital Goods via Fair Data Exchange. To handle the atomic sale of off-chain digital goods (e.g., software licenses or encrypted documents), the protocol integrates with a Bitcoin-compatible *Fair Data Exchange (FDE)* protocol [58]. This is achieved by locking the seller's payment from the `Close` transaction in an output that requires an additional witness to be spent. For example, the seller might be required to reveal a decryption key for the purchased data as the witness, directly linking payment to fulfillment.

Example: rollup data availability auction. A rollup [32], [59] is a Layer 2[10] scaling solution that executes transactions off-chain and periodically anchors a cryptographic summary of its state to a parent blockchain. This requirement creates a market where the rollup's batch of transaction data is the asset being auctioned. As shown in [58], this trade can be implemented

[10]Layer 2 (L2) refers to a framework built atop a blockchain (Layer 1) to increase throughput and reduce costs that relies on the Layer 1 for security.

efficiently if the buyer holds a cryptographic commitment to the specific data they wish to purchase.

Blockchain Assets via On-chain Verification. To settle the transfer of on-chain assets, the auction uses a trust-minimized verifier of events on a target chain, including Bitcoin itself [3], [2], [89], [90]. The seller's payment is conditioned on them providing a succinct proof (e.g., a SNARK) of the asset's transfer to the buyer within a time limit; failure to do so results in their collateral being slashed. This proof is verified within a SNARK that is setup for a *light client* relation for the target blockchain [91], [92], [93], [94]. For instance, this allows for verifying the transfer of a Bitcoin *ordinal* (a Bitcoin-based NFT) [95] or interfacing with other chains using protocols like IBC [96]. The result is atomic cross-chain settlement without requiring pre-deployed contracts on the target chain.

Example: future block space auction. A time-sensitive user privately agrees on a fee with a set of mining pools for inclusion within a window of W blocks. The contract enforces an *if-mined-then-include* rule: if the winning pool mines any block in $[h_0, h_0 + W]$[11], it must include the precommitted transaction. This is enforced by a BitVM light client instance set up at h_0 where the pool's identity is bound via a coinbase commitment. Thus, to claim the output that is determined by BitMarket the pool must provide a proof of inclusion of the transaction in a block.

Appendix I
Additional Evaluation Results

A. Graph Pre-Signing

For t buyers, n cut-and-choose circuits, and an k-bit output, setup contains $tn2^k$ Close templates. Each template carries $t+1$ signatures and corresponds to one Taproot leaf, so the template and leaf counts coincide in this construction. We generated the complete graph in parallel using eight CPU cores, as described in §VIII. Table IV presents both parameter sweeps. The template count, time, signature material, and serialized graph size scale linearly with n. Additional buyers increase the template count and the signatures carried by each template.

TABLE IV: Graph pre-signing (k=16): (a) circuit count at t=1; (b) buyer count at n=4. The sweeps coincide at n=4, t=1; those rows are separate runs.

(a) Varying circuit count n

n	Detection	Templates	Time	Sigs	Graph	B/template
4	75.0%	262,144	57.2 s	33.6 MB	0.50 GB	1,945
8	87.5%	524,288	119.0 s	67.1 MB	1.07 GB	2,045
16	93.8%	1,048,576	247.1 s	134 MB	2.24 GB	2,145
32	96.9%	2,097,152	507.1 s	268 MB	4.70 GB	2,245

(b) Varying buyer count t

t	Templates	Sigs/template	Time	Sigs	Graph
1	262,144	2	58.0 s	33.6 MB	0.50 GB
2	524,288	3	132.3 s	101 MB	1.08 GB
4	1,048,576	5	293.7 s	336 MB	2.35 GB
8	2,097,152	9	715.8 s	1.21 GB	5.30 GB

[11] This window must be carefully selected to avoid penalizing variance.

At n=16 and t=1, the graph carries 134 MB of signature material per party, of which 67 MB crosses the setup channel because the designated broadcaster omits its own signature on each template. Each party deterministically constructs the templates and stores the signed graph locally, requiring 2.24 GB when serialized. Packing reduces this without changing the graph shape, by hashing the concatenated output labels (Packed-Concat) or representing the label vector as a single 33-byte point (Packed-SECP), yielding serialized graphs of 1.68 GB/1.45 GB.

B. OT-Verification Cost

A dispute opens the single OT instance λ^*, so OT verification costs the same regardless of how many buyers the auction has. At t=2 the predicate executes 6,816,169 steps, of which 1,151,117 remain when OT verification is compiled out. Table V sweeps the buyer count: the OT term is constant (with 1.8% of execution variance) across $t \in \{1, \ldots, 32\}$, and what grows is gate re-garbling, which scales with circuit size rather than with t.

TABLE V: Auction-predicate trace against buyer count (n=16, 16-bit prices). Re-garbling is the trace without OT verification.

t	Evaluator inputs	Trace	Re-garbling	OT term
1	16	6,139,392	443,498	5,695,894
2	32	6,816,169	1,151,117	5,665,052
4	64	8,357,644	2,590,792	5,766,852
8	128	11,165,612	5,502,696	5,662,916
16	256	17,154,129	11,388,233	5,765,896
32	512	28,981,995	23,272,589	5,709,406

Fixed-base arithmetic reduces the OT-verification cost by approximately 5× for every value of t. Its end-to-end benefit decreases, however, because the unaffected re-garbling cost grows with t. Total predicate execution improves by 4.7× at $t = 1$, from 28.8 M to approximately 6.1 M steps, but by only 1.8× at $t = 32$, from 52.4 M to approximately 29.0 M steps.

In the BitVMX backend, the parties fix an execution-step ceiling during setup, which determines the depth of the pre-signed dispute search. Radix-16 fixed-base arithmetic reduces the executed instructions within this ceiling by replacing repeated arithmetic with lookups into a precomputed table. However, including this table in the BitVMX-committed program image increases the number of ROM leaves from 612 to 3,732. This creates a deployment trade-off between prover computation and BitVMX setup costs.

C. Transaction-Size Scaling

Table VI reports asymptotic per-transaction sizes and representative costs at n=k=16 and 3 sat/vB. Every transaction after Setup is a Taproot script-path spend, so each carries a control block whose size grows with the depth of the taptree it spends from. Summing the rows, the optimistic path (Setup with Close-PS) totals 690 vB ($1.68) and the liveness-guaranteeing path (Setup, CEI, RS) totals 1,367 vB ($3.34).

TABLE VI: Transaction scaling and representative BitMarket costs (n=k=16, 3 sat/vB). Asymptotic entries omit $O(\cdot)$. l denotes the bits of the input. U/PC/PS denote Unpacked, Packed-Concat, and Packed-SECP.

Transaction	Witness	Leaf	Control	vB / USD
Setup	1	—	—	251 / $0.61
CEI	$\log n$	$\log n$	$k + \log n$	≈ 375 / $0.92
RS	$l + n$	$l + n$	$\log n$	≈ 741 / $1.81
Close-U	$k + \log n$	$k + \log n$	$k + \log n$	630 / $1.54
Close-PC	$k + \log n$	$\log n$	$k + \log n$	495 / $1.21
Close-PS	$\log n$	$\log n$	$k + \log n$	439 / $1.07

Close transaction optimizations. Table VII separates the effect of the number of output bits from the complete-graph count. Packed-Concat hashes the concatenated labels and is valid through k=32 for 16-byte labels under the 520-byte stack-element limit. Packed-SECP represents the output-label vector with one 33-byte point.

TABLE VII: Representative Close size in vB against output width.

Output bits	Unpacked	Packed-Concat	Packed-SECP
1	314	314	319
2	336	327	327
4	378	351	343
8	462	399	375
16	630	495	439
32	966	687	567
64	1,630	—	815

D. Dispute-Cost Details

Table IX complements the workload–backend comparison in Table II with the state, witness, round, and transaction profiles of the two BitVMX predicates. Table XI gives the corresponding fee sensitivity.

Predicate Structure. The auction predicate re-garbles the challenged circuit from its revealed seed and checks the garbled tables, Garbler input labels, and output labels against their commitments. It also re-derives the single challenged OT instance λ^* and verifies its transcript. For larger circuits, we implement an authenticated single-gate optimization: the Garbler stores with each gate's table the difference between its output label and a pseudorandom value fixed by the seed. Then, it commits a single Merkle root over these extended garbled tables. This allows that during a dispute phase, instead of re-garbling the entire circuit, the predicate recomputes the contested gate's input labels from the seed, authenticates one opening of the Merkle commitment, and verifies only a single gate. The trace length in our optimized version, therefore, grows with the commitment depth, which is logarithmic in the circuit size (and not linear, as in the naive solution). This variant changes how check (2) of Φ_{mis} (§V-C) is evaluated, but does not change it's relation. The root instantiates $\text{Com}(F_i)$ and is fixed at Setup alongside $\text{Com}(seed_i)$, so binding follows from the same random-oracle assumption as the single commitment. Thus, the Garbler cannot open any leaf to two distinct values except with negligible probability. During cut-and-choose verification, the Evaluator re-garbles each opened circuit from its revealed seed and compares it leaf by leaf against the committed tree, so locating a deviation falls out of the comparison she already performs when checking the root. The predicate first checks that the revealed seed opens $\text{Com}(seed_i)$. If the committed tree then deviates anywhere from $\mathsf{Gb}(1^\kappa, \Phi_{\text{settle}}, seed)$, the topologically first deviating leaf fails a locally checkable condition, as a garbled table is inconsistent with the labels the seed and its stored difference give. The Evaluator submits that index, and the predicate authenticates the opening and repeats the single local check. Against an honest Garbler, every opening equals the honestly derived value, so no index satisfies the predicate, and defamation-freeness is preserved. Theorem 1 therefore carries over unchanged, since the proof uses only binding and extractability of the commitments, both of which the Merkle instantiation provides. Table VIII evaluates the re-garbling structure on the auction and the single-gate structure on settlement.

TABLE VIII: BitVMX trace length by workload and predicate structure.

Workload	Predicate	Steps
Auction	Re-garble, single OT instance	6,816,169
Settlement	Authenticated single gate	17,173,511

The persistent BitVMX commitment state is 32 B for the disputed auction instance and 52 B for settlement's 20-byte Merkle root and 32-byte seed commitment. Commitment openings can be regenerated on demand. The committed witnesses are 40 B and 744 B, respectively. Table IX reports the remaining resource profile.

TABLE IX: BitVMX resource profile for the auction and settlement predicates.

Cost dimension	Auction Φ_{mis}	Settlement Φ_{mis}
Dispute-time compute	1.40 s / 6.82 M steps	3.87 s / 17,173,511 steps
Peak RAM	38.5 MB	43.7 MB
Persistent commitment state	32 B	52 B
Committed witness	40 B	744 B
Search rounds	11	7–13
On-chain cost	126,421 vB	243,436 vB

Setup and Fee Sensitivity. Table X shows how setup costs vary with the cut-and-choose parameter n. Increasing n increases the number of garbled circuits and therefore the setup time and storage, whereas an active dispute verifies only the selected evaluation instance and does not replay all n circuits. The probability of detecting a malformed circuit is $1 - 1/n$, giving 75%, 87.5%, and 93.75% for $n = 4, 8, 16$, respectively. Figure 3 shows how these detection probabilities affect the required collateral. In a t-solver BitBridge instance, each solver generates its own garbled circuits, so aggregate setup storage scales linearly with t. Finally, Table XI reports the cost of each complete pessimistic path across fee rates from 3 to 300 sat/vB.

Measurement Provenance. Our measurements on the experimental configuration described in §VIII cover BitPriv setup, BitVMX execution, complete transaction paths, and SP1/Groth16 proof generation. The settlement proof takes 2,650 s, peaks at 106.93 GB, verifies in 400 ms, and is 356 bytes. The BitVM3 and BABE backend setup, MCC evaluation, storage, and vsize figures are drawn from [41]. Table II combines

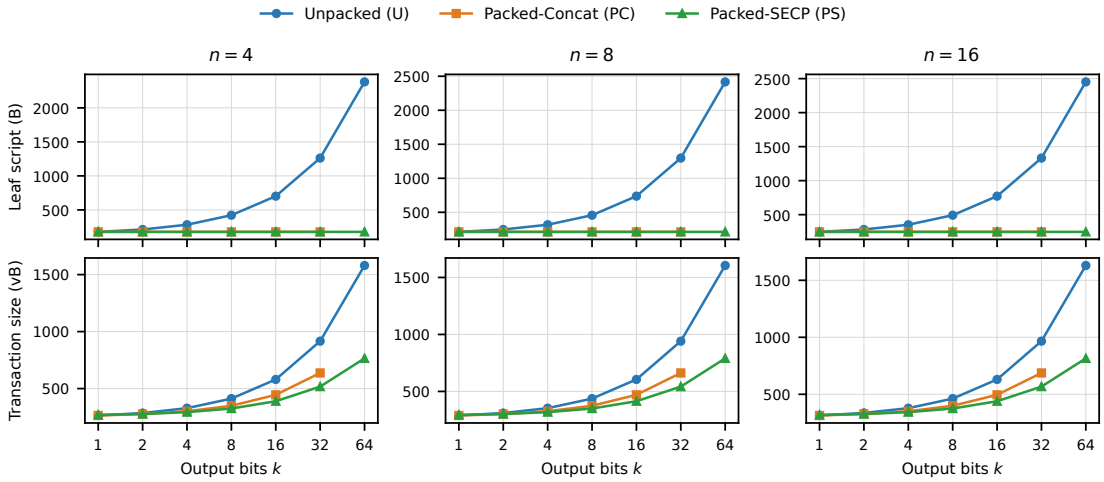

Fig. 9: Close scaling for the three witness designs. Values through k=16 are constructed Taproot spends, while k=32, 64 use byte-accurate serialization. Packed-Concat ends at k=32 because the next point exceeds Bitcoin's 520-byte stack-element limit.

TABLE X: Cryptographic setup against n for the auction and settlement instances of BitBridge. Auction times scale the measured EMP garbling and OT cost at t=2, n=16 linearly with n. Settlement setup times use `[h:]mm:ss`, with hours omitted when zero.

		Auction		Settlement	
n	Detection	Garble+OT	Storage	Setup/solver	Material/solver
4	75.0%	0.017 s	34.4 KB	35:45	347.5 GB
8	87.5%	0.034 s	65.9 KB	1:04:30	695 GB
16	93.8%	0.067 s	128.8 KB	2:09:01	1,390 GB
32	96.9%	0.135 s	254.8 KB	4:18:01	2,780 GB

those figures with the proof-generation measurements from this work. Tables use M for measured, D for derived, and R for reported components.

Appendix J
Extended Comparison to Prior Work

To facilitate comparison between our scheme and the related work discussed in §IX, we provide Table XII. In Table XII, each row records over which functions the parties' inputs remain private, whether payment is conditioned on the computed output, whether the construction runs on current Bitcoin Script, and whether it requires trust beyond the ledger it assumes. Footnotes record the specific assumption behind each qualified mark. For merged rows, cells reflect the strongest construction among the cited works. In the payout column, "transfer" denotes conditional transfer of a fixed amount upon a validated output, and "allocation" an output-dependent division of the locked funds. BitPriv is the only entry achieving output-conditioned payout over private inputs on the current Bitcoin Script without external trust. The nearest entries each fail on at least one axis: deposit-based fair 2PC [22] cannot allocate the deposits by the computed output, the claim-or-refund line [24], [25], [26] requires ledger predicates beyond current Script, and privacy-free BitVM circuits [5], [40], [41] compute over public inputs only.

Appendix K
Extended Discussion

In this section, we develop the limitations and deployment implications summarized in §X. We focus on connecting the quantitative security and evaluation results to their operating assumptions.

A. *Limits of economic deterrence*

The cut-and-choose protocol opens $n-1$ circuits for checking and leaves one randomly selected circuit for evaluation. A Garbler that commits to one malformed circuit therefore escapes detection only if that circuit is selected for evaluation, an event occurring with probability $1/n$. Its consequences depend on how the evaluation circuit was malformed. A function-manipulation attack can produce valid output labels for an unfavorable pre-signed allocation, forcing the Evaluator to settle on that outcome. A denial-of-service attack can instead produce no spendable output labels, causing the timeout branch to treat the honest Evaluator as refusing to settle and transfer its penalty γ to the Garbler. Simply treating the evaluation circuit as a check circuit would compromise both input privacy and settlement correctness: the required openings would both expose the Evaluator's input and reveal labels for outcomes not produced by the evaluation of the function.

The compliance bound accounts for both outcomes. The parameters $u_{G,\max}$ and α capture the Garbler's gains from function manipulation and information leakage, whereas γ and α_{DoS} capture the consequences of denial-of-service. Increasing γ strengthens compliance incentives for the Evaluator, but also increases the honest Evaluator's conditional loss when an undetected denial-of-service circuit is selected. Increasing n reduces the probability of any undetected deviation to $1/n$. The Garbler's expected-gain bound incorporates this probability, making deviation economically irrational rather than cryptographically impossible. Fully malicious cut-and-choose, or proofs of correct garbling can remove this economic dependence, but require heavier setup as in [5]. Fully malicious cut-and-choose also

TABLE XI: Full pessimistic-path cost across fee rates, at BTC = $81,377. BitVM3 and BABE backend vsizes come from [41]. Per-transaction sizes exclude the fee-bump output and CPFP child. The BitVMX totals include 3,080 vB of CPFP children on the dispute path.

Configuration	vB	USD at fee rate (sat/vB)				
		3	10	30	100	300
BitMarket–BitVMX	127,788	312	1,040	3,120	10,399	31,197
BitMarket–BitVM3	19,287	47	157	471	1,570	4,709
BitMarket–BABE	28,156	69	229	687	2,291	6,874
BitBridge settlement–BitVMX	269,116	657	2,190	6,570	21,900	65,700
BitBridge settlement–BitVM3	43,600	106	355	1,064	3,548	10,644
BitBridge settlement–BABE	52,469	128	427	1,281	4,270	12,809

TABLE XII: Comparison with the closest prior approaches to private computation and computation-dependent payment on Bitcoin. A ✓ in "Current Bitcoin Script" means the construction needs no consensus or script extension; a ✓ in "No external trust" means it needs nothing beyond the ledger it assumes.

Approach	Private inputs	Output-conditioned payout	Current Bitcoin Script	No external trust	Mechanism
Secure lotteries [23]	✗ (revealed at settlement)	✓ (lottery-specific)	✓	✓	Hash-locked deposits and timelocks
Fair 2PC via deposits [22]	Arbitrary 2PC	✗ (abort penalties)	✓	✓	Timed-commitment deposits
Claim-or-refund frameworks [24], [25]	Arbitrary MPC	✓ (transfer, in hybrid)	✗[a]	✓	Claim-or-refund deposits with sender-specified predicates
Secure cash distribution [26]	Bounded reactive MPC	✓ (allocation, in hybrid)	✗[a]	✓	Reactive claim-or-refund ladder
On-chain PVC judge [20], [68]	Arbitrary 2PC	✗ (cheating penalties)	✗[b]	✓	Judge contract adjudicating PVC certificates
ZK private contracts [11]	General contracts	✓	✗[b]	✗[c]	ZK proofs over private contract state
TEE private contracts [21]	General contracts	✓	✗[d]	✗[e]	Attested off-chain execution
Privacy-free BitVM garbled circuits [5], [40], [41]	✗	✓ (transfer, public inputs)	✓	✓	Verification of a fixed predicate, on-chain fraud proof
BitPriv (this work)	Arbitrary single-round 2PC[f]	✓ ($g(f(x,y))$ allocation)	✓	✓	GHTLC settlement, PVC BitVM disputes, collateral-backed deterrence

[a] Requires opcodes realizing the general claim-or-refund predicate for arbitrary functions ($\mathcal{F}^*_{\mathsf{CR}}$). $\mathcal{F}^*_{\mathsf{CR}}$. [b] Expressive smart-contract chain. [c] Manager and trusted setup. [d] External execution layer. [e] Trusted hardware. [f] Single-round, stateless f, see §X.

increases the on-chain cost as it retains multiple evaluation circuits whose input labels need to be revealed.

Equation (5) assumes that the external value of information leakage has a finite upper bound. No collateral can provide deterrence when α is unbounded or when an adversary values harm outside the modeled utility. Even with bounded α, the mechanism may be impractical if the required collateral is prohibitively large, and deterrence is weaker for one-shot identities with no reputational loss ($R = 0$). These regimes require a lower transaction notional, a larger n, or a stronger malicious-security construction.

In multi-user applications such as BitMarket, an aborting or unresponsive participant can delay or block the completion of the protocol even when such behavior is economically irrational. A threshold authorization policy for the Close transaction can permit completion without every buyer's participation, thereby reducing griefing and delays caused by partial participation.

B. Liveness under adverse network conditions

Signatures prevent a party from inventing a payout not authorized at setup, but they do not guarantee that an authorized transaction confirms before its competing timeout. Congestion and ordinary fee spikes are absorbed through child-pays-for-parent (CPFP) fee bumping. Each pre-signed transaction carries a dedicated fee-bump output, and at broadcast either party attaches a child transaction that spends this output with a high fee. Miners must confirm the parent to collect the child's fee, so the child's fee rate lifts both transactions into a block. The fee bid therefore lives in the child, and the parent's transaction identifier and its pre-signed signatures stay unchanged. Targeted miner censorship withholds a transaction regardless of the offered fee, so no fee bid can counter it.

A deployment chooses its confirmation depth and timelocks to cover the assumed inclusion delay, fee ceiling, and reorganization margin, and acts on a transaction only after that depth. Reorganizations within the margin are handled by waiting and rebroadcasting, so they delay settlement without changing any allocation. If the network exceeds the assumed delay, fee ceiling, or reorganization margin, BitPriv makes no liveness guarantee, and a dependent transaction chain may need restarting. A party censored beyond a relative timelock can lose the corresponding branch, turning a liveness failure into a monetary one. Inclusion, fee, and reorganization behavior are inherited Bitcoin liveness assumptions, not cryptographic properties of GHTLC. Because the fee ceiling is the one assumption with a direct price, Table XI reports every full pessimistic path from 3 to 300 sat/vB. The 3 sat/vB base is conservative for this setting, because every

pessimistic-path transaction faces a challenge window of days rather than a next-block target. Over the year to August 2026, the per-block median feerate stayed below 2 sat/vB outside brief congestion episodes [97]. Half the transactions in a typical block therefore paid under two thirds of the base rate, even before the multi-day window slack.

C. Privacy scope and coalitions

BitMarket keeps quotes hidden until the on-chain transaction graph fixes the auction outcome. Once buyers reveal the evaluation-circuit labels needed for settlement, the Garbler can match them to the labels it created and reconstruct every quote. The guarantee therefore covers pre-settlement front-running, the economically relevant phase, while post-settlement secrecy and transaction-graph privacy fall outside its scope. The auction is a two-party computation between the Garbler and a logical group of evaluators, so privacy among evaluators is not part of the model. A coalition of evaluators that gathers enough labels can therefore learn more than the public output.

A coalition could also recover a slashed member's penalty through the redistribution shares its members receive, making deviations cheap. Appendix G-D models the Garbler and colluding solvers as one adversary and derives the burn fraction that makes every such deviation a net loss. When the coalition's gain exceeds the slashed penalty, no burn fraction suffices. Full multi-party privacy would require distributed label generation or a general MPC construction, which is outside this work.

D. Scalability

The two applications stress different parts of setup. BitMarket is dominated by the pre-signed transaction graph, whose template count is exponential in the number of output bits. BitBridge settlement has a one-bit output, so pre-signing is negligible and garbling and committing the verifier dominate. The garbled circuits must cross the setup channel to Alice and remain available through cut-and-choose, reaching 1,390 GB per solver at n=16 (Table X). This figure is a property of the public Groth16 verifier, whose pairing checks dominate its AND-gate count, rather than of BitPriv itself: with the designated-verifier circuit discussed below it is 5.2 GB. Aggregate retained material scales linearly in the solver count, at 5.56 TB for four solvers and 11.12 TB for eight (20.6 and 41.2 GB respectively for the designated-verifier circuit), while solvers prepare concurrently and off the user's critical path. The choice of n balances these setup costs against the detection probability and required stake of Figure 3.

Our primary settlement instantiation uses the public Groth16 verifier of [61], the circuit that BitVM3 [5] also garbles, so the per-circuit garbling time and material in Table X are consistent with those of a BitVM3 instance. BitPriv adds the free parameter n and the 16-byte per-gate commitment of Appendix I-D, and it avoids the heavy malicious cut-and-choose of BitVM3 using appropriate collateral. The choice is not binding, because BitPriv requires only deterministic regeneration from a seed (§V), so any verifier circuit can be substituted. To show what this buys, we measured the storage and input width of a designated-verifier circuit [98], [40], defined over a binary curve where almost every gate is a free XOR, and report both in Figure 5. It has 6,709,930 non-free gates against 2.72 billion. Its trapdoor is Alice's secret and must reach the circuit by oblivious transfer, so the settlement instance is no longer privacy-free and each AND gate costs two ciphertexts rather than one, which with the per-gate commitment makes a 48-byte leaf. Even so, the solver's material falls from 347.5 to 1.29 GB at n=4. Its proof is 2,095 bits against 1,024, and RS reveals the Garbler's input labels, so the settlement liveness path grows from 25,680 to 49,974 vB. Alice must additionally generate the SRS and prove it well-formed before the solver fronts funds, a cost that [98] reports as dominant and that we do not count. Argo MAC [75] and Duty-Free Bits [76] reduce the garbled-material footprint further, by a reported order of magnitude, and compose with either choice of verifier.

E. Scope and applications

Our construction targets single-round, stateless two-party computation. BitMarket and BitBridge instantiate two applications of the primitive. BitPriv can be extended to multiple rounds by state chaining, which propagates a committed state root from one instance to the next using the cross-instance Lamport binding already present in BitBridge. One round's closing distribution becomes the committed input for the following round. Formalizing a stateful application such as a "BitAMM" using this mechanism is future work.

BitBridge **Pseudocode, Part I: Setup**

Setup

Public parameters. The following are fixed during setup: a_{btc}, a_{addr}, the destination-chain asset, the clearing rule, deadlines $\Delta_{\mathsf{disp}}, \Delta_{\mathsf{rev}}, \Delta_{\mathsf{handoff}}, \Delta_{\mathsf{settle}}$, the number of auction circuits n, and the number of settlement circuits n'.

Alice

1) Choose private reserve price p_{reserve}.
2) Define the auction circuit $f_{\mathsf{auc}}(p_{\mathsf{reserve}}, b_1, \ldots, b_t) = \mathsf{AuctionCircuit}(p_{\mathsf{reserve}}, b_1, \ldots, b_t)$.
3) For each $j \in [n]$, sample a fresh seed $\mathsf{seed}_j \in \{0,1\}^\kappa$ and garble f_{auc}: $(F_j, e_j, d_j) \leftarrow \mathsf{Gb}(1^\kappa, f_{\mathsf{auc}}, \mathsf{seed}_j)$. The output labels of each F_j on the bits encoding (w, a_{eth}) implicitly define a per-circuit Lamport key $\ell^{(j)}_{\mathsf{auc}}$: producing the labels for a tuple is equivalent to producing $\sigma_{\ell^{(j)}_{\mathsf{auc}}}$ on that tuple.
4) For each solver S_i and each auction circuit F_j, run OT so that S_i obtains the input labels corresponding to its quote b_i.
5) Receive from each solver S_i the fresh Lamport key $\ell^{(i)}_b$ and the per-instance keys $\{\ell^{(i,k)}_{\mathsf{stmt}}\}_{k\in[n']}$, where $\ell^{(i,k)}_{\mathsf{stmt}}$ is defined by the public-statement input-label commitments of S_i's k-th settlement circuit.
6) For each solver S_i in parallel, jointly pre-sign the settlement transaction graph for S_i's pre-committed settlement instance, hardcoding $\ell^{(i)}_b$ in the locking scripts of both the auction Close branch for S_i and the settlement RS of S_i's instance. The hardcoding of $\{\ell^{(i,k)}_{\mathsf{stmt}}\}_{k\in[n']}$ in settlement RS is implicit: the script's input-label-commitment checks fix $\ell^{(i,k)}_{\mathsf{stmt}}$ for each circuit.
7) Lock a_{btc} BTC and Alice's dispute collateral.

Solver S_i

1) Choose private quote b_i.
2) For each auction circuit F_j, receive through OT the input labels corresponding to b_i.
3) Generate a fresh Lamport keypair $(\mathsf{sk}^{(i)}_b, \ell^{(i)}_b) \leftarrow \mathsf{LamportGen}(1^\kappa)$.
4) Prepare a privacy-free settlement $\mathsf{BitPriv}$ instance with Alice in which S_i is Garbler. For each $k \in [n']$, sample a fresh seed $\mathsf{seed}_{i,k} \in \{0,1\}^\kappa$ and garble: $(F_{\mathsf{settle},i,k}, e_{i,k}, d_{i,k}) \leftarrow \mathsf{Gb}(1^\kappa, \Phi_{\mathsf{settle}}, \mathsf{seed}_{i,k})$. The public-statement input-label commitments of the k-th garbled copy implicitly define a Lamport key $\ell^{(i,k)}_{\mathsf{stmt}}$: revealing the public-statement labels for a statement ϕ at settlement RS is equivalent to producing $\sigma_{\ell^{(i,k)}_{\mathsf{stmt}}}(\phi)$. The key $\ell^{(i)}_b$ is a separate, freshly generated bridge key used in the locking scripts of the auction Close branch and settlement RS.
5) Send $\ell^{(i)}_b$ and $\{\ell^{(i,k)}_{\mathsf{stmt}}\}_{k\in[n']}$ to Alice, together with commitments $\{\mathsf{Com}(\mathsf{seed}_{i,k})\}_{k\in[n']}$.
6) Post the cut-and-choose commitments for the n' settlement circuits and jointly pre-sign the settlement transaction graph with Alice.
7) Lock an input-reveal deposit, dispute collateral, and execution collateral.

Fig. 10: BitBridge pseudocode, Part I. Setup prepares the private multi-party auction instance and one privacy-free settlement instance per solver. Each garbled circuit is produced from a fresh seed via $\mathsf{Gb}(1^\kappa, \cdot, \mathsf{seed})$. The seeds for check circuits are revealed during cut-and-choose for verification. Each solver also contributes a fresh bridge key $\ell^{(i)}_b$ (hardcoded into the auction Close and settlement RS scripts) and per-circuit statement keys $\{\ell^{(i,k)}_{\mathsf{stmt}}\}_{k\in[n']}$ (implicitly fixed by the input-label commitments of the settlement garbled circuits). Continued in Fig. 11.

BitBridge **Pseudocode, Part II: Auction Execution**

Auction Execution

Solvers

1) Jointly choose an auction evaluation index $m \in [n]$.
2) Broadcast the auction CEI transaction, whose witness commits to m via a Lamport signature on m under the auction-side evaluation-index key (the analogue of ℓ_m in base $\mathsf{BitPriv}$, Section V, S3), and wait for Alice to broadcast the auction RS transaction.

Alice

1) After auction CEI confirms, broadcast the auction RS transaction.
2) The auction RS witness contains the seeds for all check circuits $j \neq m$ and Alice's input labels for the evaluation circuit F_m.

Solver S_i (dispute window and reveal)

1) During $[0, \Delta_{\mathsf{disp}}]$, verify Alice's revealed check-circuit seeds, commitments, and OT consistency data.
2) If verification fails, spend the dispute leaf for S_i and initiate a BitVM dispute for Φ_{mis}.
3) If no dispute is opened before Δ_{disp}, wait until the transition leaf is spendable.
4) Spend the transition leaf, or observe another party spending it. The transition creates one input-reveal UTXO per solver and one auction-output handoff UTXO.
5) Spend S_i's input-reveal UTXO by revealing its input labels for F_m received through OT at Setup. If S_i does not reveal before Δ_{rev}, it forfeits its input-reveal deposit.

Any solver (off-chain evaluation)

1) After all required solver labels are revealed, aggregate the input labels for F_m.
2) Evaluate the auction circuit: $z \leftarrow \mathsf{Ev}(F_m, L_{\mathit{Preserve}}, L_{b_1}, \ldots, L_{b_t})$.
3) If $z = \perp$: spend the no-match branch of the auction-output handoff UTXO using the $\perp$ output labels, refund Alice, release the solvers' execution collateral, and **return** $\perp$.
4) Otherwise parse $z = (w, a_{\mathsf{eth}})$. The output labels for (w, a_{eth}) constitute $\sigma_{\ell_{\mathsf{auc}}^{(m)}}(w, a_{\mathsf{eth}})$.

Winning solver S_w (handoff)

1) Produce a Lamport signature $\sigma_{\ell_b^{(w)}}(w, a_{\mathsf{eth}})$ under S_w's key $\ell_b^{(w)}$ (using $\mathsf{sk}_b^{(w)}$).
2) Spend the winning branch of the auction-output handoff UTXO with witness $(\sigma_{\ell_{\mathsf{auc}}^{(m)}}(w, a_{\mathsf{eth}}), \sigma_{\ell_b^{(w)}}(w', a'_{\mathsf{eth}}))$. This is a different pre-signed transaction per solver that additionally requires the signature of the solver so punishments can be attributed.
3) The handoff transaction's output funds the pre-committed settlement UTXO.
4) **return** (w, a_{eth}).

Alice or any solver (handoff timeout)

1) If S_w has not spent the winning branch by $\Delta_{\mathsf{handoff}}$, spend the handoff-timeout branch for (w, a_{eth}), whose witness is the auction-output signature $\sigma_{\ell_{\mathsf{auc}}^{(m)}}(w, a_{\mathsf{eth}})$ alone. Every party holds it after off-chain evaluation, so the path needs no cooperation from S_w.
2) Refund Alice's a_{btc}, slash S_w's execution collateral, release the remaining solvers' execution collateral, and **return** $\perp$.

Auction Close (handoff) script

1) Verify the auction-output Lamport signature: $\mathsf{LamportVerify}(\ell_{\mathsf{auc}}^{(m)}, (w, a_{\mathsf{eth}}), \sigma_{\ell_{\mathsf{auc}}^{(m)}}) = 1$.
2) Verify S_w's key signature: $\mathsf{LamportVerify}(\ell_b^{(w)}, (w', a'_{\mathsf{eth}}), \sigma_{\ell_b^{(w)}}) = 1$.
3) Cross-key equality: $\mathsf{Equal}((w, a_{\mathsf{eth}}), (w', a'_{\mathsf{eth}})) = 1$.
4) If any check fails, reject the spend.

Auction Close (handoff timeout) script

1) Verify that $\Delta_{\mathsf{handoff}}$ has elapsed since the transition.
2) Verify the auction-output Lamport signature: $\mathsf{LamportVerify}(\ell_{\mathsf{auc}}^{(m)}, (w, a_{\mathsf{eth}}), \sigma_{\ell_{\mathsf{auc}}^{(m)}}) = 1$. This fixes w, so the slash is attributable without S_w's signature.
3) If any check fails, reject the spend.

Fig. 11: BitBridge pseudocode, Part II. Auction execution selects a winning solver and amount (or returns $\perp$). The auction-Close (handoff) script enforces cross-key equality between the auction-output signature and S_w's $\ell_b^{(w)}$ signature, binding S_w to the auction outcome under its own key. If S_w never takes that branch, the handoff-timeout script refunds Alice and slashes S_w's execution collateral after $\Delta_{\mathsf{handoff}}$, so declining the handoff after learning the clearing price is not a costless option. Continued from Fig. 10 and continued in Fig. 12.

BitBridge **Pseudocode, Part IIIa: Settlement Execution**

Settlement Execution

Alice (settlement CEI)

1) Act as Evaluator in S_w's settlement instance.
2) Choose a settlement evaluation index $m' \in [n']$.
3) Broadcast the settlement CEI transaction, whose witness contains a Lamport signature on m' under the settlement instance's evaluation-index key (analogous to ℓ_m in base BitPriv, Section V, S3).
4) Wait for S_w to broadcast the settlement RS transaction.

Winning solver S_w (settlement RS)

1) Act as Garbler in the activated settlement instance.
2) Choose the public statement ϕ on which the settlement verifier will run. Honest behavior requires $\phi = (w, a_{\mathsf{eth}})$.
3) After settlement CEI confirms, broadcast the settlement RS transaction. Its witness contains the seeds for all settlement check circuits $j \neq m'$, the public-statement labels for ϕ (which by the wiring fixed at setup are equivalent to $\sigma_{\ell_{\mathsf{stmt}}^{(w,m')}}(\phi)$), and an explicit Lamport signature $\sigma_{\ell_b^{(w)}}(\phi_2)$ under the bridge key $\ell_b^{(w)}$.

Settlement RS script

1) Verify the standard BitPriv checks for settlement RS: multisignature, the Lamport signature on the settlement evaluation index m', and the cut-and-choose seed/label openings against the commitments fixed at setup.
2) Verify that the public-statement labels open under the per-circuit input-label commitments fixed at setup (implicitly verifying $\sigma_{\ell_{\mathsf{stmt}}^{(w,m')}}(\phi_3)$, where ϕ_3 is the statement encoded by the revealed labels).

3) Verify the explicit bridge-key signature: $\mathsf{LamportVerify}(\ell_b^{(w)}, \phi_2, \sigma_{\ell_b^{(w)}}) = 1$.
4) Settlement-side equality: $\mathsf{Equal}(\phi_2, \phi_3) = 1$.
5) If any check fails, reject the spend.

Alice (equivocation enforcement)

1) If the settlement RS witness contains $\sigma_{\ell_b^{(w)}}(\phi_2)$ with $\phi_2 \neq (w, a_{\mathsf{eth}})$:
2) S_w has signed two distinct messages under the one-time key $\ell_b^{(w)}$, namely (w, a_{eth}) in the auction Close witness and ϕ_2 in the settlement RS witness.
3) Construct a fraud proof from the conflicting Lamport preimages and submit it to spend S_w's equivocation-slashing leaf, recovering Alice's BTC and slashing S_w's execution collateral. **return** $\perp$.

Fig. 12: BitBridge pseudocode, Part IIIa. The settlement RS script verifies the standard BitPriv checks plus settlement-side equality between the explicit bridge-key signature $\sigma_{\ell_b^{(w)}}(\phi_2)$ and the public-statement labels, implicitly $\sigma_{\ell_{\mathsf{stmt}}^{(w,m')}}(\phi_3)$. If S_w commits to a statement $\phi_2 \neq (w, a_{\mathsf{eth}})$ at settlement RS, the conflicting $\ell_b^{(w)}$ preimages already published at auction Close form a slashable equivocation fraud proof. Continued from Fig. 11.

BitBridge **Pseudocode, Part IIIb: Settlement Evaluation and Close**

Settlement Evaluation and Close

Alice (settlement dispute and evaluation)

1) During the settlement dispute window, verify the opened settlement check circuits and commitments.
2) If verification fails, open a BitVM dispute against S_w (which slashes S_w's dispute collateral).
3) If no dispute is opened, wait for S_w to deliver the destination-chain asset and produce a SNARK proof π.
4) Evaluate the settlement circuit: $y \leftarrow \mathsf{Ev}(F_{\mathsf{settle},m'}, L_{(w,a_{\mathsf{eth}})}, L_\pi)$, then decode to True/False.
5) If $y = \mathsf{False}$, obtain the output label decoding to False, spend the rejecting branch, recover Alice's BTC, slash S_w's execution collateral, and **return** $\perp$.
6) If S_w does not complete settlement before Δ_{settle}, spend the timeout path, recover Alice's BTC, slash S_w's execution collateral, and **return** $\perp$.

Winning solver S_w (settlement Close)

1) Deliver a_{eth} to a_{addr} on the destination chain.
2) Generate a SNARK proof π for the statement $\phi = (w, a_{\mathsf{eth}})$.
3) If $\Phi_{\mathsf{settle}}(w, a_{\mathsf{eth}}; \pi) = \mathsf{True}$, spend the accepting settlement Close branch after the prescribed timelock.
4) Claim Alice's BTC and recover the execution collateral.
5) **return** True.

Fig. 13: BitBridge pseudocode, Part IIIb. Alice checks the settlement instance during the dispute window and evaluates the selected settlement circuit once S_w provides the destination-chain delivery proof. Settlement Close releases Alice's BTC to S_w if and only if the SNARK-verified delivery predicate accepts. Continued from Fig. 12.