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

Ioannis Alexopoulos[1], Zeta Avarikioti[1,2], Paul Gerhart[1],
Matteo Maffei[1], Dominique Schröder[1,3]

[1]*TU Wien* [2]*Common Prefix* [3]*Friedrich-Alexander-Universität Erlangen-Nürnberg*

Abstract

Bitcoin secures over a trillion dollars in assets but remains largely absent from decentralized finance (DeFi) due to its restrictive scripting language. The emergence of BitVM, which enables verification of arbitrary off-chain computations via on-chain fraud proofs, opens the door to expressive Bitcoin-native applications without altering consensus rules. A key challenge for smart contracts executed on a public blockchain, however, is the privacy of data: for instance, bid privacy is crucial in auctions and transaction privacy is leveraged in MEV-mitigation techniques such as proposer-builder separation. While different solutions have been recently proposed for Ethereum, these are not applicable to Bitcoin.

In this work, we present BitPriv, the first Bitcoin-compatible protocol to condition payments based on the outcome of a secure two-party computation (2PC). The key idea is to let parties lock collateral on-chain and to evaluate a garbled circuit off-chain: a cut-and-choose mechanism deters misbehavior and any violation can be proven and punished on-chain via BitVM. This design achieves security against rational adversaries, ensuring that deviation is irrational under financial penalties.

We showcase the new class of applications enabled by BitPriv as well as evaluate its performance through a privacy-preserving double-blind marketplace in Bitcoin. In the optimistic case, settlement requires only two transactions and under $3 in fees; disputes are more expensive (≈ $507) with their cost tied to the specific BitVM implementation, but their mere feasibility acts as a strong deterrent. BitPriv provides a blueprint for building enforceable, privacy-preserving DeFi primitives on Bitcoin without trusted hardware, sidechains, or protocol changes.

1 Introduction

Bitcoin secures over a trillion dollars in assets, making it the most trusted and widely held cryptocurrency. The common belief that Bitcoin's limited scripting language can only support payments and simple applications [17] without requiring intrinsic changes [15] has been confuted by the recent advent of BitVM [7, 67], which has enabled for the first time to build and run expressive DeFi applications within Bitcoin's native environment. BitVM allows two parties to execute a program off-chain and enforce its output on-chain, without altering consensus rules or introducing new trust assumptions. However, BitVM is inherently limited to computations where all inputs are publicly known to both parties. This makes it unsuitable for decentralized applications where inputs must remain hidden, since revealing them can enable financial exploits, such as front-running and back-running in marketplaces, auctions, and exchanges (aka settings vulnerable to maximal extractable value (MEV) [34]). This limitation naturally opens a new frontier: *can we unlock privacy-preserving and financially secure (DeFi) applications on Bitcoin without additional trust assumptions or consensus changes?*

Prior works on privacy-preserving applications on blockchains highlight both the promise and the challenge of this task. On expressive platforms like Ethereum, a wide range of protocols enable privacy-preserving computation between two (or more) parties (2PC/MPC) [12, 13, 18, 24, 44, 53, 60, 73–76, 81, 85]. Several of these works leverage MPC in the context of DeFi applications, e.g., auctions [12, 18, 60, 85], DEXes [18, 24, 76]. However, all these works rely on Ethereum's rich smart-contract functionality (e.g., for verification of zero-knowledge proofs), which Bitcoin does not natively support.

Prior Bitcoin-relevant approaches to secure computation fall into three main categories: First, early protocols focused on specific applications like lotteries or poker, or assumed extended script support for practical realization over the Bitcoin network, preventing them from providing a general-purpose 2PC primitive [2, 3, 22, 61, 63]. Second, theoretical frameworks for fair or insured MPC, while efficient, rely on stateful contracts and other abstractions not natively supported by Bitcoin Script [19, 21, 23, 27, 36, 56, 62, 64]. Third, solutions that achieve privacy do so by moving execution off-chain to systems like other blockchains or Trusted Execution Environments (TEEs), thereby introducing external trust assumptions

and departing from Bitcoin's native security model [53,85,87].

In this work, we close this gap. At the core of our approach lies BitPriv, the first protocol to enforce the outcome of a secure two-party computation (2PC) entirely on Bitcoin. BitPriv allows two parties to lock collateral and evaluate an agreed-upon function off-chain using a garbled circuit protocol. A cut-and-choose mechanism deters the circuit generator from misbehaving with a high probability of detection, while Lamport commitments bind the evaluator to a single evaluation circuit, preventing equivocation (e.g., the malicious use of secrets from other checked circuits to claim a fraudulent payout). If any violation is detected, a fraud proof can be generated and enforced on-chain to slash the dishonest party's collateral. To enforce fraud proofs within Bitcoin's restricted scripting language, BitPriv leverages BitVM. This design realizes *publicly verifiable covert security* [4, 51, 87]: a rational adversary, facing high detection probability and certain financial loss if caught, is deterred from deviation.

BitPriv enables for the first time the design of Bitcoin-native DeFi applications that are both fair and private, eliminating, e.g., the information leaks that drive maximal extractable value (MEV). We showcase this through BitMarket, the first double-blind Bitcoin marketplace, where bids and asks remain hidden unless a match occurs, ensuring that trades execute fairly or dishonest parties are penalized.

BitPriv is not only an expressive but also practical design tool. In the optimistic case, settlement completes with only two Bitcoin transactions and incurs less than $3 in fees at typical rates. Even in the pessimistic case of an on-chain dispute, enforcement remains feasible (around $507 with BitVMX [65], which are paid out of the misbehaving party's collateral) ensuring that the threat of slashing is credible. These costs are well within the range needed to deter rational adversaries, making BitPriv suitable for real-world deployment.

Contributions. This paper makes the following contributions:

- We design BitPriv, the first protocol to enforce the outcome of a 2PC entirely within Bitcoin's native environment, without consensus changes or additional trust assumptions.
- We prove BitPriv secure in the real/ideal paradigm, analyze its deterrence for rational adversaries, and formalize compliance conditions.
- We build a proof-of-concept, measure on-chain and off-chain costs, and show that BitPriv is efficient in both optimistic and pessimistic cases.
- We design BitMarket, a double-blind Bitcoin marketplace with no MEV leakage, demonstrating how BitPriv enables new classes of DeFi applications on Bitcoin.

2 Model

In this section, we informally present our model, assumptions, and protocol goals. The formal model can be found in Appendix A.

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 [43].

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.

Threat Model. Either party may be corrupted. We assume adversaries are polynomial-time bounded, but *rational:* they deviate only when the expected gain outweighs the risk of detection and financial 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 Section A using the notion of *covert adversaries* [6].

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 agreed output and publicly observable transactions.

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.

3 Background

3.1 Garbled Circuits and Oblivious Transfer

We use Yao-style garbled circuits (GC) [47, 83] with a seeded, reproducible garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De}, \mathsf{Ve})$ following [87]. Given f, security parameter κ, and a seed, Gb deterministically outputs (F, e, d) where F is the garbled circuit and e, d are the encoding and decoding information respectively. Inputs x are encoded as *labels* $X \leftarrow \mathsf{En}(e, x)$, the output is computed as $Y \leftarrow \mathsf{Ev}(F, X)$, and decoded to plain-text as $y \leftarrow \mathsf{De}(d, Y)$; $\mathsf{Ve}(f, \text{seed}, F, d)$ checks correct generation of the circuit. We rely on standard properties (correctness, privacy, authenticity, verifiability). Low-level optimizations (e.g., Free-XOR [59], Half-Gates [70, 84]) are orthogonal and compatible with our protocol as long as they keep the process deterministic. To enable secure two party computation, one party acts as the *garbler*, generating the circuit and therefore knows the encoding and decoding information. The other party, the *evaluator*, obtains her input labels via OT[1] [20, 33], which lets her learn exactly the labels consistent with her input bits while revealing no information to the garbler. After OT, the evaluator holds the complete label set $X = (X_G \parallel X_E)$ and evaluates the circuit as $Y \leftarrow \mathsf{Ev}(F, X)$. The final plaintext output is obtained in one of two ways: either the evaluator returns the encoded output Y to the garbler for decoding, or the garbler provides the necessary decoding information to the evaluator.

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.

3.2 Cryptographic primitives

We use a hash function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{2\kappa}$ (e.g., for $\kappa = 128$, a 256-bit digest) for the construction of a non-interactive commitment scheme Com as follows: the commitment is $\mathsf{Com}(m; r) := \mathcal{H}(m \parallel r)$, where the decommitment is simply $m \parallel r$. In the RO model, this is binding and computationally hiding. For messages with high min-entropy, like a garbled label L, we use a saltless form $\mathsf{Com}(L) := \mathcal{H}(L)$. In practice, we instantiate the commitment scheme with a standard cryptographic hash function. On Bitcoin, this is implemented in script using native opcodes such as `OP_SHA256` (32-byte digests) or `OP_HASH160` (20-byte digests). To ensure deterministic reproducibility for on-chain verification, all randomness required by the protocol is derived from a private seed s. This is done by expanding s via a PRF in counter (CTR) mode to obtain a stream of pseudorandom blocks, from which coins are drawn as needed. For a transcript $\tau = (m_1, m_2, \ldots)$ of public messages exchanged during a protocol execution, its *transcript hash sequence* is defined as $\mathcal{H}(\tau) := \big(\mathcal{H}(m_1), \mathcal{H}(m_2), \ldots\big)$.

[1] We use OT-extension [52] to amortize public-key costs.

Example Transaction (1-in/2-out)	
Inputs	(1) $(\mathsf{txid}_1, 0, \varphi_1)$
Outputs	(1) (4 BTC, $\langle\, \psi_1 \,\rangle$) (2) (2 BTC, ψ_2)
Witness	(1) σ_A

Table 1: UTXO notation for the example transaction.

3.3 UTXO Model

In Bitcoin's UTXO-based model, coins are held in Unspent Transaction Outputs (UTXOs). A transaction's core body ($[\![\mathsf{tx}]\!]$) consists of *inputs*, which point to the UTXOs being consumed, and *outputs*, which create new UTXOs. Each output[2] specifies a coin value and a new locking script. A complete transaction is the pair of its body and a *witness*, which provides the data required to satisfy the locking scripts of the consumed inputs.

Standard locking script functionalities include signature checks (`CheckSig`,`CheckMultiSig`), absolute and relative timelocks, and hash-preimage locks. Taproot, introduced in [80], enhances privacy and efficiency by committing to multiple, alternative spending conditions in a Merkle tree structure. To spend a Taproot output, a party reveals only the specific script leaf they are satisfying and a Merkle path proving its inclusion in the tree. We denote a Taproot output as $\langle \cdot \rangle$.

For concreteness, consider a 1-input/2-output transaction (Table 1) involving parties A, B holding public keys pk_A and pk_B respectively. The transaction spends the first (i.e., with index 0) output of txid_1[3] which is locked behind $\varphi_1 \equiv \mathtt{CheckSig}_{\mathsf{pk}_A}$ with witness σ_A.
Output (1) (4 BTC) is a Taproot output committing to ψ_1 with two script-path spends: $\big(\mathtt{CheckSig}_{\mathsf{pk}_A} \wedge \mathrm{Hashlock}(h)\big)$ or $\big(\mathrm{RelTimelock}(10) \wedge \mathtt{CheckMultiSig}_{\mathsf{pk}_A, \mathsf{pk}_B}\big)$. Hence, A can spend by revealing a preimage of h, or both parties need to co-sign to spend after a 10-block delay. Output 2 (2 BTC) requires $B's$ signature to be spent i.e., $\psi_2 \equiv \mathtt{CheckSig}_{\mathsf{pk}_B}$.

Pre-Signed Transaction Graphs as Covenants. Bitcoin *covenants* [14] 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 [7, 67]. Since transaction identi-

[2] In every Bitcoin transaction, the total value of all outputs must not exceed the total value of the inputs; the remainder (inputs minus outputs) is implicitly paid as the transaction fee.

[3] The transaction's unique identifier, txid, is the cryptographic hash of its body: $\mathsf{txid} := \mathcal{H}([\![\mathsf{tx}]\!])$.

fiers are deterministic prior to broadcast[4], 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 m-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.

3.4 BitVM

Bitcoin's intentionally non–Turing-complete Script precludes on-chain verification of arbitrary computations; proposed workarounds [14, 16, 35, 41, 69] either require consensus changes, add significant on-chain cost, or introduce additional trust assumptions. BitVM [7] addresses these challenges by shifting complex computations off-chain while maintaining on-chain verifiability through a dispute resolution mechanism. The initial design of BitVM is a protocol between two parties, a *prover* P and a *verifier* V. At a high level, the parties agree on a program Π and a fund allocation function that maps outputs to spending conditions. The prover claims to have computed Π correctly on some input S_0 and commits to the result. If the verifier accepts the result as correct, settlement proceeds with just two on-chain transactions. If not, the parties engage in an interactive binary search over the $|C|$ execution steps of Π to locate a single disputed operation following the paradigm introduced by Canetti [29][5]. This step is then verified directly using Bitcoin Script, requiring at most $O(\log|C|)$ additional on-chain transactions. The original BitVM protocol has inspired several variants that build on the same paradigm. BitVM2 [67], for example, supports permissionless verification of Groth16 [50] zk-SNARKs, while BitVMX [65] introduces execution of arbitrary programs compiled to RISC-V instructions using an off-chain virtual CPU model. All variants retain the same optimistic architecture: computation is performed off-chain and verified on-chain only in case of dispute, only using Bitcoin's native script. BitVM requires parties to reveal their inputs.

4 Technical Overview

This section introduces the core components and design rationale behind BitPriv. We begin by outlining the protocol goals, followed by a naive strawman construction that exposes key vulnerabilities. These insights incrementally motivate the full protocol described in the next section, where we show how BitPriv resolves these challenges.

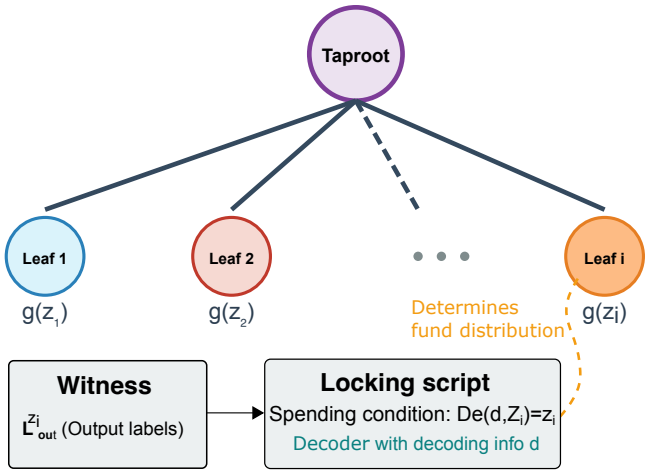


Figure 1: Taptree implementing the *decoder* of the garbling scheme $\mathcal{G}$; only the satisfied leaf is revealed on spend (the witness contains the leaf script and a control block proving inclusion).

4.1 BitPriv Overview

Our protocol allows two parties, P_A and P_B, to securely compute a function[6] $f : \mathcal{I} \times \mathcal{I} \to \mathcal{O}$ while keeping their private inputs x and y secret. The primary goals of the protocol are twofold: **(i)** to enforce an on-chain distribution of funds according to the computed output $z = f(x, y)$, and **(ii)** to ensure that after an initial setup, no party can prevent this distribution without incurring a financial penalty, except with low probability. We formally define these properties in §A.

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 $g : \mathcal{O} \times C_A \times C_B \to C_A \times C_B$ 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).[7]

Our construction is based on *garbled circuits*, hence one party assumes the role of the *evaluator* (E) and the other of the *garbler* (G).

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

[5] This paradigm was first introduced in the blockchain world in [53]

[6] We use f to refer to both the high-level function and the circuit that implements it.

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

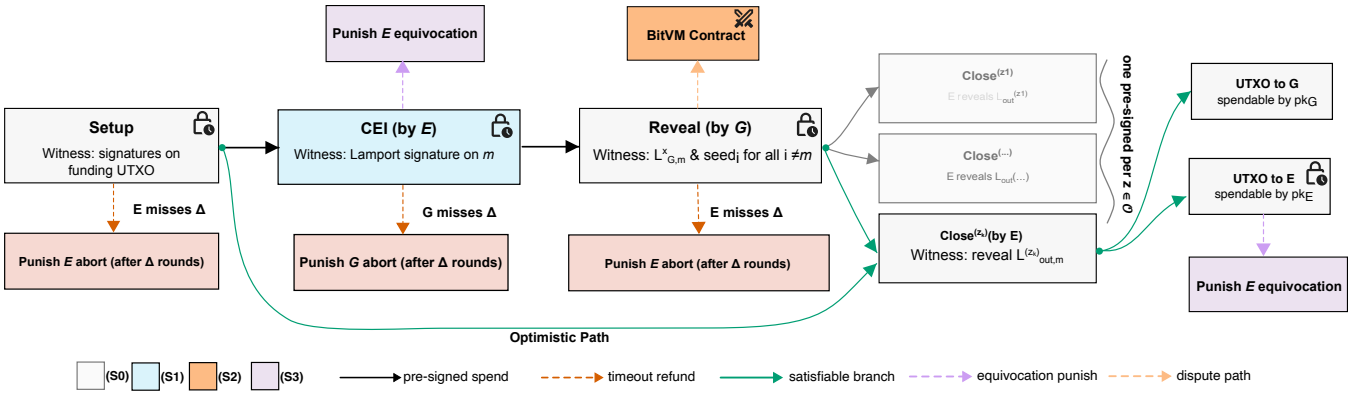


Figure 2: BitPriv protocol flow. The protocol builds on the foundation of (S0), with a cut-and-choose from (S1), the BitVM dispute from (S2) and finally the equivocation punishment of (S3)

(S0) Strawman Construction

A natural starting point is to adapt the Hash Time-Locked Contract (HTLC) flow described in [66] within the garbled circuit framework. However, garbled circuits are only semi-honestly secure, which leads to the following issues. The garbler G garbles f into a circuit F; the evaluator E obtains her input labels L_y via OT. The parties then pre-sign a sequence of Bitcoin transactions:

1. Setup locks the parties' inputs and collateral coins.

2. Reveal lets G spend the Setup output by revealing his input labels L_x within Δ. The resulting output is a Taproot whose taptree represents the *decoder* De of $\mathcal{G}$.

3. $\mathsf{Close}^{(z)}$ a *family* of pre-signed transactions, one template per possible decoded output $z \in O$. After E evaluates F and obtains the output label(s), she can satisfy *exactly one* script-path leaf of the Reveal Taproot thereby making only the corresponding $\mathsf{Close}^{(z)}$ transaction spendable. The spend then allocates funds according to $g(z)$ (see Fig. 1).

Insecurity under a malicious garbler. At pre-sign time, G knows *all* secret labels that act as hash preimages driving the spend conditions (L_x and L_{out}), while E holds only L_y and a non-evaluable F. This asymmetry enables:

Unverifiable computation & allocation: G can supply $F' \neq \mathsf{Gb}(*, f, *)$ so that decoding miscomputes f or leaks information about E's input; *or* keep F correct while choosing the output-decoding map and Taproot script mapping so that the output label(s) L_{out} that E can reveal do *not* correspond to the intended payout $g(z)$. In both cases, E cannot, at setup time, audit the binding from $\{L_{\text{out}}\}$ to payout branches.

Inconsistent labels (input/wire encoding abuse): Because E does not know the encoding map for F, she cannot validate that the labels she receives (via OT for her inputs, directly for G's inputs) are the right wire labels. G can therefore (i) bias the computation (labels for E's chosen y do not decode to y), or (ii) cause evaluation/decoding to fail (no valid pair on some wires), leaving E unable to complete the spend.

Thus E can neither (i) produce a valid witness to enforce an allocation nor (ii) generate public fraud evidence of G's misbehavior. Ultimately, G can either steer the payout in his favor *or* render the circuit unevaluable and recover the *abort punish* value from the timeout branch after the deadline since E is seemingly unresponsive.

(S1) Probabilistic Verification

In our setting, using well-known techniques to convert the protocol into a secure one is prohibitively expensive. Towards our goal, we follow the cut-and-choose approach but tailored to our setting. More precisely, to neutralize the Garbler's informational advantage, the Garbler G generates n distinct garbled circuits and commits to their descriptions and the random seeds used in their creation. To ensure the honest execution of the OT, both parties commit to the public OT transcript including their randomness. The Evaluator E then challenges G to open $n-1$ of the circuits. Using each revealed seed, E deterministically reconstructs the corresponding garbled circuit. Any deviation in a checked circuit is detected with certainty. Under a uniform challenge, the probability of detecting misbehavior is $\frac{n-1}{n}$.

Lack of public verifiability. Although (S1) can detect misbehavior with high probability, it remains insufficient: there must be a mechanism to publicly prove and punish this misbehavior on the blockchain. However, such proofs are typically very large in size, and using a blockchain such as Bitcoin with its limited expressivity to directly validate this proof is practically prohibitive. Therefore, a solution is needed to shift the majority of the proof verification off-chain while maintaining the security guarantees of Bitcoin.

(S2) On-chain Dispute Resolution with BitVM

BitVM as a Black-Box Dispute Hook We use BitVM as a two-party *black-box* dispute mechanism that decides a predicate $\Phi(w, o)$ stating that output o is a correct evaluation of a fixed function f on input w.

BitVM Interface

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

Semantics. 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.setupTx. Once C.setupTx confirms on-chain, Enforce ensures eventual settlement according to Alloc, conditioned on the value of $\Phi(w, o)$.

Hooking into our protocol. During off-chain setup the parties call BitVM.Compile to obtain C; the parties can fund C.setupTx dynamically during execution. In the optimistic case C.setupTx is never broadcast.

Economic intuition. The protocol's security is rooted in economic incentives designed to make deviation irrational. In a dispute, the evaluator (E) acts as a Prover by presenting evidence of the garbler's (G) fault. Triggering the dispute by publishing C.setupTx places all locked funds under the BitVM contract's control. The process is collateral-backed: a valid claim slashes G's collateral, while a false claim slashes E's. This ensures that for rational parties, initiating a baseless dispute is unprofitable, and contesting a valid claim is futile. Furthermore, considering the reputational harm, the expected cost of cheating (which combines the slashed collateral with future lost earnings) far outweighs any potential gain, even with a small $1/n$ probability of the cheat going undetected. This alignment of incentives enforces rational compliance with significantly lower computational and communication overhead compared to schemes offering full (i.e., negligible error probability) security guarantees, as formalized in §6.2.

Ability to spend a *check* circuit Tapleaf. Even under cooperation, cut-and-choose introduces a specific risk: after G reveals seeds for the $n-1$ *check* circuits, E can reconstruct those circuits and obtain their output labels (hash preimages). If the payout taptree is not bound to a single evaluation index, a malicious E could attempt to redeem a script-path branch using labels from a *checked* (non-evaluation) circuit (we refer to this problem as *equivocation*), violating security.

(S3) Preventing Equivocation and an Optimistic Path

We solve this by using a Bitcoin Script-verifiable Lamport one-time signature (OTS) scheme. This irrevocably commits the Evaluator to a single evaluation index m.[8] The OTS provides twofold security: it is unforgeable by the Garbler, and any attempt by the Evaluator to sign a different index reveals cryptographic secrets that serve as a public proof of fraud, triggering an on-chain penalty. This commitment mechanism is the key to the protocol's efficiency. It enables a highly efficient *optimistic path* where honest parties bypass the complex on-chain adjudication entirely, settling the protocol privately with just two transactions. The precise mechanics of this settlement are detailed in §5.2.

5 BitPriv Protocol Description

This section details the full BitPriv protocol, which integrates the design principles motivated in Section 4.1. The full protocol flow and the contribution of each (S0-S3) to the construction is visualized in Figure 2.

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.

5.1 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.

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 $j \in [\text{outbits}]$ (one plaintext output bit), G constructs two vectors: $\mathsf{Output}_j^0 = [\mathsf{Com}(L_{\mathsf{out},i,j}^0)]_{i\in[n]}$ and $\mathsf{Output}_j^1 = [\mathsf{Com}(L_{\mathsf{out},i,j}^1)]_{i\in[n]}$, where $L_{\mathsf{out},i,j}^b$ 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 $(L_{G,i,j}^b, L_{G,i,j}^{1-b})$ for a random bit b, and sends the commitment $\mathsf{Com}(X_{i,j})$ to E.

[8] The Lamport OTS can be replaced by any one-time scheme implementable in Script (e.g., Winternitz/W-OTS(+) [25, 86]).

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^λ) 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 §4.1, they compile this contract:

$$C \leftarrow \mathsf{BitVM.Compile}(\Phi_{\mathsf{mis}}, \mathsf{Alloc}, P_{\mathrm{params}}).$$

They then collaboratively pre-sign the transaction graph, embedding the necessary commitments and C in the spending scripts. The concrete checks, allocation policy and witness format appear in §5.4.

5.2 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.

5.2.1 Optimistic Path (Cooperative settlement)

If both parties are honest, they bypass the on-chain adjudication logic. Over the authenticated off-chain channel, E sends G a Lamport signature[9] σ_m on her chosen evaluation index m. As explained in §4.1, G can safely reveal the secrets for all circuits except F_m: the decommitments to the seeds of the $n-1$ check circuits and the decommitments to G's input-label commitments for those circuits. If E's verification of the check circuits passes and she can evaluate F_m to obtain valid output labels, she spends the valid leaf of the Setup output Taproot with a Close transaction to distribute funds according to $g(f(x,y))$, settling immediately. Because E can attempt to close on a different evaluation index, her part of the funds is locked behind a PunishEquiv timelock, where G can claim them by providing the conflicting preimages in the witness.

[9] A Lamport signature on m binds m and is unforgeable under one-time use; in this role it effectively serves as a commitment to m, so we sometimes refer to it as a "Lamport commitment."

5.2.2 Pessimistic Path (On-Chain Adjudication)

If either party deviates or becomes unresponsive, the protocol defaults to a sequence of on-chain transactions that publicly resolve the outcome.

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

2. RevealSeeds (RS). G must then spend the CEI output by broadcasting a RS transaction. Its witness contains decommitments for (a) all seeds seed_i with $i \neq m$ and (b) 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. Verification and Final Action. Upon observing RS, E verifies the $n-1$ opened circuits and associated transcripts as in §5.4.

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

C.setupTx. If any of G's revealed data is inconsistent, E initiates a dispute by spending the output of RS (via `BVMScript`) with the BitVM contract's funding transaction, C.setupTx. This triggers BitVM.Enforce which handles the distribution of funds according to Alloc (see §5.4).

Timeout$_E$ [10]. 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, the timeout path allows G to claim this penalty amount.[11]

5.3 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`).

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

[10] We use the notation $\mathsf{Timeout}_X$ for the transaction punishing party X for being unresponsive

[11] 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 §6.2.

cooperative Close for each possible output for each possible m (i.e., $n \cdot 2^{\text{outbits}}$ total leaves), and a Timeout_E.

Setup Transaction	
Inputs	(1) $(*, *, \texttt{CheckSig}_{\text{pk}_G})$ (2) $(*, *, \texttt{CheckSig}_{\text{pk}_E})$
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`), Timeout_G, and PunishEquiv path (via `PEScript`). The complete unlocking script is $\texttt{CEIScript} \equiv \texttt{CheckMultiSig}_{\text{pk}_{GE}} \wedge \texttt{LampSigVerify}_{\text{pk}_m}$[12].

CommitEvaluationIndex Transaction	
Inputs	(1) $(\text{Setup}, 0, \texttt{CEIScript})$
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 Timeout_E.

RevealSeeds Transaction	
Inputs	(1) $(\text{CEI}, 0, \texttt{RSScript})$
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 E provides valid output labels $\{Z\}$ for which $\text{De}(Z) = z$ 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) $(\text{RS}, 0, \texttt{CloseScript})$
Outputs	(1) $(\text{value}_G, \texttt{CheckSig}_{\text{pk}_G})$ (2) $(\text{value}_E, \langle \texttt{CheckSig}_{\text{pk}_E} \wedge \text{TL}(\Delta), \texttt{PEScript} \rangle)$
Witness	(1) $(\sigma_{GE}, m, \sigma_m, [\text{outPreimages}])$

From pseudocode to Bitcoin Script. The algorithms we present in G 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.

Optimization. In the original design, the locking script of the Close transaction requires a separate hash preimage for each output label, causing the transaction's witness and script size to scale linearly with the number of output bits. To mitigate this on-chain cost, we introduce two optimizations: i) using the hash of the concatenated labels as a single witness, an approach limited only by Bitcoin's stack size, or ii) providing a single preimage that serves as a cryptographic commitment to the vector of all output labels. Both optimizations reduce the on-chain footprint of Close. This efficiency comes with a trade-off: the verification burden is shifted to the BitVM predicate, which must perform additional computations to validate the commitment's construction against the individual labels of the opened *check* circuits during a dispute.

5.4 Dispute Resolution via BitVM

Predicate for BitVM: Φ_{mis} To instantiate Φ_{mis}, we use the following deterministic subroutines (see G.3 for pseudocode):

- π_{Gb} (Simple Garbler): Given a seed, deterministically generates F, output labels O, and input labels I_G, I_E.
- π_{OT} (OT Transcript Reconstruction): Reconstructs the public message transcript of a given OT instance using seeds of both parties.
- π_ϕ (Message Sender Identification): Given a transcript message, outputs whether the sender was G or E.
- π_{Vc} (Commitment Verification): Returns 1 iff a commitment com opens to a value v.

Given revealed seeds $(\text{seed}^*_G, \text{seed}^*_E)$ for an opened check index i^* and an OT instance λ^*, Φ_{mis} computes:

[12] 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.

1. **Seed Validity:** Verify that the openings seed_G^* and seed_E^* match the initial commitments $\mathsf{Com}(\mathsf{seed}_{G,i^*})$ and $\mathsf{Com}(\mathsf{seed}_{E,i^*})$.

2. **Garbling Correctness:** Re-garble the circuit using seed_G^* and verify that the resulting components match the commitments $\mathsf{Com}(F_{i^*})$, $\mathsf{Com}(d_{i^*})$, and the output label commitments in $\mathbf{Output}_j^b[i^*]$ for all output wires j and bits b.

3. **Input Label Integrity:** Check that the revealed input labels for the Garbler are consistent with the commitments $\mathsf{Com}(X_{i^*,j})$ for all input wires j.

4. **OT Transcript Verification:** Recompute the public OT transcript from the seeds; verify each message against its commitment and attribute any deviation via π_ϕ.

Off-Chain Verification and Dispute Initiation. After the RevealSeeds transaction is broadcast, E locally executes the checks that constitute Φ_{mis} for the $n-1$ opened circuits using the revealed seeds. These checks are efficient and do not require interaction. If any verification fails, E initiates a dispute by publishing $C.\mathsf{setupTx}$ and providing (w,o) such that $\Phi_{\mathsf{mis}}(w,o)=1$; then $\mathsf{BitVM.Enforce}(C;w,o)$ settles per Alloc.

6 BitPriv Security Analysis

6.1 Proof of Security

We define security in the standard real/ideal paradigm, where a real-world protocol $\mathcal{P}$ is compared to an ideal functionality $\mathcal{F}$ that specifies an execution minimizing privacy leakage for honest parties. Indistinguishability between real and ideal executions ensures that any adversary against $\mathcal{P}$ learns no more than this minimal leakage.

We define the security of BitPriv relative to ideal functionality $\mathcal{F}_{\mathsf{BitPriv}}$, which extends the classical publicly verifiable covert (PVC) model by incorporating a *fund allocation function* that specifies how coins initially locked by the parties are distributed according to the computed output. In this model, the garbler can attempt to deviate from the protocol to get information on the evaluator's input and violate the output allocation function while the evaluator, who learns the output first, only has the ability to violate fairness. Crucially, $\mathcal{F}_{\mathsf{BitPriv}}$ guarantees that any deviation by the garbler is detected and punished with probability ϵ. If the garbler's deviation is not detected, which occurs with probability 1-ϵ, $\mathcal{F}_{\mathsf{BitPriv}}$ offers no guarantees to the evaluator. If the parties are honest, the correct allocation is enforced on the underlying ledger functionality; if cheating is detected, $\mathcal{F}_{\mathsf{BitPriv}}$ invokes the dispute-resolution functionality $\mathcal{F}_{\mathsf{BitVM}}$ to penalize the misbehaving party, ensuring that honest participants receive either their correct output or appropriate compensation. Additionally, $\mathcal{F}_{\mathsf{BitPriv}}$ enforces liveness properties, i.e., financial fairness and financial guaranteed output delivery through the ledger functionality $\mathcal{L}$.

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 existentially unforgeable under a chosen-message attack (EUF-CMA). Then* BitPriv *securely realizes $\mathcal{F}_{\mathsf{BitPriv}}$ in the $\mathcal{F}_{BitVM}, \mathcal{L}$-hybrid world.*

Proof. The proof is deferred to D.

6.2 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. We prove the following theorems which formalize the above intuition.

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

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

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:*

$$\gamma + r_E > \beta$$

Proof. The proofs are deferred to E.

7 BitMarket: Blind Auctions on Bitcoin

We exemplify the class of protocols enabled by BitPriv by instantiating a privacy-preserving marketplace using a sealed-bid auction secure against rational adversaries [42]. In this type of auction, the *seller* has a secret ask, i.e., the minimum price he is willing to sell an asset, each of the *buyers*[13] has a secret bid, and the auction winner is determined by the highest bid that exceeds the ask. If there is none, the auction fails. The protocol can be designed so that in case of a success, the winning buyer either pays his own bid (sealed bid first-price auction [38]) or the price of the second highest bidder (sealed-bid second-price or Vickrey auction [79])[14]. We present two models:

[13] The construction can also be adapted to act as a reverse-auction i.e., facilitating a single buyer and multiple sellers.

[14] Sealed-bid second-price auctions are incentive compatible i.e., incentivizes the buyers to bid their true value.

1. A **decentralized model** that slightly adapts BitPriv, accepting that buyers' inputs may be revealed *after* the auction's outcome is determined.

2. A **trusted auctioneer model**, which simplifies the interaction by reducing it to a two-party protocol between the seller and a non-colluding *auctioneer*, to which BitPriv can be directly applied.

7.1 Auction Protocol Design with BitPriv

7.1.1 Decentralized Model

Modifications to BitPriv. The auction involves t *buyers* $B_1, \ldots, B_t$ and one *seller* who acts as the garbler: (i) The seller runs a separate OT with *each* buyer for every circuit, so each B_i obtains labels for their private bid. (ii) After the seller reveals the $n-1$ seeds and their own input labels in RS, buyers must reveal *their* input labels for the fixed index m so that any buyer can aggregate all labels and evaluate F_m off-chain. The on-chain flow is shown in Fig. 3.

Privacy note. Buyers reveal their input labels only *after* the outcome is already determined and any attempt to block its on-chain enforcement comes with collateral slashing. Thus, the seller gains no advantage from seeing bids.

Burning collateral. In a permissionless multi-party setting, participants may collude (e.g., a seller controlling one or more buyers). To make incentives robust to arbitrary coalitions, punishments are *partially burned*: on any verified violation, a fraction β of the slashed amount is sent to a provably unspendable output, and only the remainder is redistributed per policy. This prevents a coalition from recycling penalties internally; for example, a seller–buyer coalition cannot engineer a "punishment" after learning the inputs and then recapture the entire slash.

Reveal output. The RS spend creates *one* Taproot output with the following leaves:

1. **Dispute leaves** (Δ_{disp} window). For each buyer B_i there is a leaf that funds the BitVM setup and opens a seller–buyer dispute. In a valid (honest) challenge, the seller's collateral is divided among the buyers and in a dishonest challenge by B_i, its collateral is divided among the seller and the other buyers. In both cases, the collateral of the parties who are not punished is returned. Because there is a *single* RS Taproot UTXO, at most one dispute can execute, and any honest buyer recovers at least their dispute collateral.

2. **Transition leaf ($\geq \Delta_{\mathsf{disp}}$).** After the dispute window, any of the buyers can spend the output creating the following $t+1$ UTXOs:

   (a) *Per-buyer inputs-reveal UTXOs.* A dedicated output is created for each buyer B_i, enabling them to reclaim an inputs-reveal deposit by providing the hash preimages for the labels received via Oblivious Transfer (OT). To ensure these labels are correctly bound, each buyer proves during setup using a zero-knowledge proof that their commitment[15] corresponds to the labels delivered during the OT protocol. In case of a failure to reveal by the specified deadline, the deposit is redistributed to the seller and the remaining buyers.

   (b) *Extended Close Taproot.* The leaves of the payout tree are replicated per buyer; each buyer co-signs others' copies but keeps the signature for their own. This serves to identify the party who broadcasts the closing transaction as any buyer equivocating on Close pays an equivocation penalty to the rest of the parties. Finally, the seller's payout is additionally locked behind a trade-fulfilment condition to make payment atomic with delivery as explained in §7.2. If the seller fails to fulfill the promise, he suffers a monetary penalty.

Execution sketch. After Setup is confirmed: (1) buyers jointly[16] select m and post CEI with a Lamport signature σ_m; (2) the seller posts RS, revealing the $n-1$ check-circuit seeds and input labels for F_m; (3) during $[0, \Delta_{\mathsf{disp}}]$ any buyer can activate a dispute leaf; if none does, (4) a buyer spends the *transition leaf* to create the per-buyer inputs-reveal UTXOs and the extended Close Taproot; (5) each buyer reveals their input labels on-chain to reclaim their deposit; with the aggregated labels, any buyer evaluates F_m and broadcasts Close.

7.1.2 Simplified Model: Using a Trusted Auctioneer

We can avoid the complexity of buyer-to-buyer input sharing through an alternative construction, which introduces a *trusted auctioneer* as shown in [58], who is assumed not to collude with the seller or any of the buyers. This model effectively reduces the problem to a standard instance of BitPriv in a two party setting, with the auctioneer acting as the proxy between the parties.

7.2 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

[15] The use of per-label salts prevents the seller from reverse-engineering the inputs from these commitments

[16] Distributed randomness generation is an orthogonal problem [32].

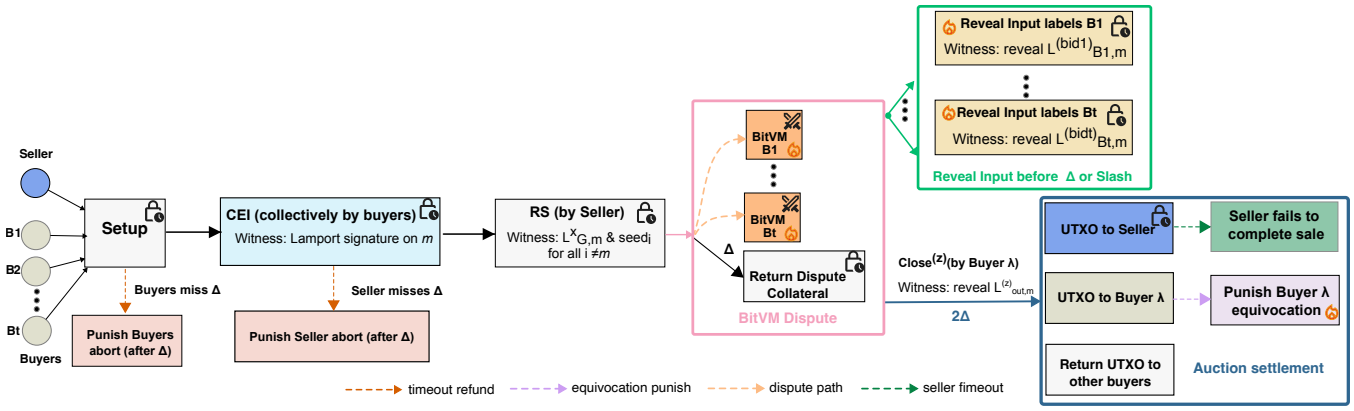


Figure 3: BitMarket decentralized protocol flow. (a) Pink box: buyer–seller dispute; (b) Green box: buyers reveal input labels; (c) Blue box: auction settlement (see § 7.2).

a Bitcoin-compatible *Fair Data Exchange (FDE)* protocol [77]. 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 [53, 71] is a Layer 2[17] 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 [77], this trade can be implemented efficiently if the buyer holds a cryptographic commitment to the specific data they wish to purchase.

Blockchain Assets via BitVM Verification. To settle the transfer of on-chain assets, the auction uses a BitVM instance as a trust-minimized verifier of events on a target chain, including Bitcoin itself [1, 7, 11, 67]. 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 BitVM program that implements a *light client* for the target blockchain [8, 26, 45, 55]. For instance, this allows for verifying the transfer of a Bitcoin *ordinal* (a Bitcoin-based NFT) [72] or interfacing with other chains using protocols like IBC [46]. The result is atomic and secure cross-chain settlement without requiring pre-deployed contracts on the target chain.

Example: future block space fee 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 $[h_0, h_0 + W]$[18], 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.

[17] 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.

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

8 Implementation and Evaluation

The primary contribution of our protocol is its on-chain enforcement mechanism. Our evaluation, therefore, focuses on quantifying the on-chain transaction costs and data footprint of this mechanism on the Bitcoin network. To provide a complete performance profile, we first characterize the costs of the off-chain cryptographic components to demonstrate they are not a practical bottleneck.

8.1 Experimental Setup

System Environment. All off-chain benchmarks were executed on a commercial laptop with an Apple M3 CPU (8 performance cores) and 8 GB of RAM.

Cryptographic Primitives. Our protocol is modular and can be instantiated with any standard garbling scheme. For this evaluation, we use the high-performance half-gate garbling scheme from the EMP-Toolkit [78] and the malicious-secure correlated OT (COT) extension Ferret [82], with the security parameter set to $\kappa = 128$ bits.

8.2 Off-Chain Performance Characterization

Our off-chain benchmarks confirm the cryptographic components are not a performance bottleneck. The dominant cost, Garbled Circuit (GC) generation, remains efficient (≈ 5 ms for $n = 16$ AES circuits), while Oblivious Transfer (OT) costs are negligible ($\approx 25\,\mu$s for a 128-bit input) with throughput exceeding 80×10^6 COTs/sec.

Table 2: Per-transaction scaling and representative costs for BitMarket (n=16, k=16, l_{inbits}=16).

Transaction	Witness size	Leaf script size	Control block size	vB	cost (sat / $)
Setup	$O(1)$	—	—	251	753 / $0.84
CommitEvalIndex (CEI)	$O(\log n)$	$O(\log n)$	$O(k \log n)$	≈ 375	1,125 / $1.26
RevealSeeds (RS)[†]	$O(l_{\text{inbits}}+n)$	$O(l_{\text{inbits}}+n)$	$O(\log n)$	≈ 741	2,223 / $2.49
Close (Unpacked - U)	$O(k+\log n)$	$O(k+\log n)$	$O(k \log n)$	630	1,890 / $2.12
Close (Packed-Concat - PC)	$O(k+\log n)$	$O(\log n)$	$O(k \log n)$	495	1,485 / $1.66
Close (Packed-SECP - PS)	$O(\log n)$	$O(\log n)$	$O(k \log n)$	439	1,317 / $1.47
BitVM1 Dispute [‡]	—	—	—	$\approx 221,344$	$\approx 664,032$ / $\approx$ $744
BitVMX Dispute	—	—	—	$\approx 149,000$	$\approx 448,000$ / $\approx$ $502

[†] For l_{inbits}=16, our byte-accurate simulator yields $\approx$ 741 vB. See Appendix F for scaling.

[‡] Scaled cost estimate based on the trace length of our predicate and the original BitVM proposal [7].

Table 3: End-to-end path costs (including the constant Setup). Representative parameters: n=16, k=16, l_{inbits}=16.

Path	Constituent Txs	Total vB	Total fee (sat / $)
Optimistic	Setup + Close (PS)	690	2,070 / **$2.32**
Liveness Guarantee	Setup + CEI + RS	1,367	4,101 / $4.59
Full Pessimistic (with BitVM1)	Setup + CEI + RS + BitVM1	$\approx 222,711$	$\approx 668,133$ / $\approx$ $748.29
Full Pessimistic (with BitVMX)	Setup + CEI + RS + BitVMX	$\approx 150,367$	$\approx 452,101$ / $\approx$ **$506.59**

8.3 On-Chain Cost Analysis

Methodology. We measure the virtual size (vB) of Taproot (BIP-341) script-path spends for all transactions in our protocol.

Fee Model. We assume a conservative mempool fee rate of $f = 3$ sat/vB and convert to USD at $0.001119924 per satoshi (BTC price on Aug. 18, 2025).

BitMarket Price Resolution. For our BitMarket instantiation, we set the input and output bit length to $l_{\text{inbits}} = k = 16$, which supports $2^{16} = 65{,}536$ distinct price levels. The financial precision is determined by the value of each 'tick'. Adopting a 1,000-satoshi tick provides a price granularity of $\approx$ $1.16, covering a total range of up to 0.65535 BTC ($\approx$ $76,224), a range suitable for many digital goods marketplaces.[19]

Leaf-path overhead. All transactions, except for Setup spend Taproot outputs. A Taproot script-path[20] spend must reveal the leaf script, its witness arguments, and a control block proving membership of the leaf in the Taproot Merkle root.

8.3.1 Fully Optimistic Path

When both parties cooperate, only two transactions are broadcast: a Setup to lock funds and a Close to distribute them. The cost of Setup is practically constant at $0.84, while the cost of Close in the original construction scales with $O(k+\log n)$ in the leaf witness, $O(k+\log n)$ in the leaf script (since all witness elements are separately hashed) and $O(k \log n)$ in the control block (since the tree has $\approx n \cdot 2^k$ leaves). We additionally consider the optimizations defined in §5.3: i) Packed-Concat (PC) where all output labels are concatenated[21] into a single preimage, and ii) Packed-SECP (PS) where a single group element preimage is hashed (only scales with $\log n$ due to the Lamport signature). Scaling tables and plots of Setup against k are deferred to the appendix.

8.3.2 Liveness-Guaranteeing Path

If the optimistic path fails, for example because G refuses to share the requested seeds, the parties need to publish the transactions that enforce the logic on-chain: CommitEvaluationIndex followed by RevealSeeds. The cost of CEI is modest ($\approx$ 350–400 vB) and scales logarithmically with n. The dominant cost is RS, whose witness contains $n-1$ seeds and l_{inbits} input-labels. Example costs are summarized in Tables 2 and 3 and further evaluation results are in the appendix (§F).

8.3.3 Pessimistic Path (Dispute Resolution)

Because BitPriv treats BitVM as a black box, our on-chain dispute cost is *parametric* in the underlying implementation. We therefore estimate costs from *public baselines* for BitVM1 [7] and BitVMX [65]. To estimate the on-chain cost

[19] At BTC$\approx$ $116,310 on Aug. 18, 2025, 1,000 sats $\approx$ $1.163. A 100-sat tick ($\approx$ $0.116) would still cover a range of $\approx$ $7,622.

[20] We disable the key path as described in [80], so all spends must reveal a Tapleaf and control block.

[21] This is only possible when the total concatenated preimage is less than 520 bytes i.e., the maximum stack element size

of a dispute we first profile our misbehavior predicate. As a proxy, we use the `emp-toolkit` [78] implementation of Yao's Millionaire Problem (32-bit inputs). Compiled with default optimization levels and pinned to a single core on an 11th Gen Intel i7-1165G7 CPU, the `perf` [54] tool reports 235.7 million x86 instructions. To align with BitVMX's RISC-V CPU model, we apply a +10% overhead[22] [30] and then double the value as a conservative margin for OT verification and other auxiliary checks, yielding a total estimated trace length of $N \approx 5.19 \times 10^8$ steps.

BitVM1 [7]. We estimate the cost for the original BitVM protocol by scaling its public baseline for a 2^{32}-step trace. This baseline specifies a constant optimistic path of 1,944 vB and a worst-case dispute of 244,040 vB. As the dispute cost scales logarithmically with the trace length, we adjust it for our estimated trace of N steps (where $\lceil \log_2 N \rceil = 29$). This yields a total dispute cost of $(244{,}040 - 1{,}944) \cdot (29/32) + 1{,}944 \approx 221{,}344$ vB. Following the fee model in §8.3 ($f = 3$ sat/vB), this corresponds to approximately 664,032 sat (≈ $744).

BitVMX [39,65]. For BitVMX, we adapt Fairgate's baseline, which reports 34 rounds and ∼ 160 KiB of on-chain data for a 2^{32}-step trace. Scaling these figures logarithmically to our trace length N results in a reduced round count of $R = \lceil 34 \cdot (\log_4 N/32) \rceil = 31$ and a proportional data size of $160 \cdot (31/34) \approx 146$ KiB. This is equivalent to approximately 149,000 vB, costing 447,000 sat (≈ $502) at the same fee rate.

9 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 [28]. Aumann and Lindell [6] provided the first formalizations, later extended to the dishonest-majority setting by Goyal et al. [48]. 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 [5, 51, 57].

Zhu et al. [87] 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 [68], and generalized by Faust et al. [40], who formalized *financially backed covert security (FBC)*. *Our work* builds on this paradigm by introducing a PVC construction that is compatible with Bitcoin and enforces the computational output directly on-chain.

[22] This covers difference of RISC-V and x86 instruction sets.

Private smart contracts on expressive blockchains. Privacy on expressive blockchains has been pursued through several cryptographic paradigms. Early zkSNARK-based systems like Hawk [60] required a trusted manager i.e., a centralization point that persists in later function-hiding protocols like ZEXE and VeriZEXE [24, 81], while other tools have suffered from information leakage or anonymity breaks [75, 76]. Efforts to decentralize this trust with MPC, as in zkHawk [12, 13], introduced high client availability requirements, whereas others like Eagle [18] shift trust to an external committee. Alternative approaches rely on homomorphic encryption, which can expose client identities [73, 74], or on trusted execution environments (TEEs), which introduce trusted hardware assumptions [53, 85]. While recent work targets even stronger guarantees like *doubly private* contracts [44], a common thread unites these solutions: 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 leveraged public ledgers to achieve *fairness* in specific applications like lotteries and games, often assuming an enhanced version of Bitcoin [2, 3, 61]. This approach was later generalized to arbitrary secure function evaluation and to stateful settings with *secure cash distribution with penalties* [22, 63]. The interaction between MPC and ledgers was formalized in the GUC framework [27, 56], inspiring a line of work on reducing on-chain costs for output fairness [21, 23, 36, 62, 64]. This research culminated in protocols like *Insured MPC*, which provides UC-secure functionalities with identifiable aborts [19]. However, these protocols are either theoretical frameworks or require stateful contracts not natively supported by Bitcoin. In contrast, we provide an efficient, constant-round 2PC protocol that operates entirely on native Bitcoin.

Concurrent work combining *privacy free* garbled circuits within the construction of BitVM (e.g., [31, 37]) is orthogonal to our work.

10 Conclusion

In this work, we introduced BitPriv, the first protocol to enforce the outcome of a secure two-party computation directly on Bitcoin. By combining garbled circuits and a cut-and-choose mechanism with on-chain punishment via BitVM, BitPriv provides fairness, privacy, and verifiability without requiring consensus changes or new trust assumptions. Our evaluation demonstrates that the protocol is practical, with low optimistic costs (under $3) that make it suitable for real-world deployment. The high cost of a dispute serves as a strong economic deterrent against malicious behavior, ensuring security for rational adversaries. BitPriv serves as the backbone of a new generation of DeFi applications on Bitcoin, with BitMarket as the first example.

Acknowledgments

The work was partially supported by CoBloX Labs, by the European Research Council (ERC) under the European Union's Horizon 2020 research (grant agreement 771527-BROWSEC), by the Austrian Science Fund (FWF) through the SFB SpyCode project F8510-N and F8512-N, by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development and the Christian Doppler Research Association through the Christian Doppler Laboratory Blockchain Technologies for the Internet of Things (CDL-BOT), and by the WWTF through the project 10.47379/ICT22045.

References

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

[2] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Fair two-party computations via bitcoin deposits. In Rainer Böhme, Michael Brenner, Tyler Moore, and Matthew Smith, editors, *Financial Cryptography and Data Security - FC 2014 Workshops, BITCOIN and WAHC 2014, Christ Church, Barbados, March 7, 2014, Revised Selected Papers*, volume 8438 of *Lecture Notes in Computer Science*, pages 105–121. Springer, 2014.

[3] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on bitcoin. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 443–458. IEEE Computer Society, 2014.

[4] Gilad Asharov and Claudio Orlandi. Calling out cheaters: Covert security with public verifiability. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 681–698. Springer, 2012.

[5] Gilad Asharov and Claudio Orlandi. Calling out cheaters: Covert security with public verifiability. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 681–698. Springer, 2012.

[6] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *J. Cryptol.*, 23(2):281–343, 2010.

[7] Lukas Aumayr, Zeta Avarikioti, Robin Linus, Matteo Maffei, Andrea Pelosi, Christos Stefo, and Alexei Zamyatin. BitVM: Quasi-turing complete computation on bitcoin. Cryptology ePrint Archive, Paper 2024/1995, 2024.

[8] Lukas Aumayr, Zeta Avarikioti, Matteo Maffei, Giulia Scaffino, and Dionysis Zindros. Blink: An optimal proof of proof-of-work. *IACR Cryptol. ePrint Arch.*, page 692, 2024.

[9] Lukas Aumayr, Oguzhan Ersoy, Andreas Erwig, Sebastian Faust, Kristina Hostáková, Matteo Maffei, Pedro Moreno-Sanchez, and Siavash Riahi. Generalized channels from limited blockchain scripts and adaptor signatures. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part II*, volume 13091 of *Lecture Notes in Computer Science*, pages 635–664. Springer, 2021.

[10] Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 324–356. Springer, 2017.

[11] Ekrem Bal, Lukas Aumayr, Atacan İyidoğan, Giulia Scaffino, Hakan Karakuş, Cengiz Eray Aslan, and Orfeas Stefanos Thyfronitis Litos. Clementine: A collateral-efficient, trust-minimized, and scalable bitcoin bridge. Cryptology ePrint Archive, Paper 2025/776, 2025.

[12] Aritra Banerjee, Michael Clear, and Hitesh Tewari. zkhawk: Practical private smart contracts from mpc-based hawk. In *3rd Conference on Blockchain Research & Applications for Innovative Networks and Services, BRAINS 2021, Paris, France, September 27-30, 2021*, pages 245–248. IEEE, 2021.

[13] Aritra Banerjee and Hitesh Tewari. Multiverse of hawkness: A universally-composable mpc-based hawk variant. *Cryptogr.*, 6(3):39, 2022.

[14] Massimo Bartoletti, Stefano Lande, and Roberto Zunino. Bitcoin covenants unchained. *CoRR*, abs/2006.03918, 2020.

[15] Massimo Bartoletti, Riccardo Marchesin, and Roberto Zunino. Secure compilation of rich smart contracts on poor UTXO blockchains. In *9th IEEE European Symposium on Security and Privacy, EuroS&P 2024, Vienna, Austria, July 8-12, 2024*, pages 235–267. IEEE, 2024.

[16] Massimo Bartoletti, Riccardo Marchesin, and Roberto Zunino. Secure compilation of rich smart contracts on poor UTXO blockchains. In *9th IEEE European Symposium on Security and Privacy, EuroS&P 2024, Vienna, Austria, July 8-12, 2024*, pages 235–267. IEEE, 2024.

[17] Massimo Bartoletti and Roberto Zunino. Bitml: A calculus for bitcoin smart contracts. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 83–100. ACM, 2018.

[18] Carsten Baum, James Hsin-yu Chiang, Bernardo David, and Tore Kasper Frederiksen. Eagle: Efficient privacy preserving smart contracts. In Foteini Baldimtsi and Christian Cachin, editors, *Financial Cryptography and Data Security - 27th International Conference, FC 2023, Bol, Brač, Croatia, May 1-5, 2023, Revised Selected Papers, Part I*, volume 13950 of *Lecture Notes in Computer Science*, pages 270–288. Springer, 2023.

[19] Carsten Baum, Bernardo David, and Rafael Dowsley. Insured MPC: efficient secure computation with financial penalties. In Joseph Bonneau and Nadia Heninger, editors, *Financial Cryptography and Data Security - 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers*, volume 12059 of *Lecture Notes in Computer Science*, pages 404–420. Springer, 2020.

[20] Donald Beaver. Correlated pseudorandomness and the complexity of private computations. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 479–488, New York, NY, USA, 1996. Association for Computing Machinery.

[21] Fabrice Benhamouda, Shai Halevi, and Tzipora Halevi. Supporting private data on hyperledger fabric with secure multiparty computation. *IBM J. Res. Dev.*, 63(2/3):3:1–3:8, 2019.

[22] Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 421–439. Springer, 2014.

[23] Iddo Bentov, Ranjit Kumaresan, and Andrew Miller. Instantaneous decentralized poker. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II*, volume 10625 of *Lecture Notes in Computer Science*, pages 410–440. Springer, 2017.

[24] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. ZEXE: enabling decentralized private computation. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 947–964. IEEE, 2020.

[25] Johannes Buchmann, Erik Dahmen, Sarah Ereth, Andreas Hülsing, and Markus Rückert. On the security of the winternitz one-time signature scheme. *Int. J. Appl. Cryptogr.*, 3(1):84–96, 2013.

[26] Benedikt Bünz, Lucianna Kiffer, Loi Luu, and Mahdi Zamani. Flyclient: Super-light clients for cryptocurrencies. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 928–946. IEEE, 2020.

[27] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*, volume 4392 of *Lecture Notes in Computer Science*, pages 61–85. Springer, 2007.

[28] Ran Canetti and Rafail Ostrovsky. Secure computation with honest-looking parties: What if nobody is truly honest? (extended abstract). In Jeffrey Scott Vitter, Lawrence L. Larmore, and Frank Thomson Leighton, editors, *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, May 1-4, 1999, Atlanta, Georgia, USA*, pages 255–264. ACM, 1999.

[29] Ran Canetti, Ben Riva, and Guy N. Rothblum. Refereed delegation of computation. *Inf. Comput.*, 226:16–36, 2013.

[30] Christopher Celio, Daniel Palmer Dabbelt, David A. Patterson, and Krste Asanovic. The renewed case for the reduced instruction set computer: Avoiding ISA bloat with macro-op fusion for RISC-V. *CoRR*, abs/1607.02318, 2016.

[31] Weikeng Chen. SoK: BitVM with succinct on-chain cost. Cryptology ePrint Archive, Paper 2025/1253, 2025.

[32] Kevin Choi, Aathira Manoj, and Joseph Bonneau. Sok: Distributed randomness beacons. In *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*, pages 75–92. IEEE, 2023.

[33] T. Chou and C. Orlandi. The simplest protocol for oblivious transfer. In K. Lauter and F. Rodríguez-Henríquez, editors, *Progress in Cryptology - LATINCRYPT 2015 : 4th International Conference on Cryptology and Information Security in Latin America, Guadalajara, Mexico, August 23-26, 2015*, Lecture Notes in Computer Science, pages 40–58, Germany, 2015. Springer. 4th International Conference on Cryptology and Information Security in Latin America (LATINCRYPT 2015), August 23-26, 2015, Guadalajara, Mexico, LATINCRYPT 2015 ; Conference date: 23-08-2015 Through 26-08-2015.

[34] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 910–927. IEEE, 2020.

[35] Poulami Das, Lisa Eckey, Tommaso Frassetto, David Gens, Kristina Hostáková, Patrick Jauernig, Sebastian Faust, and Ahmad-Reza Sadeghi. FastKitten: Practical smart contracts on bitcoin. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 801–818, Santa Clara, CA, August 2019. USENIX Association.

[36] Bernardo David, Rafael Dowsley, and Mario Larangeira. Kaleidoscope: An efficient poker protocol with payment distribution and penalty enforcement. In Sarah Meiklejohn and Kazue Sako, editors, *Financial Cryptography and Data Security - 22nd International Conference, FC 2018, Nieuwpoort, Curaçao, February 26 - March 2, 2018, Revised Selected Papers*, volume 10957 of *Lecture Notes in Computer Science*, pages 500–519. Springer, 2018.

[37] Liam Eagen. Glock: Garbled locks for bitcoin. Cryptology ePrint Archive, Paper 2025/1485, 2025.

[38] Bernard B. Elyakime, Jean-Jacques Laffont, Patrice Loisel, and Q.H. Vuong. First-price sealed-bid auctions with secret reservation prices. *Annales d'Economie et de Statistique*, (34):115–141, 1994. Egalement paru dans : INRA-ESR Toulouse Série D ; 93-03D.

[39] Fairgate. BitVMX. https://www.fairgate.io/bitvmx. Accessed: 2025-08-27.

[40] Sebastian Faust, Carmit Hazay, David Kretzler, and Benjamin Schlosser. Financially backed covert security. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *Public-Key Cryptography - PKC 2022 - 25th IACR International Conference on Practice and Theory of Public-Key Cryptography, Virtual Event, March 8-11, 2022, Proceedings, Part II*, volume 13178 of *Lecture Notes in Computer Science*, pages 99–129. Springer, 2022.

[41] Tommaso Frassetto, Patrick Jauernig, David Koisser, David Kretzler, Benjamin Schlosser, Sebastian Faust, and Ahmad-Reza Sadeghi. POSE: practical off-chain smart contract execution. In *30th Annual Network and Distributed System Security Symposium, NDSS 2023, San Diego, California, USA, February 27 - March 3, 2023*. The Internet Society, 2023.

[42] Chaya Ganesh, Bhavana Kanukurthi, and Girisha Shankar. Secure auctions in the presence of rational adversaries. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 1173–1186, New York, NY, USA, 2022. Association for Computing Machinery.

[43] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. *J. ACM*, 71(4), August 2024.

[44] Sanjam Garg, Aarushi Goel, Dimitris Kolonelos, and Rohit Sinha. Jigsaw: Doubly private smart contracts. Cryptology ePrint Archive, Paper 2025/1147, 2025.

[45] Peter Gazi, Aggelos Kiayias, and Dionysis Zindros. Proof-of-stake sidechains. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 139–156. IEEE, 2019.

[46] Christopher Goes. The interblockchain communication protocol: An overview, 2020.

[47] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229. ACM, 1987.

[48] Vipul Goyal, Payman Mohassel, and Adam D. Smith. Efficient two party and multi party computation against covert adversaries. In Nigel P. Smart, editor, *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, volume 4965 of *Lecture Notes in Computer Science*, pages 289–306. Springer, 2008.

[49] Mike Graf, Daniel Rausch, Viktoria Ronge, Christoph Egger, Ralf Küsters, and Dominique Schröder. A security framework for distributed ledgers. In Yongdae Kim,

Jong Kim, Giovanni Vigna, and Elaine Shi, editors, *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, pages 1043–1064. ACM, 2021.

[50] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326. Springer, 2016.

[51] Cheng Hong, Jonathan Katz, Vladimir Kolesnikov, Wenjie Lu, and Xiao Wang. Covert security with public verifiability: Faster, leaner, and simpler. *IACR Cryptol. ePrint Arch.*, page 1108, 2018.

[52] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 145–161. Springer, 2003.

[53] Harry A. Kalodner, Steven Goldfeder, Xiaoqi Chen, S. Matthew Weinberg, and Edward W. Felten. Arbitrum: Scalable, private smart contracts. In William Enck and Adrienne Porter Felt, editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 1353–1370. USENIX Association, 2018.

[54] Kernel.org. perf Tutorial. https://perfwiki.github.io/main/tutorial/. Accessed: 2025-08-27.

[55] Aggelos Kiayias, Andrew Miller, and Dionysis Zindros. Non-interactive proofs of proof-of-work. In Joseph Bonneau and Nadia Heninger, editors, *Financial Cryptography and Data Security - 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers*, volume 12059 of *Lecture Notes in Computer Science*, pages 505–522. Springer, 2020.

[56] Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. Fair and robust multi-party computation using a global transaction ledger. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 705–734. Springer, 2016.

[57] Vladimir Kolesnikov and Alex J. Malozemoff. Public verifiability in the covert model (almost) for free. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 210–235. Springer, 2015.

[58] Vladimir Kolesnikov, Ahmad-Reza Sadeghi, and Thomas Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. *IACR Cryptol. ePrint Arch.*, page 411, 2009.

[59] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations*, volume 5126 of *Lecture Notes in Computer Science*, pages 486–498. Springer, 2008.

[60] Ahmed E. Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*, pages 839–858. IEEE Computer Society, 2016.

[61] Ranjit Kumaresan and Iddo Bentov. How to use bitcoin to incentivize correct computations. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 30–41. ACM, 2014.

[62] Ranjit Kumaresan and Iddo Bentov. Amortizing secure computation with penalties. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 418–429. ACM, 2016.

[63] Ranjit Kumaresan, Tal Moran, and Iddo Bentov. How to use bitcoin to play decentralized poker. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 195–206, New York, NY, USA, 2015. Association for Computing Machinery.

[64] Ranjit Kumaresan, Vinod Vaikuntanathan, and Prashant Nalini Vasudevan. Improvements to secure computation with penalties. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 406–417. ACM, 2016.

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

[66] Lightning Network Developers. lnd: Lightning network daemon. https://github.com/lightningnetwork/lnd, 2025. Version v0.19.3-beta (released 2025-08-21); commit 2fb725e; MIT License. Accessed 2025-08-24.

[67] Robin Linus, Lukas Aumayr, Zeta Avarikioti, Matteo Maffei, Andrea Pelosi, Orfeas Thyfronitis Litos, Christos Stefo, David Tse, and Alexei Zamyatin. Bridging bitcoin to second layers via BitVM2. Cryptology ePrint Archive, Paper 2025/1158, 2025.

[68] Yi Liu, Junzuo Lai, Qi Wang, Xianrui Qin, Anjia Yang, and Jian Weng. Robust publicly verifiable covert security: Limited information leakage and guaranteed correctness with low overhead. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology - ASIACRYPT 2023 - 29th International Conference on the Theory and Application of Cryptology and Information Security, Guangzhou, China, December 4-8, 2023, Proceedings, Part I*, volume 14438 of *Lecture Notes in Computer Science*, pages 272–301. Springer, 2023.

[69] Varun Madathil, Sri Aravinda Krishnan Thyagarajan, Dimitrios Vasilopoulos, Lloyd Fournier, Giulio Malavolta, and Pedro Moreno-Sanchez. Cryptographic oracle-based conditional payments. In *30th Annual Network and Distributed System Security Symposium, NDSS 2023, San Diego, California, USA, February 27 - March 3, 2023*. The Internet Society, 2023.

[70] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, EC '99, page 129–139, New York, NY, USA, 1999. Association for Computing Machinery.

[71] Optimism Foundation. Rollup protocol overview. https://docs.optimism.io/stack/rollup/overview, 2025. Optimism Docs. Accessed: 2025-08-22.

[72] Casey Rodarmor and contributors. *Ordinal Theory Handbook*, 2023. Accessed: 2025-08-22.

[73] Ravital Solomon, Rick Weber, and Ghada Almashaqbeh. smartfhe: Privacy-preserving smart contracts from fully homomorphic encryption. In *8th IEEE European Symposium on Security and Privacy, EuroS&P 2023, Delft, Netherlands, July 3-7, 2023*, pages 309–331. IEEE, 2023.

[74] Samuel Steffen, Benjamin Bichsel, Roger Baumgartner, and Martin T. Vechev. Zeestar: Private smart contracts by homomorphic encryption and zero-knowledge proofs. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*, pages 179–197. IEEE, 2022.

[75] Samuel Steffen, Benjamin Bichsel, Mario Gersbach, Noa Melchior, Petar Tsankov, and Martin T. Vechev. zkay: Specifying and enforcing data privacy in smart contracts. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 1759–1776. ACM, 2019.

[76] Samuel Steffen, Benjamin Bichsel, and Martin T. Vechev. Zapper: Smart contracts with data and identity privacy. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 2735–2749. ACM, 2022.

[77] Ertem Nusret Tas, István András Seres, Yinuo Zhang, Márk Melczer, Mahimna Kelkar, Joseph Bonneau, and Valeria Nikolaenko. Atomic and fair data exchange via blockchain. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, pages 3227–3241. ACM, 2024.

[78] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit, 2016.

[79] Robert J. Weber. Auction theory: By vijay krishna. academic press, 2002. *Games Econ. Behav.*, 45(2):488–497, 2003.

[80] Pieter Wuille, Jonas Nick, and Anthony Towns. Bip 341: Taproot: Segwit version 1 spending rules. Bitcoin Improvement Proposal, 2020. Section "Constructing and spending Taproot outputs". Status: Final. License: BSD-3-Clause.

[81] Alex Luoyuan Xiong, Binyi Chen, Zhenfei Zhang, Benedikt Bünz, Ben Fisch, Fernando Krell, and Philippe

Camacho. Verizexe: Decentralized private computation with universal setup. In Joseph A. Calandrino and Carmela Troncoso, editors, *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, pages 4445–4462. USENIX Association, 2023.

[82] Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast extension for correlated OT with small communication. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 1607–1626. ACM, 2020.

[83] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 160–164. IEEE Computer Society, 1982.

[84] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 220–250. Springer, 2015.

[85] Fan Zhang, Warren He, Raymond Cheng, Jernej Kos, Nicholas Hynes, Noah M. Johnson, Ari Juels, Andrew Miller, and Dawn Song. The ekiden platform for confidentiality-preserving, trustworthy, and performant smart contracts. *IEEE Secur. Priv.*, 18(3):17–27, 2020.

[86] Kaiyi Zhang, Hongrui Cui, and Yu Yu. Revisiting the constant-sum winternitz one-time signature with applications to sphincs$^+$ and XMSS. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part V*, volume 14085 of *Lecture Notes in Computer Science*, pages 455–483. Springer, 2023.

[87] Ruiyu Zhu, Changchang Ding, and Yan Huang. Efficient publicly verifiable 2pc over a blockchain with applications to financially-secure computations. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 633–650. ACM, 2019.

[88] Ruiyu Zhu, Yan Huang, Jonathan Katz, and Abhi Shelat. The Cut-and-Choose game and its application to cryptographic protocols. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 1085–1100, Austin, TX, August 2016. USENIX Association.

A 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 2.

A.1 Transaction Ledger Model

Transaction Structure. We adopt the notation of [9]. 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.

Ledger Ideal Functionality $\mathcal{L}$. We slightly adapt the ledger functionality $\mathcal{L}$ from [9], 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 [10, 49], 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 that are in the set $\mathcal{P}$ and maintains a PKI *for those*

parties. The functionality maintains the set of all accepted transactions TX *and all unspent transaction outputs* UTXO.

Initialize public keys: Upon $(\texttt{REGISTER}, pk_P) \xleftarrow{\tau_0}$ 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., $\texttt{tx.TimeLock} \leq \texttt{now}$.

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 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 $(\mathsf{state}, \mathsf{PKI}, \mathsf{TX}) \xhookrightarrow{\tau_0} X$.

A.2 Ideal Functionality $\mathcal{F}_{\mathbf{BitVM}}$

In our analysis, we abstract BitVM and its variants into a single ideal functionality, denoted by $\mathcal{F}_{\mathrm{BitVM}}$, parameterized by a dispute bound T_{BitVM} representing the maximum number of steps or rounds allowed during dispute resolution. Conceptually, $\mathcal{F}_{\mathrm{BitVM}}$ captures an interactive 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.

We note that financial enforcement in $\mathcal{F}_{\mathrm{BitVM}}$ is the mechanism that underlies our protocol's financial fairness guarantee (see Section 2).

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

Let P_A assume the role of the prover and P_B the verifier and *v be the verification function i.e., $v(w, o) = 1$ iff $p.f(w) = o$*

Setup Phase

Agree on Parameters: Upon $(\texttt{SETUP}, p) \hookleftarrow P_A$ and $(\texttt{SETUP}, p') \hookleftarrow P_A$, 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 $tx.Out = \big((c_A, \text{One-Sig}_{Pk_A}), (c_B, \text{One-Sig}_{Pk_B})\big)$ or $tx.Out = \big((0, \text{One-Sig}_{Pk_A}), (\mathsf{setup.all}, \text{One-Sig}_{Pk_B})\big)$; if none exists, output error and halt.
     - if $v(w, o) \neq 1$: verify if $tx \in \texttt{TX}$ with $tx.Out = \big((0, \text{One-Sig}_{Pk_A}), (\mathsf{setup.all}, \text{One-Sig}_{Pk_B})\big)$; if none exists, output error and halt
2. Idle A (no $(\texttt{CLAIM}, \mathtt{w}, \mathtt{o})$ 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 $tx \in$ TX with $tx.Out = ((0, \text{One-Sig}_{Pk_A}), (\text{setup.all}, \text{One-Sig}_{Pk_B}))$; if none exists, output error and halt.

read-ledger: (READ) $\overset{\tau}{\hookrightarrow} \mathcal{L}$; return (state, PKI, TX) $\overset{\tau}{\hookleftarrow} \mathcal{L}$.

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.

A.3 Ideal Functionality $\mathcal{F}_{\text{OT}}$

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.

A.4 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 [4, 6], 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.

A.5 Security Defintions

We formalize the security of BitPriv via an ideal functionality $\mathcal{F}_{\text{BitPriv}}$, 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 and financial penalty. This notion serves as a stepping stone toward proving that BitPriv achieves the security goals outlined in Section 2.

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 [6], 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 $\mathcal{F}_{\text{BitPriv}}^{\mathcal{L}(u,\Sigma,\mathcal{V})}$

Setup Phase

Register to Ledger: Upon (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 (SETUP, p, att) $\overset{\tau}{\hookleftarrow} P_A$ and (SETUP, $p', [y]$) $\overset{\tau}{\hookleftarrow} P_B$, or (ABORT) $\overset{\tau}{\hookleftarrow}$ from either:

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

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

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

1. If (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 (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 $tx \in$ 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 (OPT-CLOSE, j', k') in the optimistic path with $j' \neq j$, wait for u call read-ledger and verify if $tx \in$ TX with $tx.Out = (\text{all}, \text{One-Sig}_{Pk_A})$; if none exists, output error and halt.
2. If (REQUEST, m) $\overset{\tau_2 \le \tau_1 + \Delta}{\longleftarrow}$ from P_B:
   - If $detectedA$: (REVEAL-DETECTED, m) $\overset{\tau_2}{\hookrightarrow} P_A$
   - If $undetectedA$: (REVEAL-UNDETECTED, m) $\overset{\tau_2}{\hookrightarrow} P_A$
   - Else: (REVEAL, m) $\overset{\tau_2}{\hookrightarrow} P_A$
3. Idle B (no (REQUEST, j) within Δ for some $j \in [n]$):
   - $corruptedB := 1$
   - (ABORTED-B) $\overset{\tau_2' = \tau_1 + \Delta}{\longrightarrow} P_A$
4. Upon (PUNISH-B) $\overset{\tau_3 \le \tau_2' + u}{\longleftarrow} P_A$:
   - If (REQUEST) $\overset{\tau_3' \le \tau_3}{\longleftarrow} P_B$: handle as above.
   - Else:
     - (PUNISH-ABORT) $\overset{\tau_3}{\hookrightarrow} (P_A, P_B)$
     - read-ledger and verify if $tx \in$ TX with $tx.Out = \big((\text{all}, \text{One-Sig}_{Pk_A}), (0, \text{One-Sig}_{Pk_B})\big)$; if none exists, (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 ((REVEAL-A), z') $\overset{\tau_4 \le \tau' + \Delta}{\longleftarrow} P_A$ and ($detectedA \vee undetectedA$):
   - If $detectedA$: ((DETECTED-A), witness) $\overset{\tau_4}{\hookrightarrow} P_B$
   - If $undetectedA$: ((OUTPUT), z') $\overset{\tau_4}{\hookrightarrow} P_B$
2. Else if ((REVEAL-A), f, x) $\overset{\tau_4 \le \tau' + \Delta}{\longleftarrow}$ and $\neg(detectedA \vee undetectedA)$:

- $z = f(x, y)$
- $(\texttt{(OUTPUT)}, z) \overset{\tau_4}{\hookrightarrow} P_B$

3. Idle A (no `(REVEAL)` within Δ):
   - `(ABORTED-A)` $\overset{\tau_5 = \tau_4 + \Delta}{\longleftrightarrow} P_B$
4. Upon `(PUNISH-A)` $\overset{\tau_6 \leq \tau_5 + u}{\longleftrightarrow}$ from P_B:
   - If `(REVEAL-A)` $\overset{\tau'_6 \leq \tau_6}{\longleftrightarrow}$: handle as above.
   - Else:
     - `(PUNISH-ABORT)` $\overset{\tau_6}{\hookrightarrow} (P_A, P_B)$
     - read-ledger and verify if $tx \in \texttt{TX}$ with $tx.Out = \big((0, \text{One-Sig}_{Pk_A}), (\text{all}, \text{One-Sig}_{Pk_B})\big)$; if none exists, `(ERROR)` $\overset{\tau_6}{\hookrightarrow} (P_A, P_B)$ 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)` $\overset{\tau_7 \leq \tau'' + \Delta}{\longleftrightarrow} P_B$: for every round until $\mathcal{T}_{\text{BitVM}}$ call read-ledger:
   - If *detectedA*: verify if $tx \in \texttt{TX}$ with $tx.Out = \big((0, \bot), (\text{all}, \text{One-Sig}_{Pk_B})\big)$; if none exists, `(ERROR)` $\overset{\tau_6}{\hookrightarrow} (P_A, P_B)$ and halt. Otherwise: `(FINISH)` $\overset{\tau_6}{\hookrightarrow} (P_A, P_B)$ and halt.
   - If $\neg$*detectedA*: verify if $tx \in \texttt{TX}$ with $tx.Out = \big((\text{all}, \text{One-Sig}_{Pk_A}), (0, \bot)\big)$; if none exists, `(ERROR)` $\overset{\tau_6}{\hookrightarrow} (P_A, P_B)$ and halt. Otherwise: `(FINISH)` $\overset{\tau_6}{\hookrightarrow} (P_A, P_B)$ and halt.
2. If `(CLOSE,`z'`)` $\overset{\tau_7 \leq \tau'' + \Delta}{\longleftrightarrow} P_B$:
   - If *undetectedA*: `(OUTPUT-UNDETECTED,` z'`)` $\overset{\tau_7}{\hookrightarrow} P_A$. Let $(c_A, c_B) \leftarrow g(z')$, wait for Δ rounds 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. Otherwise send `(FINISH)` to both parties and halt.
   - If P_A honest: $z' = f(x, y)$, send `(OUTPUT,` $f(x, y)$`)` $\overset{\tau_7}{\hookrightarrow} P_A$. Let $(c_A, c_B) \leftarrow g(f(x, y))$, wait for Δ rounds to call read-ledger and verify if $tx \in \texttt{TX}$ 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. Otherwise send `(FINISH)` to both parties and halt.
3. If `(CLOSE-EQUIV,` z''`)` $\overset{\tau_7 \leq \tau'' + \Delta}{\longleftrightarrow} P_B$: send `(EQUIV-ATTEMPT)` $\overset{\tau_7}{\hookrightarrow} P_A$
   - If `(PUNISH-EQUIV)` $\overset{\tau_{10} \leq \tau_7 + \Delta}{\longleftrightarrow}$, 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. Otherwise, send `(EQUIV-DETECTED)` to P_B
   - Otherwise `(IDLE-EQUIV)` $\overset{\tau_{11} = \tau_7 + \Delta}{\longleftrightarrow}$

   Upon `(CLAIM-EQUIV)` $\overset{\tau_{12} \leq \tau_{11} + u}{\longleftrightarrow}$:
   - send `(EQUIV-UNDETECTED,` z''`)` $\overset{\tau_{12}}{\hookrightarrow} P_A$. Let $(c_A, c_B) \leftarrow g(z'')$, 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.
4. Idle B (no `(PUNISH-CHEAT)` or `(CLOSE,`z'`)` within Δ):
   - `(ABORTED-B)` $\overset{\tau_8 = \tau'' + \Delta}{\longleftrightarrow} P_A$.

   Upon `(PUNISH-B)` $\overset{\tau_9 \leq \tau_8 + u}{\longleftrightarrow} P_A$:
   - If `(PUNISH-CHEAT)` $\vee$ `(CLOSE)` $\overset{\tau'_9 \leq \tau_9}{\longleftrightarrow} P_B$: handle as above.
   - Else:
     - `(PUNISH-ABORT)` $\overset{\tau_9}{\hookrightarrow} (P_A, P_B)$
     - 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, `(ERROR)` $\overset{\tau_9}{\hookrightarrow} (P_A, P_B)$ and halt.

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

B Garbled Circuits and Oblivious Transfer

B.1 Garbled Circuits

A Garbled Circuit (GC) [47, 83] 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 [87] as a tuple of efficient, deterministic algorithms $(\mathsf{Gb}, \mathsf{Ev}, \mathsf{En}, \mathsf{De}, \mathsf{Ve})$, 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.

B.2 Oblivious Transfer

OT [20, 33] 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* [52] 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* [88] approach: the garbler generates multiple circuits, a random subset of which are opened for verification to ensure correctness.

C BitPriv in the $\mathcal{F}_{\text{BitVM}}, \mathcal{L}$-hybrid world

In this section, we specify BitPriv as a protocol in the $\mathcal{F}_{\text{BitVM}}, \mathcal{L}$-hybrid world. 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 $(F_i, e_i, d_i) \leftarrow \mathsf{Gb}(1^\kappa, f, \text{seed}_i)$ 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 $j \in [\text{outbits}]$ (corresponding to one plain-text output bit), G constructs two vectors: $\mathsf{Output}^0_j = [\mathsf{Com}(L^0_{\text{out},i,j})]_{i\in[n]}$ and $\mathsf{Output}^1_j = [\mathsf{Com}(L^1_{\text{out},i,j})]_{i\in[n]}$, 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. 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 $\text{Com}_{\text{seed}_{E,i}}$ 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 5.3, 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 5*

(a) **(Post Setup)** E or G sends $(\texttt{POST}, tx) \overset{\tau_0}{\hookrightarrow} \mathcal{L}$ with $tx.id = \text{Setup}.id$. 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 $(\texttt{POST}, \text{CommitEvaluationIndex}) \overset{\tau_2}{\hookrightarrow} \mathcal{L}$

ii. If Setup remains unspent on $\mathcal{L}$ for Δ rounds, G sends $(\texttt{POST}, tx_G) \overset{\tau_2'}{\hookrightarrow} \mathcal{L}$ with $tx_G.Out = (\text{all}, \text{One-Sig}_{Pk_G})$

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 $(\texttt{POST}, \text{RevealSeeds}) \overset{\tau'}{\hookrightarrow}$, 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 $\text{Com}_{X_{m,\lambda}}$.

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 $tx_G.Out = (\mathsf{all}, \mathsf{One\text{-}Sig}_{Pk_G})$

iii. If CommitEvaluationIndex remains unspent on $\mathcal{L}$ for Δ rounds, E sends $(\texttt{POST}, tx_E) \overset{\tau_5}{\hookrightarrow} \mathcal{L}$ with $tx_E.Out = (\mathsf{all}, \mathsf{One\text{-}Sig}_{Pk_E})$

Wait (up to maximum $\tau_5 + u$) for a transaction spending CommitEvaluationIndex to appear on $\mathcal{L}$: If $(tx_E \vee tx_G) \in \mathcal{L}.\mathsf{TX}$ halt, otherwise let τ'' be the time that RevealSeeds appears on $\mathcal{L}$

(d) **(upon** RevealSeeds $\in \mathcal{L}.\mathsf{TX})$
Distinguish between the following cases:

i. E sends valid $(\texttt{POST}, \mathsf{Close}_{out,i}) \overset{\tau_7}{\hookrightarrow} \mathcal{L}$ 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 $\mathsf{Close}_{out,i})$ is equivalent to $\big((c_A, \mathsf{One\text{-}Sig}_{Pk_A}), (c_B, \mathsf{One\text{-}Sig}_{Pk_B})\big)$

ii. E sends valid $(\texttt{POST}, \mathsf{Close}_{out',i}) \overset{\tau_7}{\hookrightarrow} \mathcal{L}$ 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 $(\texttt{POST}, \mathsf{setup}) \overset{\tau_7}{\hookrightarrow} \mathcal{L}$ 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 $tx_G.Out = (\mathsf{all}, \mathsf{One\text{-}Sig}_{Pk_G})$

D 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 leverage PVC to demonstrate that BitPriv satisfies the protocol goals defined in Section 2.

Theorem 4. *Assume a straight-line extractable commitment scheme in the global random oracle model (ROM), the garbling scheme is secure, Π_{OT} UC-realises $\mathcal{F}_{OT}$, and the signature scheme is existentially unforgeable under a chosen-message attack (EUF-CMA). Then* BitPriv *securely realises $\mathcal{F}_{\mathsf{BitPriv}}$ in the $\mathcal{F}_{Bit_{VM}}, \mathcal{L}$-hybrid world.*

Proof. **Covert Security - Malicious G**

Let $\mathcal{A}$ be an adversary that corrupts G. We construct the following simulator S[23] 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 $\{\mathsf{com}_{seed_{E_i}}\}_{i=1}^n$ 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 $\mathcal{A}$:
   - 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 $\mathsf{Output}_j^0 = [\mathbf{com}(L_{\mathsf{out},i,j}^0)]_{i\in[n]}$ and $\mathsf{Output}_j^1 = [\mathbf{com}(L_{\mathsf{out},i,j}^1)]_{i\in[n]}$:
   - 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 $X_{i,j} := (X_{i,j}^b, X_{i,j}^{1-b})$, a randomly permuted pair of the j-th input label of G.

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

5. S runs n Π_{OT} instances with $\mathcal{A}$, where S plays the receiver using 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 $v(\text{witness}, 1)$ 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 $\textit{undetectedA} \wedge \neg \textit{uneval}$ or $|J| = 0$, S sends (POST, $\text{Close}_{z',m}$) to $\mathcal{L}$ and when the transaction appears on informs $\mathcal{F}$ by sending (CLOSE,z'). S outputs whatever $\mathcal{F}$ outputs.

   (c) If $\textit{undetectedA} \wedge \textit{uneval}$, S wait for Δ rounds, receives (ABORTED-B) from $\mathcal{F}$m forwards it to $\mathcal{A}$ and halts. TODO: add transaction Close-collateral

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 $\{\mathsf{com}_{seed_{E_i}}\}_{i=1}^n$ 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 $\mathsf{Output}^0_j = [\mathbf{com}(L^0_{\mathsf{out},i,j})]_{i\in[n]}$ and $\mathsf{Output}^1_j = [\mathbf{com}(L^1_{\mathsf{out},i,j})]_{i\in[n]}$:

   - 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 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 $\mathsf{Output}_j^0 = [\mathbf{com}(L^0_{\text{out},i,j})]_{i\in[n]}$ and $\mathsf{Output}_j^1 = [\mathbf{com}(L^1_{\text{out},i,j})]_{i\in[n]}$ 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 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 $com_{OT_{i,\lambda}} := [com_{\text{msg}_{i,\lambda,j}}]\ \forall i \in [n],\ \forall \lambda \in [l],\ \forall j \in |transcript_{i,\lambda}|$. Verify that the commitments received from $\mathcal{A}$ match the transcript and otherwise E aborts and outputs $\perp$. 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: $F'_{\rho'} \neq F_\rho \vee \text{Output}^0_\rho \neq \text{Output}'^0_\rho \vee \text{Output}^1_\rho \neq \text{Output}'^1_\rho \vee \exists \lambda : com'_{OT_{\rho,\lambda}} \neq com_{OT_{\rho,\lambda}} \vee \exists j : com'_{X_{\rho,j}} \neq com_{X_{\rho,j}}$.
   (a) if $|J| = 1 \wedge m \notin J$ 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_{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

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 $tx \in \mathcal{L}.\mathsf{TX}$, 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 $\mathcal{F}_{\text{BitVM}}$. If received (INIT) from $\mathcal{F}_{\text{BitVM}}$, monitor $\mathcal{L}$ for T_{BitVM} rounds and if $tx \in \mathsf{TX}$ with $tx.Out = \big((0, \bot), (\mathsf{all}, \mathsf{One\text{-}Sig}_{Pk_B})\big)$ send (FINISH) to $\mathcal{A}$.
   (b) If $undetectedA \wedge \neg uneval$ or $|J| = 0$, send (POST, $\text{Close}_{z',m}$) 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 $\mathcal{F}_{\text{BitVM}}$ 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 $(\texttt{ABORT}) \overset{\tau+t_a}{\hookrightarrow} \mathcal{F}$

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 $\mathsf{Output}_j^0 = [\mathbf{com}(L_{\text{out},i,j}^0)]_{i\in[n]}$ and $\mathsf{Output}_j^1 = [\mathbf{com}(L_{\text{out},i,j}^1)]_{i\in[n]}$, where $L_{\text{out},i,j}^b$ 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 $(L_{G,i,j}^x, 0^\lambda)$ where $L_{G,i,j}^x$ 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 $L_{G,m}^x$ 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 RevealSeeds. Send tx to $\mathcal{L}$ spending the output through the expired timelock. Monitor $\mathcal{L}$ for the next u rounds, and if $tx \in \mathcal{L}$.TX, send (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 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 $\text{Output}_j^0(m)$ and $\text{Output}_j^1(m)\ \forall j \in [l]$ 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 $\text{Output}_j^0(m)$ and $\text{Output}_j^1(m)$, are $L_{\text{out},j,m}^{z[j]}$ which unlock commitments $\text{Output}_j^{z[j]}(m)$ 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 $\Phi_{\text{mis}}(w, o) = 1$ when cheating is detected, and (iii) defamation-freeness (no certificate can be forged against an honest party, except with negligible probability).

Corollary 4.1 (PVC Security of BitPriv)**.** *In the $(\mathcal{F}_{BitVM}, \mathcal{L}, \mathcal{F}_{OT})$-hybrid model, and under the assumptions in Section 2, 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 $\mathcal{F}_{\text{BitPriv}}$. In particular, when an honest evaluator E initiates the dispute mechanism through $\mathcal{F}_{\text{BitVM}}$, 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 4.2. BitPriv *achieves* privacy, financial fairness, *and* guaranteed output delivery *with probability at least* ϵ.

Proof. Implied by Theorem 4.1, the financial enforcement of $\mathcal{F}_{\text{BitVM}}$, and the security of $\mathcal{L}$.

E 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.

E.1 Garbler Compliance

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

Model parameters:

- $\alpha(\sigma)$: utility from learning $\sigma \in \{0, \ldots, \text{outbits}\}$ bits about G'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: G 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:

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

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:

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

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 $\mathcal{W}_2$:

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

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

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

E.2 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 6 (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:*

$$\gamma + r_E > \beta$$

F Additional Evaluation Results

In this section we provide further measurements that complement the evaluation in the main text. Table 4 reports garbling performance for SHA-256 compression circuits, showing the cost of garbling, hashing, and communication under LAN/WAN settings as n increases. Tables 5–7 detail the scaling behavior of the Close transaction for different witness designs (U, PC, PS) across k at $n = 4, 8, 16$. Tables 8 and 9 give the size and fee of the RevealSeeds transaction as a function of input length l_{inbits} and number of circuits n, respectively. Finally, Figure 4 summarizes the scaling trends of Close across tapscript and transaction size, with full numeric values provided in the tables above. All results are reproducible using the artifact accompanying this paper.

n	Garble+ser (ms)	Hash (ms)	LAN (s)	WAN (s)	MB
1	1214	336	0.003	0.069	0.689
4	3353	1023	0.012	0.156	2.756
8	6329	1905	0.023	0.271	5.511
16	12823	3952	0.046	0.502	11.022

Table 4: SHA-256 (compression) garbling performance.

Table 5: Close scaling vs. k (vB and witness *out_pre*) for $n = 4$.

k	vB_U (vB)	vB_PC (vB)	vB_PS (vB)	W_U (B)	W_PC (B)	W_PS (B)
1	264	264	269	17	17	34
2	285	276	277	34	33	34
4	328	301	293	68	65	34
8	412	349	325	136	129	34
16	580	445	389	272	259	34
32	916	637	517	544	515	34
64	1580	1013	765	1088	1027	34

Table 6: Close scaling vs. k (vB and witness *out_pre*) for $n = 8$.

k	vB_U (vB)	vB_PC (vB)	vB_PS (vB)	W_U (B)	W_PC (B)	W_PS (B)
1	289	289	294	17	17	34
2	310	301	302	34	33	34
4	353	326	318	68	65	34
8	437	374	350	136	129	34
16	605	470	414	272	259	34
32	941	662	542	544	515	34
64	1605	1038	790	1088	1027	34

Table 7: Close scaling vs. k (vB and witness *out_pre*) for $n = 16$.

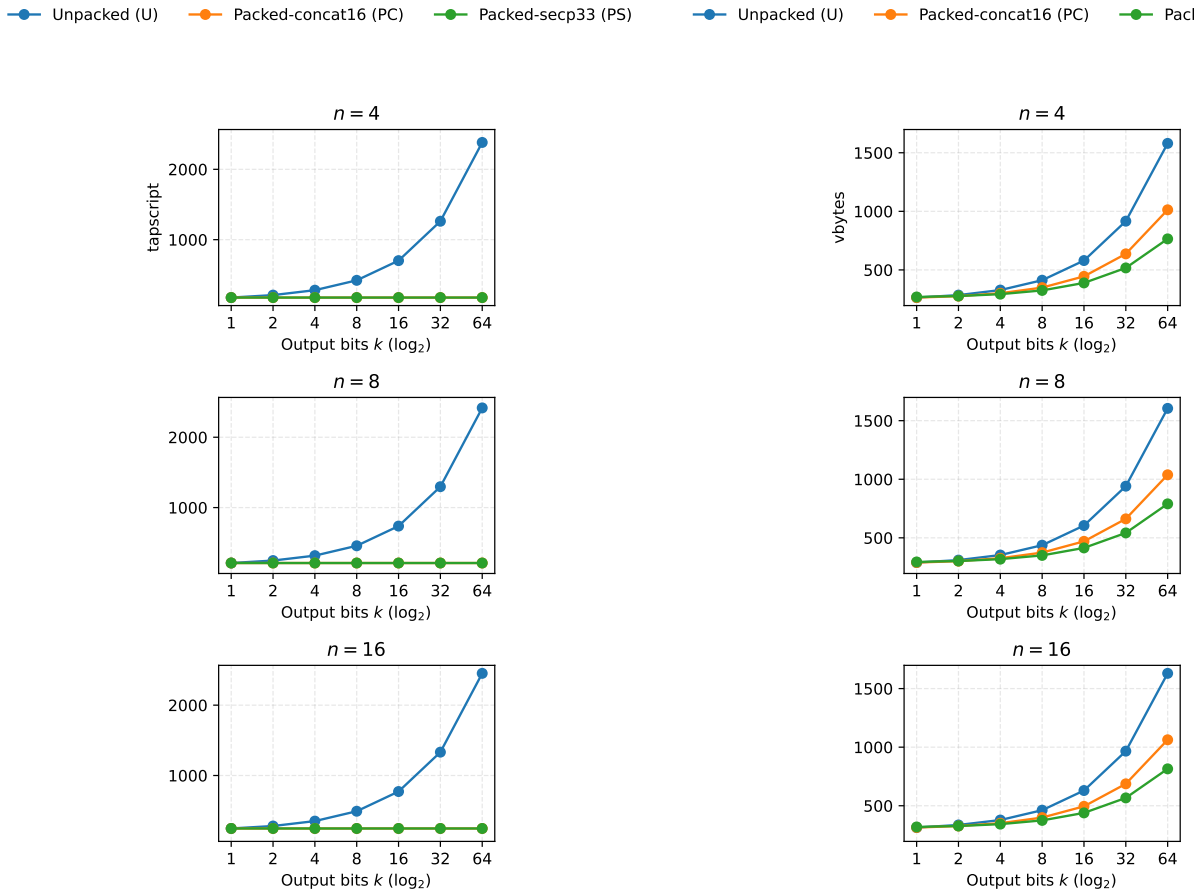
k	vB_U (vB)	vB_PC (vB)	vB_PS (vB)	W_U (B)	W_PC (B)	W_PS (B)
1	314	314	319	17	17	34
2	336	327	327	34	33	34
4	378	351	343	68	65	34
8	462	399	375	136	129	34
16	630	495	439	272	259	34
32	966	687	567	544	515	34
64	1630	1063	815	1088	1027	34

Table 8: RevealSeeds transaction size & cost vs. l_{inbits} for $n = 16$ circuits. Fee rate $f = 4$ sat/vB.

l_{inbits}	vbytes	fee (sat)	fee ($)
64	1,821	7,284	8.16
128	3,261	13,044	14.61
256	6,143	24,572	27.52
512	11,903	47,612	53.32
1024	23,423	93,692	104.93
2048	46,462	185,848	208.14

Table 9: RevealSeeds transaction size & cost vs. n for fixed $l_{\text{inbits}} = 128$. Fee rate $f = 4$ sat/vB.

n	vbytes	fee (sat)	fee ($)
4	3,089	12,356	13.84
8	3,149	12,596	14.11
16	3,261	13,044	14.61
32	3,477	13,908	15.58
64	3,901	15,604	17.48
128	4,742	18,968	21.24
256	6,414	25,656	28.73



(a) Tapscript size vs. k at n=16 for U/PC/PS.

(b) Virtual size (vbytes) vs. k at n=16 for U/PC/PS.

Figure 4: Scaling of Close for different witness designs. Complete per-k tables for $n \in \{4, 8, 16\}$ are provided in the appendix.

G Algorithm Pseudocode

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.

G.1 RevealSeeds Script

The RevealSeeds script (Algorithm 1) enforces that the evaluator reveals valid openings for all but one circuit seed and for the garbler's 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.

G.2 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.

G.3 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.

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

Function $\texttt{RevealSeedsScript}_j(\sigma_{GE}, m, \sigma_m, [seedPreimages], [inputPreimages])$**:**
  CheckMultiSigVerify$_{pk_{GE}}(\sigma_{GE})$
  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

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 $\texttt{CloseScript}_{out,j}(\sigma_{GE}, m, \sigma_m, [outPreimages])$**:**
  CheckMultiSigVerify$_{pk_{GE}}(\sigma_{GE})$
  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

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;
end
$F, e, d, O, I_G, I_E \leftarrow \pi_{Gb}(1^\kappa, f, \mathsf{seed}^*)$;
if $\pi_{Vc}(F, \mathsf{Com}_{F_{i^*}}) = 0 \ \vee\ \pi_{Vc}(d, \mathsf{Com}_{d_{i^*}}) = 0$ **then**
  return 1;
end
for $j \leftarrow 1$ **to** *outbits* **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;
  end
end
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;
  end
end
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 $1 - r$;
  end
end
return 0;