MuCPay: Breaking the Quadratic Bottleneck in Scriptless Multi-Party Oracle-Based Conditional Payments

Yaning Wang
Beihang University
China
yennywang2002@gmail.com

Na Wang
Beihang University
China
nawang@buaa.edu.cn

Chuan Li
Beihang University
China

Yongda Zhang
Beihang University
China

Junsong Fu
Beijing University of Posts and Telecommunications
China
fujs@bupt.edu.cn

Abstract

Oracle-based conditional payments triggered by real-world events support decentralized finance applications. However, dense independent bilateral deployments require $\Theta(Nn^2)$ outcome authorizations and $\Theta(n^2)$ settlement signatures for n parties and N outcomes. To go beyond two-party settings, we propose MuCPay, a multi-party oracle-based conditional payment protocol combining (i) $\Theta(Nn)$ off-chain signing shares with one on-chain settlement signature and (ii) BIP340-compatible scriptless settlement on Bitcoin. At its core, MuCPay introduces hash-bound adaptor multi-signatures (HB-AMS) that aggregate parties' conditional authorizations into a compact joint pre-signature. Its single-nonce structure and hash-bound adaptor integration enable efficient multi-party pre-signing. Unlike Double-Layer AMS (EuroS&P'26), HB-AMS produces a standard BIP340 signature under an aggregate public key, enabling direct Taproot key-path settlement. Under static corruption with one honest signer, we prove concurrent new-message unforgeability (CUF) and early-settlement resistance (ES) under our stated algebraic random-oracle model. For 128 parties and 16 outcomes, median complete local preparation takes 2.093 s. For protocol preparation (1.862s for MuCPay), native-profile projections place VweTS (NDSS'23) and ParaSwap (USENIX'25) work at 1.67×10^4 and 5.05×10^5 times MuCPay's, respectively. Settlement uses one 64-byte signature, while the modeled independent-DLC deployment requires 281.5× as many transaction vbytes. These results demonstrate practical, scalable multi-party oracle-based conditional payments.

Keywords

Oracle-Based Conditional Payments, Multi-Party, Adaptor Signatures, Adaptor Multi-Signatures, Bitcoin, Taproot

1 Introduction

Oracle-based conditional payments link settlement to real-world events, supporting decentralized finance applications such as insurance and prediction markets [37]. To avoid centralized escrow risks, these systems combine blockchains [28] with oracle services that attest event outcomes [11]. Smart contracts can implement the payment conditions, but on-chain execution incurs cost, privacy, and compatibility challenges [19, 26]. Cryptographic solutions enforce these conditions through off-chain authorization [16, 24, 26]. Discreet Log Contracts (DLCs) [16, 24] use adaptor signatures to condition payments. VweTS [26] extends the oracle model. Han et al. [21] extend this approach to joint addresses using two-party signatures. These designs target two-party payments, whereas one event may determine payments among multiple mutually distrusting parties. Extending them bilaterally requires $\Theta(Np)$ authorizations for p obligations and N outcomes, where $p = n(n-1)/2$ for n fully connected parties. Conduition's work [12] supports multiple participants with funding from a market maker, but does not support unilateral settlement in a single transaction. TUSwap and ParaSwap [35, 41] study atomic asset exchange and do not specify this conditional payment. The goal is to enable scalable, secure conditional payments for decentralized Web applications through joint multi-party authorization, avoiding redundant bilateral protocols.

Adaptor multi-signatures (AMS) let several signers prepare one signature that a witness later completes [22, 23]. Related work on threshold adaptor signatures addresses fair exchange between groups sharing keys [6], but does not directly address joint authorization of oracle-conditioned payments under independent signing keys. Khuc et al. [23] give a Double-Layer AMS construction. However, their construction does not support public-key aggregation or the Bitcoin Improvement Proposal 340 (BIP340) signature format. MuSig-style key aggregation [27, 30] and an implemented MuSig2-adaptor extension [13] provide a Bitcoin-compatible route. Our goal is to reduce local pre-signing computation and prove the conditional security needed by the payment protocol. MuSig2 unforgeability [31] does not establish that unreleased branches remain locked after another outcome is attested. This requires a separate proof under concurrent signing and malicious aborts, beyond standard multi-signature and adaptor-signature guarantees [15, 20, 25].

These observations lead to three challenges:

- **C1: Quadratic off-chain coordination.** Independent bilateral deployment repeats pre-signing across pairwise payment obligations, creating quadratically many instances.
- **C2: Prohibitive on-chain bloat.** In all-pairs payment settings, independent settlements cause quadratic transaction growth and excessive on-chain overhead.
- **C3: AMS for multi-party oracle-based conditional payments.** The signing protocol must combine efficient adaptor aggregation, BIP340-compatible aggregate-key signing, and concurrent protection of unreleased payment branches.

To address C1–C3, we propose MuCPay, a multi-party oracle-based conditional payment protocol built on hash-bound adaptor multi-signatures (HB-AMS). HB-AMS uses three rounds and one nonce per signer. Round 1 fixes nonce commitments before Round 2 derives an adaptor coefficient bound to the signing context, preventing direct cancellation of the adaptor offset by a malicious nonce. A fixed payment policy binds each authorized transaction to its oracle condition, protecting unreleased payment branches under concurrent signing *(to C3)*. For off-chain preparation, parties jointly pre-sign one transaction per outcome using a common funding unspent transaction output (UTXO), requiring $\Theta(Nn)$ signing shares instead of quadratic bilateral coordination *(to C1)*. An oracle attestation then lets any participant complete the selected settlement transaction with one such signature and settle through a Taproot key-path spend [39, 40] *(to C2)*. The on-chain signature reveals neither the signer list nor the oracle condition or script.

The main contributions are:

- **HB-AMS for Multi-Party Oracle-Based Payments (C3).** We construct HB-AMS to aggregate parties' conditional authorizations into one joint pre-signature with a single nonce per signer, reducing local pre-signing work. Adaptation produces a standard 64-byte BIP340 signature for scriptless Bitcoin settlement. For static corruption with at least one honest signer, we prove concurrent new-message unforgeability (CUF) and early-settlement resistance (ES) under discrete-logarithm (DL) hardness in the first-submission algebraic random-oracle model.
- **Scalable Multi-Party Oracle-Based Conditional Payments Protocol (C1, C2).** Building on HB-AMS, we design MuCPay to aggregate oracle-dependent payment authorizations from n mutually distrusting participants through global joint pre-signing. This replaces pairwise signing instances with one joint authorization per outcome and requires $\Theta(Nn)$ signing shares for N outcomes. A unified funding UTXO enables the realized outcome to settle in one transaction with one standard BIP340 signature, providing scriptless Taproot settlement on Bitcoin.
- **Deployment and Performance Evaluation.** We validate scriptless settlement on unmodified Bitcoin Core regtest and measure computation, communication, storage, and transaction costs. With full-view checking, local pre-signing work is 22.7% lower at $n = 64$ and 26.8% lower at $n = 256$ than our MuSig2-adaptor baseline. At $n = 128, N = 16$, MuCPay complete local preparation takes 2.093 s, and settlement uses one 64-byte signature.

2 Problem Formulation

MuCPay lets mutually distrustful parties jointly authorize payments and settle the agreed transaction upon oracle attestation. We now describe the system model, threat model, assumptions, and goals.

2.1 System Model and Authorization Policy

We use an additive group $\mathbb{G} = \langle G \rangle$ of prime order q with identity O. Scalars lie in $\mathbb{Z}_q$ (integers modulo q), with $\mathbb{Z}_q^*$ excluding zero. The function enc denotes canonical compressed point encoding. We consider the following protocol roles and assumptions.

- **Contract parties.** n mutually distrustful parties $\{\mathcal{P}_i\}_{i=1}^n$ communicate over authenticated channels, each holding x_i with $P_i = x_i G$. They agree on the event, payments, and ordered signer list $\mathcal{L}$, jointly control a funding output, and pre-sign each outcome's transaction for settlement upon the attestation.
- **Oracle interface.** The oracle attests the realized outcome of an event. For an event E, it samples $x_o, r_o \leftarrow \mathbb{Z}_q^*$ and publishes $X_o = x_o G$ and a fresh $U = r_o G$. For each possible outcome M_j, the parties compute
$$c_j = H_o(\mathrm{enc}(U), \mathrm{enc}(X_o), E, M_j), \qquad A_j = U + c_j X_o. \tag{1}$$
Setup rejects invalid or identity points and repeated challenges. When M_k occurs, the oracle releases $a_k = r_o + c_k x_o$, so $a_k G = A_k$. It releases at most one response for each U. The interface need not receive party identities or contract terms. Our cryptographic guarantees cover this prescribed interface. Incorrect outcome attestations are outside the protocol setting.
- **Underlying blockchain.** A public Bitcoin UTXO ledger provides trustless settlement. It validates Taproot key-path spends under BIP340 and prevents double-spending of the joint output, without knowledge of signer or oracle details. Transaction confirmation assumes adequate fees and ledger liveness [1].

Payment authorization policy. Before signing, parties fix the signer list $\mathcal{L}$, aggregate output key X', event E, and candidate transactions $\{Tx_j\}_{j=1}^N$ spending the same funding outpoint with agreed recipients, amounts, and fees. The frozen manifest $\mathcal{M} = \{m_j \mapsto (A_j, E, M_j)\}_{j=1}^N$ binds each distinct BIP341 transaction digest m_j to its outcome adaptor A_j. The ordinary-signing whitelist $\mathcal{F}$ contains the refund digest and is disjoint from $\{m_j\}_{j=1}^N$. The honest signer persistently enforces $\mathsf{Admit}(C) = 1$ only if the signer list and output match the frozen policy and either (i) PRE matches a manifest entry (m, A, E, M) with $\mathsf{app} = (E, M)$, or (ii) ORD has $m \in \mathcal{F}$ and $A = \mathsf{app} = \bot$. Under BIP341, settlement digests can only be authorized through PRE with their assigned adaptors.

2.2 Threat Model and Assumptions

Assumptions. We assume DL hardness, independent random oracles, authenticated channels, and persistent nonce/policy state. The ledger validates key-path spends of the fixed joint UTXO u, enforces refund maturity, and permits one final spend.

Adversary model. A malicious probabilistic polynomial-time (PPT) adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ statically corrupts $C \subsetneq [n]$, leaving at least one honest signer. It controls corrupted parties and message delivery, including concurrent sessions, delays, retries, and aborts. The payment policy $\Pi = (\mathcal{L}, X', X_o, U, E, \mathcal{M}, \mathcal{F})$ is validated and frozen as in Section 2.1. Security requires that no adversary can forge an unauthorized signature or complete an unreleased settlement branch. Formally, under the assumptions of Theorem 1, $\forall \mathcal{A} \in \mathfrak{A}_\Pi$, there exists $\nu_{\mathcal{A}} \in \mathsf{negl}$ such that $\mathrm{Adv}_{\Pi,\mathcal{A}}(\lambda) :=$

$$\Pr\left[\begin{array}{l} pp \leftarrow \mathsf{PGen}(1^\lambda); (P_i, x_i) \leftarrow \mathsf{MKGen}(pp)\ (i \in \mathcal{H}); \\ x_o, r_o \leftarrow \mathbb{Z}_q^*; \quad (X_o, U) \leftarrow (x_o G, r_o G); \\ (\Pi, \mathrm{st}) \leftarrow \mathcal{A}_0^{\{H_\ell\}}(pp, (P_i)_{i\in\mathcal{H}}, X_o, U); \\ \mathsf{Auth} = \mathsf{AuthOrd} = \mathsf{Released} = \varnothing; \\ (m, \sigma) \leftarrow \mathcal{A}_1^{O_\Pi}(pp, \Pi, \mathrm{st}) : \\ \mathsf{MVerify}(X', m, \sigma) = 1 \ \wedge\ \big(m \notin \mathsf{Auth} \vee \\ \quad \exists j \in [N] \setminus \mathsf{Released} : m = m_j \wedge m \notin \mathsf{AuthOrd}\big) \end{array}\right] \le \nu_{\mathcal{A}}(\lambda).$$

Here $\mathcal{H} = [n] \setminus C \neq \varnothing$, with a fixed $h \in \mathcal{H}$, $\mathcal{D} = \{m_j : j \in [N]\}$, and st denotes adversarial state. The adversary class is $\mathfrak{A}_\Pi = \mathsf{PPT}_{\mathrm{alg}}^{O_\Pi}$,

where O_Π = (Start, Reveal, Respond, Abort, Release, $\{H_\ell\}$) follows the interface in Appendix A.2. Honest signers enforce Admit_Π, one-use nonces, and persistent state. Auth records admitted Start requests, AuthOrd their ORD subset, and Released released outcome indices, with $|\text{Released}| \le 1$. These sets persist across aborts.

2.3 System Goals

MuCPay targets three goals under our model and assumptions.
(G1) Scalability. Joint authorization should reduce off-chain computation and communication, with signing shares linear in parties and one on-chain transaction and signature per settlement.
(G2) Deployment Compatibility. The protocol should settle with standard BIP340 signatures through the Taproot key path without changing Bitcoin consensus rules.
(G3) Multi-party conditional-payment security. Unauthorized and unreleased-outcome settlement have negligible probability. Verified preparation enables attested settlement or mature refund without further signing. See Section 4.2 for details.

3 HB-AMS: Hash-Bound Adaptor Multi-Signatures

HB-AMS enables n signers to jointly pre-sign a message m and statement A, which a witness a satisfying $A = aG$ adapts into a standard Schnorr signature [2, 17, 20].

3.1 Design Rationale and Notation

An attacker can set $R_a = uG - A$ to cancel an unweighted adaptor offset without its discrete logarithm, then bypass aggregation and share checks using the honest response. Nonce commitments cannot prevent this. HB-AMS commits all nonces before deriving $\beta = H_{\text{ad}}(C, \text{enc}(S)) \neq 0$. A committed nonce $uG - vA$ leaves $(\beta - v)A$, as β is fixed afterward. Combined with weighted key aggregation and three-round signing [27], this supports the non-Prepared/Prepared proof split.

In Bitcoin, $x(P)$ and $\text{bytes}_{32}(z)$ encode the x-coordinate and scalar $0 \le z < q$ as 32-byte big-endian values. $\|$ denotes concatenation, and x-only lifting selects even y. Asymptotically, $x(P)$ encodes $\{P, -P\}$, with two nonidentity points per encoding and lengths scaling with the security parameter.

3.2 Concrete Scheme

HB-AMS consists of (PGen, MKGen, KeyAgg, MSign, pMVerify, MAdapt, MVerify, MExt). It uses the hard relation $\mathcal{R} = \{(A, a) \in \mathbb{G} \times \mathbb{Z}_q : A = aG, A \neq O\}$, where A is a statement and a its witness. Public parameters pp are implicit after setup, and $\perp$ denotes failure. The interfaces are defined below and illustrated in Figure 1:

- $\mathsf{PGen}(1^\lambda) \to pp$: Setup outputs pp = ($\mathbb{G}$,q,G,enc,x,Encode,H_{agg}, H_{ad},H_o,H_{chal},H_{com},h_{com},H_{TapTweak}). Encode is an injective context encoding. The hashes are $H_{\text{agg}}, H_{\text{ad}}: \{0,1\}^* \to \mathbb{Z}_q^*$ for key weights and adaptor coefficients, $H_o, H_{\text{chal}}: \{0,1\}^* \to \mathbb{Z}_q$ for oracle and signing challenges, and $H_{\text{com}}: \{0,1\}^* \to \{0,1\}^{h_{\text{com}}}$ for h_{com}-bit nonce commitments. Bitcoin uses secp256k1, the BIP340 challenge in Equation (2), and $H_{\text{TapTweak}}: \{0,1\}^* \to \{0,1\}^{256}$.
- $\mathsf{MKGen}(pp) \to (P_i, x_i)$: Signer $\mathcal{P}_i$ samples its secret key $x_i \leftarrow \mathbb{Z}_q^*$ and computes its public key $P_i = x_i G$.

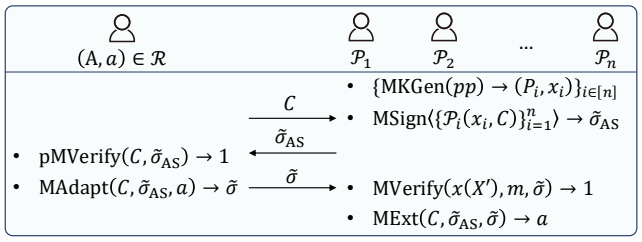


Figure 1: Flowchart of HB-AMS.

- $\mathsf{KeyAgg}(\mathcal{L}) \to (X', \{t_i\}, \gamma, \tau_{\text{eff}})/\perp$: For fixed $\mathcal{L}$, first reject invalid, identity, or duplicate keys. Any participant computes $t_i = H_{\text{agg}}(\mathcal{L}, \text{enc}(P_i))$, aggregate point $K = \sum_i t_i P_i$, and $P = \kappa_0 K$, with $\kappa_0 \in \{\pm 1\}$ chosen to make P even. With no script path, it computes $\tau = \text{int}(H_{\text{TapTweak}}(x(P)))$, tweaked point $Q_{\text{raw}} = P + \tau G$, and output key $X' = \kappa_1 Q_{\text{raw}}$, with $\kappa_1 \in \{\pm 1\}$ chosen to make X' even. The effective sign $\gamma = \kappa_0 \kappa_1$ and tweak $\tau_{\text{eff}} = \kappa_1 \tau$ satisfy $X' = \gamma \sum_i t_i P_i + \tau_{\text{eff}} G$. It rejects $K = O$, $\tau \ge q$, or $Q_{\text{raw}} = O$.
- $\mathsf{MSign}\langle\{\mathcal{P}_i(x_i, C)\}_{i=1}^n\rangle \to \tilde{\sigma}_{\text{AS}}/\perp$: The signers fix the immutable context $C = \text{Encode}(\text{mode}, sid, \mathcal{L}, X', m, A, \text{app})$, where mode is PRE or ORD, sid identifies the session, and app = (E, M) for PRE or $\perp$ for ORD. Each signer validates the encodings, $\mathsf{KeyAgg}(\mathcal{L})$, and $\mathsf{Admit}(C)$, rejects zero adaptors in PRE and reused session identifiers, and durably reserves the fresh sid.
  - **Round 1: Commitment Phase.** Signer $\mathcal{P}_i$ samples a fresh nonce scalar $r_i \leftarrow \mathbb{Z}_q^*$ and uniform λ-bit salt η_i, stores them with C and nonce point $R_i = r_i G$, and sends commitment $D_i = H_{\text{com}}(C, i, \text{enc}(R_i), \eta_i)$. Before opening, it freezes the signer-indexed commitment vector $\mathbf{D} = (D_j)_j$, requiring one entry per signer and its own entry equal to D_i.
  - **Round 2: Reveal Phase.** Each signer reveals (R_i, η_i) once, checks every opening against $(C, \mathbf{D})$, and rejects malformed or identity points. It computes aggregate nonce $S = \sum_i R_i$, adaptor coefficient $\beta = H_{\text{ad}}(C, \text{enc}(S))$, and final nonce $R' = \delta(S + \beta A)$, choosing $\delta \in \{\pm 1\}$ to make R' even. It rejects $S = O$, failed or zero β, or $S + \beta A = O$. The BIP340 challenge $c = H_{\text{chal}}(x(X'), m, x(R'))$ is
$$c = \text{int}\big(H_{\text{BIP0340/challenge}}(x(R') \| x(X') \| m)\big) \bmod q. \tag{2}$$
  - **Round 3: Response Phase.** Each signer $\mathcal{P}_i$ computes its partial response $\hat{s}_i$, which satisfies
$$\hat{s}_i = \delta r_i + c\gamma t_i x_i, \qquad \hat{s}_i G = \delta R_i + c\gamma t_i P_i. \tag{3}$$
    It durably consumes the nonce and erases r_i before sending $\hat{s}_i$.
  - **Aggregation Phase.** The aggregator verifies all partial responses using Equation (3) and computes the pre-signature scalar $\tilde{s}_{\text{AS}}$ and joint pre-signature $\tilde{\sigma}_{\text{AS}}$:
$$\tilde{s}_{\text{AS}} = \sum_{i=1}^{n} \hat{s}_i + c\tau_{\text{eff}}, \qquad \tilde{\sigma}_{\text{AS}} = (\text{enc}(S), \tilde{s}_{\text{AS}}). \tag{4}$$
    Each recipient stores $(C, \tilde{\sigma}_{\text{AS}})$ only after pMVerify succeeds.

On failure, a signer aborts, erases its secret nonce, and returns $\perp$. Completed or aborted sessions reject further messages. Retries require a fresh sid, nonce, and salt (see Appendix A.2). In ORD mode, OrdSign follows the same rounds for whitelisted messages with A = app = $\perp$, omits H_{ad}, and sets even $R' = \delta S$, rejecting

$S = \mathcal{O}$. Using Equations (2) and (4), it outputs $x(R')\|\,\mathrm{bytes}_{32}(\tilde{s}_{\mathsf{AS}})$ only if MVerify succeeds.

- $\mathsf{pMVerify}(C, \tilde{\sigma}_{\mathsf{AS}}) \to \{0,1\}$: The verifier checks PRE mode, the pre-signature $(\mathrm{enc}(S), \tilde{s}_{\mathsf{AS}})$, the key list and tweak, and all non-degeneracy conditions. It recomputes β, δ, R', c from C, S and returns 1 exactly when all checks pass and $\tilde{s}_{\mathsf{AS}}G = R' - \delta\beta A + cX'$. It returns 0 otherwise.
- $\mathsf{MAdapt}(C, \tilde{\sigma}_{\mathsf{AS}}, a) \to \tilde{\sigma}/\perp$: A party with witness a checks $\mathsf{pMVerify} = 1$ and $aG = A \neq \mathcal{O}$, then computes signature scalar $\tilde{s} = \tilde{s}_{\mathsf{AS}} + \delta\beta a \pmod q$ and signature $\tilde{\sigma} = x(R')\|\,\mathrm{bytes}_{32}(\tilde{s})$. It returns $\tilde{\sigma}$ if $\mathsf{MVerify} = 1$, and $\perp$ otherwise.
- $\mathsf{MVerify}(x(X'), m, \tilde{\sigma}) \to \{0,1\}$: The verifier applies BIP340 by checking the 64-byte signature, x-only key lift, field and scalar ranges, and challenge c in Equation (2). It accepts when $\tilde{s}G - cX'$ is nonidentity, even, and has the encoded nonce coordinate, and returns 0 on any failed check.
- $\mathsf{MExt}(C, \tilde{\sigma}_{\mathsf{AS}}, \tilde{\sigma}) \to a/\perp$: The extractor verifies both signatures under the same key and message, requires matching final nonce coordinates $x(R')$, and computes the witness $a = (\delta\beta)^{-1}(\tilde{s} - \tilde{s}_{\mathsf{AS}})$. It returns a if $aG = A$, and $\perp$ otherwise.

For the following analysis, let $\mathsf{PV}_C(\hat{\sigma}) = \mathsf{pMVerify}(C, \hat{\sigma})$ and $\mathsf{V}_m(\sigma) = \mathsf{MVerify}(X', m, \sigma)$, with $\mathsf{V}_m(\perp) = 0$. Let rn denote the canonical final nonce coordinate, derived from $(C, \hat{\sigma})$ for a pre-signature. We establish the basic adaptor-signature properties of HB-AMS in Proposition 1, with proofs in Appendix A.1.

Proposition 1 (Basic adaptor-signature properties)**.** For every admitted PRE context C and $(A, a) \in \mathcal{R}$, HB-AMS satisfies:

(i) *Correctness:* $\Pr[\hat{\sigma} \neq \perp \wedge \mathsf{PV}_C(\hat{\sigma}) = 0] = 0$ for an honest execution $\hat{\sigma} \leftarrow \mathsf{MSign}(C)$.
(ii) *Adaptability:* $\mathsf{PV}_C(\hat{\sigma}) = 1 \Rightarrow \mathsf{V}_m(\mathsf{MAdapt}(C, \hat{\sigma}, a)) = 1$.
(iii) *Matching-nonce extraction:* $\mathsf{PV}_C(\hat{\sigma}) = \mathsf{V}_m(\sigma) = 1$ and $\mathsf{rn}(C, \hat{\sigma}) = \mathsf{rn}(\sigma)$ imply $\mathsf{MExt}(C, \hat{\sigma}, \sigma) = a$.
(iv) *Single-target witness extractability* (WE_1): $\forall \mathcal{A} \in \mathfrak{A}_\Pi,\ \exists \nu_{\mathcal{A}} \in \mathsf{negl} : \mathrm{Adv}_{\mathsf{WE}_1}(\mathcal{A}; \lambda) \le \nu_{\mathcal{A}}(\lambda)$, where WE_1 admits one target-message Start and a registered complete target transcript (Appendix A.2).

3.3 Security Analysis

All games use $\mathcal{O}_\Pi$ of Section 2.2. Each manifest message admits polynomially many fresh-*sid* PRE sessions, including interleaved, incomplete, and aborted executions.

Definition 1 (Concurrent new-message unforgeability, CUF)**.** CUF prevents an adversary from producing a valid signature on any message not authorized by an honest signer, even across concurrent signing sessions. For $(m, \sigma) \leftarrow \mathcal{A}^{\mathcal{O}_\Pi}$, define $W_{\mathsf{CUF}} := \mathsf{V}_m(\sigma) \wedge [m \notin \mathsf{Auth}]$ and $\mathrm{Adv}_{\mathsf{CUF}} := \Pr[W_{\mathsf{CUF}} = 1]$.

Definition 2 (Early-settlement resistance, ES)**.** ES prevents settlement before the corresponding oracle witness is released, even after another outcome is attested and regardless of the signature nonce. For $(j, \sigma) \leftarrow \mathcal{A}^{\mathcal{O}_\Pi}$, define $W_{\mathsf{ES}} := [j \in [N]] \wedge \mathsf{V}_{m_j}(\sigma) \wedge [j \notin \mathsf{Released}] \wedge [m_j \notin \mathsf{AuthOrd}]$ and $\mathrm{Adv}_{\mathsf{ES}} := \Pr[W_{\mathsf{ES}} = 1]$.

The following theorem establishes security of Definitions 1 and 2 for HB-AMS under concurrent signing.

Theorem 1 (Concurrent security of HB-AMS)**.** Assume DL hardness in an odd-prime-order group family with efficient canonical encodings and normalization of $\{P, -P\}$. In the model of Section 2.2, with scalar random oracles, let $q = 2^{\Omega(\lambda)}$ and $h_{\mathrm{com}} = \Omega(\lambda)$. For polynomial resources, $\forall \mathcal{A}_U, \mathcal{A}_E \in \mathfrak{A}_\Pi,\ \exists \nu_{\mathcal{A}_U, \mathcal{A}_E} \in \mathsf{negl} : \mathrm{Adv}_{\mathsf{CUF}}(\mathcal{A}_U; \lambda) + \mathrm{Adv}_{\mathsf{ES}}(\mathcal{A}_E; \lambda) \le \nu_{\mathcal{A}_U, \mathcal{A}_E}(\lambda)$.

Proof sketch. The proof records $(x(X'), m, x(R'))$ in Prepared when frozen nonce commitments determine the aggregate nonce, before revealing the honest nonce; records persist across aborts. Every CUF win is non-Prepared. Such outputs, including non-Prepared ES wins, reduce to random-target identification discrete logarithm (XIDL), and hence to DL [29, Lemma 4]. For Prepared ES wins, the committed nonce contributions and fresh adaptor coefficient enable extraction of an embedded oracle DL challenge, using separate embeddings before and after release. Theorem 3 bounds the reduction losses and failure probabilities, which yield negligible advantages under the stated assumptions. The full proof is given in Appendices A.2–A.7.

These two guarantees jointly bound the attack advantage in Section 2.2. Given $\mathcal{A}$'s output (m, σ), let $\mathcal{A}_U$ return it unchanged and $\mathcal{A}_E$ return (j, σ) for the unique manifest entry $m = m_j$, or $\perp$ otherwise. The joint winning event is the union of their CUF and ES events, so Theorem 1 and the union bound give, for some $\nu_{\mathcal{A}} \in \mathsf{negl}$,

$$\mathrm{Adv}_{\Pi,\mathcal{A}}(\lambda) \le \mathrm{Adv}_{\mathsf{CUF}}(\mathcal{A}_U; \lambda) + \mathrm{Adv}_{\mathsf{ES}}(\mathcal{A}_E; \lambda) \le \nu_{\mathcal{A}}(\lambda).$$

Proposition 3 extends the bound to the binary profile with the stated statistical error. MuCPay combines CUF, ES, and adaptability (Proposition 1(ii)) in the security analysis of Section 4.2.

4 MuCPay: Multi-Party Oracle-Conditioned Payments

This section presents MuCPay, a multi-party oracle-conditioned payment protocol built on HB-AMS. It generates one joint pre-signature per outcome with $\Theta(Nn)$ signing shares and settles the selected outcome through a single BIP340-compatible Taproot signature. Figure 2 illustrates its six-phase workflow.

4.1 Concrete Protocol via HB-AMS

MuCPay enables n mutually distrustful parties $\{\mathcal{P}_i\}_{i=1}^n$ to prepare payments for N possible outcomes and settle the attested outcome from one shared UTXO.

Phase ❶: Attestation setup. For event E, the oracle samples $x_o, r_o \leftarrow \mathbb{Z}_q^*$ and publishes its public key $X_o = x_o G$ and fresh nonce commitment $U = r_o G$. It keeps both scalars secret.

Phase ❷: Contract and transaction setup.

(1) The parties agree on E, possible outcomes $\{M_j\}_{j=1}^N$, payments, and refund terms. Each party obtains (P_i, x_i) from MKGen. They fix $\mathcal{L} = (P_i)_{i=1}^n$ and derive X' via $\mathsf{KeyAgg}(\mathcal{L})$.
(2) They construct the unsigned funding transaction Tx_{fund}, which pools their collateral into a designated output u under $x(X')$. Its native SegWit inputs use wallet keys independent of the HB-AMS keys. Fixing all non-witness fields determines its txid and hence u before broadcast.
(3) For each M_j, they derive (c_j, A_j) using Equation (1) and construct Tx_j spending u to the agreed recipients with the agreed

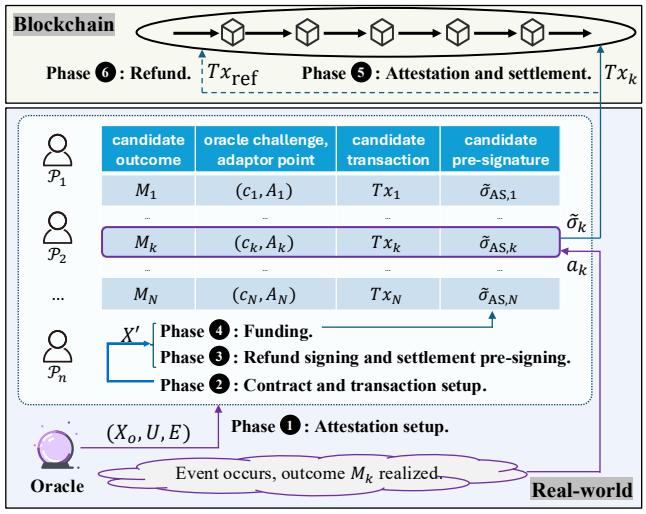


Figure 2: Workflow of MuCPay.

amounts and fees. They also construct Tx_{ref} spending u according to the refund terms, with an absolute locktime and non-final input sequence enforcing expiration.

(4) Each party validates the points and transactions, rejecting zero adaptors and repeated challenges. They compute the distinct BIP341 digests m_j and refund digest m_{ref} using SIGHASH_DEFAULT without an annex, then freeze the outcome manifest $\mathcal{M} = \{m_j \mapsto (A_j, E, M_j)\}_{j=1}^{N}$ and refund-only signing whitelist $\mathcal{F} = \{m_{\text{ref}}\}$.

Phase 3: Refund signing and settlement pre-signing.

(1) The parties run OrdSign on m_{ref} with $A = \text{app} = \perp$ to sign Tx_{ref}. For each (m_j, A_j), they run MSign with app $= (E, M_j)$ and a fresh *sid*, nonce, and salt, obtaining the joint pre-signature $\tilde{\sigma}_{\text{AS},j}$.

(2) Each party verifies the refund signature via MVerify and every pre-signature via pMVerify. It durably stores the signed refund and all $(C_j, \tilde{\sigma}_{\text{AS},j})$ before proceeding to funding. If any check fails, it aborts without signing its funding inputs.

Phase 4: Funding. Each party signs its inputs of Tx_{fund} using independent wallet keys, with signatures committing to all inputs and outputs. Parties then broadcast Tx_{fund}. Once confirmed, the collateral is locked in the joint output u.

Phase 5: Attestation and settlement. When M_k occurs, the oracle publishes $a_k = r_o + c_k x_o \bmod q$. Any party verifies $a_k G = A_k$ and computes $\tilde{\sigma}_k = \text{MAdapt}(C_k, \tilde{\sigma}_{\text{AS},k}, a_k)$. After verifying $\tilde{\sigma}_k$ with MVerify, the party inserts it into the Taproot key-path witness of Tx_k and broadcasts the transaction. Upon confirmation, Tx_k distributes the funds according to outcome M_k.

Phase 6: Refund. If u remains unspent when Tx_{ref} matures, any party can broadcast the stored, fully signed refund without further signing. A late Tx_k may race the refund, but only one can confirm.

4.2 Security Analysis

Let $\Gamma = \{Tx_1, \cdots, Tx_N, Tx_{\text{ref}}\}$, $\mathcal{F} = \{m_{\text{ref}}\}$, and Ready_h denote verified, durable preparation before honest funding. At step t, let $P_t = \text{Tpl}_u(Tx_t, w_t)$, $(m_t, \sigma_t) = \text{Parse}_u(Tx_t, w_t)$, and $V_t = \text{V}_{m_t}(\sigma_t)$.

Definition 3 (Multi-party conditional-payment security). For $\mathcal{A} \in \mathfrak{A}_\Pi$ controlling delivery, preparation inputs, and key-path spend submissions for u:

(i) *Settlement safety:* Set $U_t := \text{Active}(u) \wedge V_t \wedge [P_t \notin \Gamma]$, $C_t := \text{Active}(u) \wedge V_t \wedge \bigvee_{j \in [N] \setminus \text{Released}_t} [P_t = Tx_j]$, and $L := \text{EarlyRef} \vee \text{Double}$, denoting early refund confirmation or two final conflicting spends. Define $W_{\text{safe}} := [\exists t : U_t \vee C_t] \vee L$ and $\text{Adv}^{\text{safe}}_{\text{MP}}(\mathcal{A}; \lambda) := \Pr[W_{\text{safe}} = 1]$. The game stops at the first winning step.

(ii) *Settlement verifiability:* For stored $(C_j, \hat{\sigma}_j)$, and supplied a, let $\sigma_j = \text{MAdapt}(C_j, \hat{\sigma}_j, a)$. Define $W_{\text{ver}} := \text{Ready}_h \wedge [(A_j, a) \in \mathcal{R}] \wedge [\text{V}_{m_j}(\sigma_j) = 0]$ and $\text{Adv}^{\text{ver}}_{\text{MP}}(\mathcal{A}; \lambda) := \Pr[W_{\text{ver}} = 1]$.

The games adapt conditional-payment one-wayness and verifiability [26, Definitions 7–8].

For the following analysis, let $\epsilon_{\text{bind}} = \Pr[\text{BadBind}]$ and $\epsilon_{\text{led}} = \Pr[\text{BadLed}]$ for the transaction-binding and ledger failure probabilities, respectively, with events defined below.

Theorem 2 (Multi-party conditional-payment security of MuCPay). Under Theorem 1, $\forall \mathcal{A} \in \mathfrak{A}_\Pi$, $\exists \mathcal{B}_U, \mathcal{B}_C$ such that

$$\text{Adv}^{\text{safe}}_{\text{MP}}(\mathcal{A}) \le \text{Adv}_{\text{CUF}}(\mathcal{B}_U) + \text{Adv}_{\text{ES}}(\mathcal{B}_C) + \epsilon_{\text{bind}} + \epsilon_{\text{led}}. \quad (5)$$

If $\epsilon_{\text{bind}}, \epsilon_{\text{led}} \in \text{negl}$, then $\text{Adv}^{\text{safe}}_{\text{MP}} \in \text{negl}$. Moreover, $\text{Adv}^{\text{ver}}_{\text{MP}} = \Pr[\text{Fund}_h \wedge \neg \text{Ready}_h] = 0$. The binary-profile bound follows by applying Proposition 3 to both signature advantages.

Proof. **Preparation.** Initialize $\text{Active}(u) = 0$ and run $\mathcal{O}_\Pi$. Parse_u extracts the actual BIP341 digest and canonical BIP340 signature, returning $\perp$ on failure. Tpl_u records the SIGHASH_DEFAULT-committed fields, actual hash type, and annex, excluding signature bytes; agreed templates use this hash type without an annex.

Store $\text{Pre}_h[j] = (C_j, \hat{\sigma}_j)$ iff $j \in [N]$, C_j is admitted PRE for $(m_j, A_j, (E, M_j))$, and $\text{PV}_{C_j}(\hat{\sigma}_j) = 1$; store the refund iff $\text{V}_{m_{\text{ref}}}(\sigma_{\text{ref}}) = 1$. Let $\text{PreOK}_h(j)$ and RefOK_h indicate successful storage, and set $\text{Ready}_h := \text{CheckTx}(u, \Gamma) \wedge \text{RefOK}_h \wedge \bigwedge_{j \in [N]} \text{PreOK}_h(j)$. CheckTx requires $\text{Inputs}(T) = \{u\}$ for every $T \in \Gamma$, checks agreed payments, fees, digests, and refund locktime/sequence, and validates all non-signature conditions except funding availability and refund maturity. Fund_h requires $\text{Ready}_h = 1$ and authorizes only Tx_{fund}; ledger acceptance activates u. Crashes preserve preparation and policy.

Safety. Let $U := \bigvee_t U_t$ and $C := \bigvee_t C_t$. Define $\text{BadBind} := \bigvee_{t:\text{Parse}_u \neq \perp} ([P_t \notin \Gamma] \wedge [m_t \in \mathcal{D} \cup \mathcal{F}])$ for BIP341 binding failures [40], $\text{EarlyRef} := [\exists t : \text{Final}_t(Tx_{\text{ref}}, u) \wedge \neg \text{Mature}_{\text{ref}}(t)]$, and $\text{Double} := [\exists t, t', T \neq T' : \text{Final}_t(T, u) \wedge \text{Final}_{t'}(T', u)]$. BadLed covers invalid confirmed key-path spends, EarlyRef, and Double.

Both reductions generate independent honest funding-wallet keys, sign funding inputs locally, forward $\mathcal{O}_\Pi$, and simulate storage and ledger checks. At the first U_t, $\mathcal{B}_U$ outputs (m_t, σ_t). Outside BadBind, we have $m_t \notin \mathcal{D} \cup \mathcal{F}$. Since $\text{Auth} \subseteq \mathcal{D} \cup \mathcal{F}$, this implies $\Pr[U \wedge \neg \text{BadBind}] \le \text{Adv}_{\text{CUF}}(\mathcal{B}_U)$.

At the first C_t, $\mathcal{B}_C$ outputs (j, σ_t). Here $m_t = m_j$, $j \notin \text{Released}_t$, and $m_j \notin \text{AuthOrd}$ because $\text{AuthOrd} \subseteq \mathcal{F}$ and $\mathcal{F} \cap \mathcal{D} = \emptyset$. Hence $\Pr[C] \le \text{Adv}_{\text{ES}}(\mathcal{B}_C)$. Since $\{W_{\text{safe}} = 1\} \subseteq (U \wedge \neg \text{BadBind}) \cup C \cup \text{BadBind} \cup \text{BadLed}$, the union bound yields Equation (5), without guessing a session or outcome.

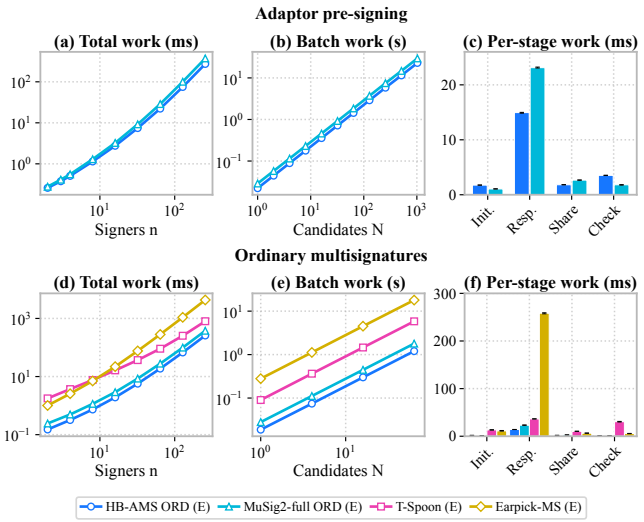


Figure 3: Evaluation of HB-AMS signing computation. Panels (a–c) show adaptor signing and (d–f) ordinary signing. Batches use $n = 64$; stage costs use $n = 64, N = 1$.

Verifiability and recovery. The storage and funding rules preserve $\mathsf{Fund}_h \Rightarrow \mathsf{Ready}_h$ under crashes, giving $\Pr[\mathsf{Fund}_h \wedge \neg\mathsf{Ready}_h] = 0$. For each stored branch, $\mathsf{Ready}_h \Rightarrow \mathsf{PV}_{C_j}(\hat{\sigma}_j) = 1$; Proposition 1(ii) then gives $\mathsf{Ready}_h \wedge [(A_j, a) \in \mathcal{R}] \Rightarrow \mathsf{V}_{m_j}(\mathsf{MAdapt}(C_j, \hat{\sigma}_j, a)) = 1$, so $\mathsf{Adv}^{\mathsf{ver}}_{\mathsf{MP}} = 0$. Writing Valid_t for ledger validation, $\mathsf{Ready}_h \wedge \mathsf{Unspent}(u, t) \wedge \mathsf{Mature}_{\mathsf{ref}}(t) \Rightarrow \mathsf{Valid}_t(Tx_{\mathsf{ref}}, \sigma_{\mathsf{ref}}) = 1$. No further signing is needed; confirmation follows the stated fee/liveness assumptions. □

5 Implementation and Evaluation

We address five research questions (RQs). E denotes measured execution or serialization, T analytical counts, and M model projections.

- **RQ1:** How does HB-AMS compare with baselines in computational cost and scalability?
- **RQ2:** How does joint authorization reduce repeated off-chain work as participants and outcomes grow?
- **RQ3:** How much communication and storage does MuCPay require compared with bilateral deployments?
- **RQ4:** Does unmodified Bitcoin Core accept settlement and enforce rejection and refund rules?
- **RQ5:** How does joint settlement reduce on-chain transactions, witnesses, and fees as participants grow?

5.1 Implementation and Evaluation Methodology

The Python prototype uses native libsecp256k1 v0.7.1 [9] on an Apple M5 Pro (15 physical cores, 24 GiB memory). Timings measure sequential local work across roles, including encoding and validation but excluding network and confirmation delays. Component sweeps use $n \in \{2, 3, 4, 8, 16, 32, 64, 128, 256\}$ at $N = 1$, while system sweeps use $n \in \{3, 4, 8, 16, 32, 64, 128\}$ at $N = 16$. Outcome sweeps use $N \in \{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024\}$ at $n = 8, 64$ for components and $n = 8$ for systems. Ordinary-signing batches use $N \in \{1, 4, 16, 64\}$ at $n = 64$. Each party contributes 10^7 satoshis, with fees calculated at 2 sat/vbyte from serialized sizes.

We report medians and interquartile ranges (IQRs) over 31 executions, reduced to 11 for batches, system configurations with $pN > 50{,}000$, native VweTS, reconstructed Vwe2psTS, and the cached shared contract execution transaction (CET) control. System-figure arrows show reference-to-MuCPay cost ratios. Regtest uses real UTXOs, while local benchmarks use mock outpoints. Scalar arithmetic is variable-time, and nonce state is volatile.

5.2 Evaluation of HB-AMS (to RQ1)

We measure checked pre-signing, completion, and ordinary signing with cached keys and fresh nonces. The coordinator checks every share; recipients check the aggregate. Delivered payloads exclude setup, contexts, authentication, and framing. Figure 3 sweeps signers and distinct-message batches at $n = 64$. Opening is included in HB-AMS totals but omitted from stage bars.

Baseline. Full-view MuSig2-adaptor follows the public equations and Bitcoin Improvement Proposal 327 [13, 30, 31]. HB-AMS ORD is compared with full-view MuSig2 and our common-backend T-Spoon [5] and Earpick-MS [4] implementations. The latter use explicit hash-to-point and proof encodings without author-code certification or dedicated multiscalar multiplication and batch verification; their security models differ from HB-AMS's. Table 1 lists Ji and Khuc analytically in their native costs and security notions.

Earpick-MS requires each signer to verify all n discrete-logarithm-equality proofs (n^2 checks per session); T-Spoon adds matrix-commitment costs. These costs explain the larger gaps in Figure 3(d–f) than against MuSig2 alone.

Table 1: Evaluation of AMS costs. Source security notions are unforgeability (UF), full extractability (FE), and unlinkability (UL).

Scheme	Security and deployment		Computation			Interaction and communication		
	Scope	BIP340	F	V	J	Pattern	Payload	Unit / scope
Ji Schnorr AMS [22]	UF, WE	–	Nonce: vF/signer Pre-verify: F + V			2 broadcasts	$v\ell_{\mathbb{G}} + \ell_p$	bits/signer
Khuc AMS [23]	FE, UL	–	Transcript: $n(T+1)$A Verify: $(T+1)$A			n sequential stages + opening/routing	$(T+1)\ell_{\mathcal{X}}$ $+\ell_{\mathcal{G}} + 2\lambda$	bits/hop
MuSig2-adaptor	No manifest ES claim	•	$4n$	$4n+1$	$2n^2 + 6n$	2 rounds	$66n^2 + 163n$	B delivered
HB-AMS	CUF, ES	•	$3n$	$4n+1$	$n^2 + 4n$	3 rounds	$97n^2 + 194n$	B delivered

F/V/J: fixed-/variable-base elliptic-curve multiplication/addition; A: group action. •/–: BIP340 profile specified/unspecified. v: unspecified nonce count; T: repetitions.

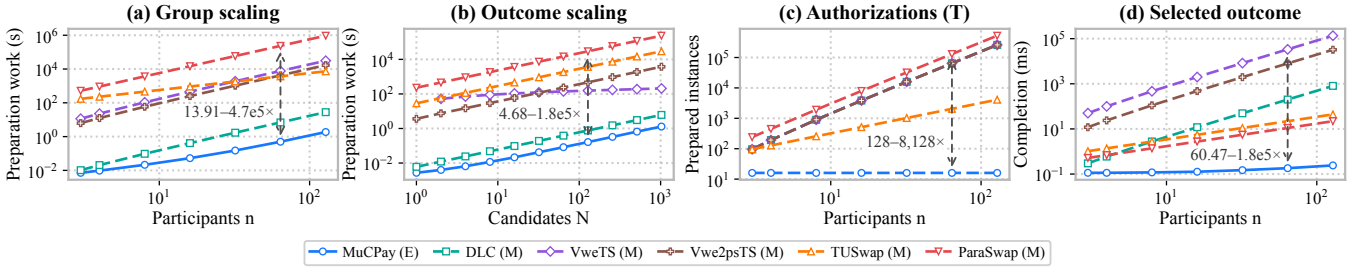


Figure 4: Evaluation of MuCPay protocol preparation and selected-outcome completion. Panels (a,c,d) use $N = 16$; (b) uses $n = 8$.

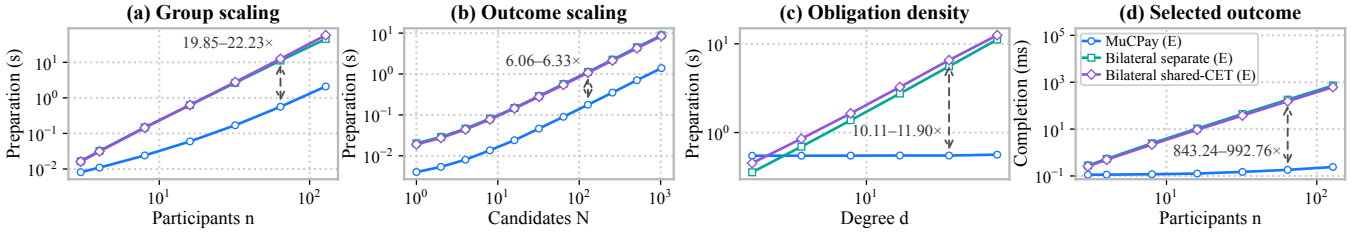


Figure 5: Evaluation of MuCPay payment preparation and completion. Panels (a,d) use $N = 16$; (b) uses $n = 8$; (c) uses $n = 64$, $N = 16$ and obligation degrees $d \in \{2, 4, 8, 16, 32, 63\}$.

Results. Checked pre-signing takes 22.18 versus 28.70 ms at $n = 64$, and 272.55 versus 372.41 ms at $n = 256$, reducing costs by 22.7% and 26.8%. Table 1 attributes the savings to n fewer fixed-base multiplications and $n^2 + 2n$ fewer additions; full-view checking remains quadratic. At $n = 64$, ordinary signing takes 18.77 ms, versus 27.84, 90.72, and 281.70 ms for MuSig2, T-Spoon, and Earpick-MS. Figure 3(c) traces the adaptor savings: at $n = 64$, response derivation takes 14.95 ms for HB-AMS versus 23.13 ms for MuSig2-adaptor, outweighing the recipient-check increase from 1.81 to 3.53 ms in the measured total.

HB-AMS requires one additional round and more communication. At $n = 8, N = 1$ with 20 ms injected round-trip time, colocated TCP emulation takes 76.09 versus 52.38 ms. At $n = 64$, pre-verification, adaptation, and extraction take 56.8/82.0/89.9 μs, versus 29.3/60.7/57.5 μs for MuSig2-adaptor.

5.3 Evaluation of MuCPay (to RQ2–RQ5)

Table 2 reports protocol operation and communication counts under shared public setup. Communication excludes redistribution and transaction bytes. Completion measures recovery or adaptation of pre-verified material. Completion and on-chain costs refer to the normal settlement path.

5.3.1 Off-chain Computation and Scalability (RQ2). We measure protocol preparation, selected-outcome completion, and the full lifecycle: key/oracle validation, transaction construction, signed refunds, candidate authorizations, and funding signatures. Dense workloads use $p = n(n-1)/2$ obligations; the density sweep uses $p = nd/2$ at $n = 64, N = 16$.

Baseline. Separate-settlement and shared-CET controls match the backend, event, collateral, and net payments, retaining per-obligation authorization. Published-scheme projections repeat native DLC [32] over p pairs, native VweTS [18, 26] over $2p$ directed payers, and reconstructed Vwe2psTS [21] over p pairs. Vwe2psTS uses one oracle, $d = 20$, secp256k1/BLS12-381, and independent 64-message batches beyond the measured range. TUSwap/ParaSwap [35, 41] replay independent outcome branches calibrated from author-code timed setup and checked authorization [42]. These M curves sum native-profile local work (VweTS: 128-bit; swap timed setup: 1024-bit Rivest–Shamir–Adleman (RSA)), without establishing equal-security or end-to-end speedups.

Results. MuCPay prepares joint authorizations using Nn signing shares, while bilateral deployments repeat the authorization process across p payment obligations (Figure 4). Panel (c) counts joint authorizations versus bilateral signatures/shares. At $n = 128, N = 16$, preparation takes 1.862 s (E), versus projected 31,009.107 s for VweTS and 940,350.705 s for ParaSwap (M).

For complete preparation (Figure 5), MuCPay takes 2.093 s at $n = 128, N = 16$, versus 45.158 and 57.833 s for separate and shared bilateral settlement (21.6 and 27.6 times MuCPay's work). At $n = 8, N = 1024$, costs are 1.391, 8.842, and 8.505 s, respectively. Bilateral work also increases with obligation density.

Table 2: Evaluation of system-wide operation and communication counts for N outcomes and n parties.

Protocol	Adaptor/ timed setup	Sign./ enc. shares	Selected compl.	Orig. share bytes	On-chain sig. checks
DLC [14]	$\Theta(Nn^2)$	$\Theta(Nn^2)$	$\Theta(n^2)$	$\Theta(Nn^2)$	$\Theta(n^2)$
VweTS [26]	$O(BNn^2)$	$O(BNn^2)$	$\Theta(n^2)$	$O(BNn^2)$	$\Theta(n^2)$
Vwe2psTS [21]	$\Theta(CNn^2)$	$\Theta(CNn^2)$	$\Theta(n^2)$	$\Theta(CNn^2)$	$\Theta(n^2)$
TUSwap† [35]	$\Theta(Nn)$	$\Theta(Nn)$	$\Theta(n)$	$\Theta(Nn)$	$\Theta(n)$
ParaSwap† [41]	$\Theta(Nn^2)$	$\Theta(Nn^2)$	$\Theta(n)$	$\Theta(Nn^2)$	$\Theta(n)$
Our MuCPay	$\Theta(N)$	$\Theta(Nn)$	$\Theta(1)$	$\Theta(Nn)$	$\Theta(1)$

B: VweTS bucket size; $C = 40$: Vwe2psTS ciphertexts per outcome/pair.
†: independent N-branch swap extensions.

At $n = 128, N = 16$, authorization takes 1.233 s for MuCPay, versus 36.486 s for separate and 35.888 s for shared-CET settlement. Shared-CET retains most authorization work and raises transaction construction from 3.476 to 16.956 s: multi-input candidate processing increases preparation cost despite on-chain savings.

At $n = 128, N = 16$, selected completion takes 0.24 ms, versus 721.30 and 617.42 ms for separate and shared-CET settlement. One attestation completes MuCPay's joint witness without further signing, while controls complete each obligation separately. Joint authorization removes this repetition.

At fixed $n = 64, N = 16$, raising d from 2 to 63 increases p from 64 to 2,016. Bilateral controls prepare more obligation-specific material as payment density increases, while MuCPay retains one joint payment per outcome. This explains its nearly flat density curve (Figure 5(c)).

5.3.2 Off-chain Communication and Storage (RQ3). We count keys, oracle metadata, funding, refunds, candidates, and authorizations under the specified holder distribution. A tested lossless codec shares fixed input templates and payment scripts while preserving transaction bytes, BIP341 digests, and signing contexts. Counts exclude transport framing, authentication, wallet journals, and heap overhead.

Baseline. Both bilateral controls use the same codec and accounting. Published schemes' candidate-only payloads omit transactions and recovery material, precluding complete-system comparisons.

Results. At $n = 128, N = 16$, MuCPay delivers 29.68 MiB, versus 132.38 and 1,019.59 MiB for separate and shared-CET settlement, reductions of 77.6% and 97.1%. Maximum retained completion material is 41.39 KiB, compared with 602.61 and 10,261.72 KiB for the controls. Each MuCPay party retains $65N$ pre-signature bytes and $\Theta(Nn)$ payment material. Joint authorization reduces duplicated contract state, while full-view delivery still costs $\Theta(Nn^2)$.

Shared-CET holders still need candidates and input authorizations; combining inputs can increase delivered and retained material under this holder policy. MuCPay instead shares fixed transaction templates and retains one joint pre-signature per outcome: 1,040 bytes per party at $N = 16$. The larger retained total includes outcome-dependent payments and recovery material.

5.3.3 Bitcoin Deployment Compatibility (RQ4). We test real-UTXO payment workflows on unmodified Bitcoin Core 29.0 regtest [8] at $(n, N) = (4, 16), (16, 4)$.

Baseline. Native DLC CET and refund transactions are tested on the same node.

Results. Joint funding and attested settlement pass with one 64-byte key-path witness and no further participant signing; altered payments or witnesses are rejected. The refund signed before funding is rejected before locktime, accepted after maturity, and rejected after confirmed settlement. Native DLC CETs and mature refunds also pass. All 32 recorded cases are in the artifact, answering RQ4 for standard transaction and locktime acceptance.

Bitcoin verifies the adapted signature against the joint output key without an on-chain oracle instruction; the prepared signing context enforces the oracle condition off-chain. Refund tests exercise recovery from the same output. Together, valid-path acceptance and altered/conflicting-spend rejection test the payment lifecycle beyond isolated Schnorr signature verification.

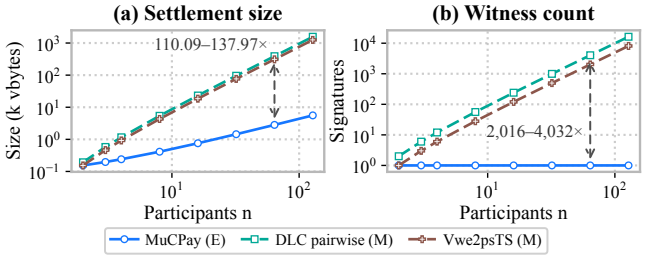


Figure 6: Evaluation of on-chain settlement size and signature checks. Ratios are marked at $n = 64$.

5.3.4 On-chain Execution Cost and Scalability (RQ5). We serialize selected settlements at 2 sat/vbyte, excluding funding and refunds. MuCPay spends one joint output to n recipients. For $n < 253$ pay-to-Taproot (P2TR) outputs, SIGHASH_DEFAULT, and no annex, vsize is $68 + 43n$.

Baseline. We compare bilateral controls with independent-pair models: one native DLC CET (193 vbytes, two signatures) or chosen Vwe2psTS P2TR spend (154 vbytes, one signature) per obligation. Vwe2psTS is an encoding model without author-wallet or regtest measurements. Both models exclude batching and joint netting.

Results. At $n = 128$, MuCPay uses 5,572 vbytes, one signature, and a 11,144 sat fee. Separate and shared-CET settlement use 1,251,712 and 491,475 vbytes, respectively, each with 8,128 signatures. Size and fees fall by 99.6% and 98.9% at the common fee rate. Shared transactions still require separate inputs for each payment obligation, whereas MuCPay uses joint authorization to settle all agreed payments through a single spend. Figure 6 contrasts linear payment growth with quadratic independent obligations. MuCPay needs one transaction and $\Theta(1)$ signature checks, with $\Theta(n)$ visible outputs.

At $n = 128$, independent DLC uses 1,568,704 vbytes across p CETs, which is 281.5 times MuCPay's footprint. The chosen Vwe2psTS encoding uses 1,251,712 vbytes and requires 8,128 signature checks. Both models settle each obligation independently, while shared-CET measurements show that MuCPay retains its advantage even when bilateral transactions are batched. Recipient payments retain the linear transaction-size term.

6 Conclusion

In this work, we address the scalability bottleneck of scriptless multi-party oracle-based conditional payments caused by the quadratic overhead of all-pairs composition. We design MuCPay, a scalable protocol powered by HB-AMS that achieves linear off-chain signing-share generation per outcome and a constant-size on-chain settlement witness. Its key design efficiently aggregates multi-party conditions into compact joint pre-signatures and produces BIP340-compatible adapted signatures for scriptless Taproot key-path settlement on Bitcoin. We formally prove CUF against unauthorized signatures and ES against settlement of unreleased outcomes, even under concurrent signing and malicious aborts, with up to $n - 1$ statically corrupted participants and a fixed single-event policy. Finally, we implement MuCPay and evaluate it against existing payment baselines, demonstrating improvements in preparation efficiency and settlement cost. Looking forward, we aim to extend MuCPay toward cross-chain oracle-based conditional payments [33] and integration with state channel networks [3, 36, 38].

References

[1] Lukas Aumayr, Zeta Avarikioti, Matteo Maffei, and Subhra Mazumdar. 2024. Securing lightning channels against rational miners. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*. ACM, 393–407. https://repositum.tuwien.at/handle/20.500.12708/209770

[2] Lukas Aumayr, Oguzhan Ersoy, Andreas Erwig, Sebastian Faust, Kristina Hostáková, Matteo Maffei, Pedro Moreno-Sanchez, and Siavash Riahi. 2021. Generalized channels from limited blockchain scripts and adaptor signatures. In *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part II (Lecture Notes in Computer Science)*. Springer, 635–664. doi:10.1007/978-3-030-92075-3_22

[3] Zeta Avarikioti, Yuheng Wang, and Yuyi Wang. 2025. Thunderdome: Timelock-Free Rationally-Secure Virtual Channels. In *34th USENIX Security Symposium, USENIX Security 2025, Seattle, WA, USA, August 13-15, 2025*. USENIX Association, 4187–4204. https://www.usenix.org/conference/usenixsecurity25/presentation/avarikioti

[4] Renas Bacho and Yanbo Chen. 2026. Earpicks: Tightly Secure Two-Round Multi and Threshold Signatures. In *Advances in Cryptology – EUROCRYPT 2026*. Springer, 154–183. doi:10.1007/978-3-032-25291-3_6

[5] Renas Bacho and Benedikt Wagner. 2025. T-Spoon: Tightly Secure Two-Round Multi-signatures with Key Aggregation. In *Advances in Cryptology – CRYPTO 2025*. Springer, 256–290. doi:10.1007/978-3-032-01887-8_9

[6] Ruben Baecker, Paul Gerhart, Jonathan Katz, and Dominique Schröder. 2026. Fair Distributed Exchange via Threshold Adaptor Signatures. Cryptology ePrint Archive, Paper 2025/388. ASIACRYPT 2026; revision of September 14, 2026. https://eprint.iacr.org/2025/388

[7] Mihir Bellare and Wei Dai. 2021. Chain Reductions for Multi-Signatures and the HBMS Scheme. Cryptology ePrint Archive, Paper 2021/404. https://eprint.iacr.org/2021/404

[8] Bitcoin Core developers. 2025. Bitcoin Core 29.0. https://bitcoincore.org/en/releases/29.0/

[9] Bitcoin Core developers. 2026. libsecp256k1. Version 0.7.1. https://github.com/bitcoin-core/secp256k1/tree/v0.7.1

[10] Dan Boneh and Victor Shoup. 2023. *A Graduate Course in Applied Cryptography*. Online manuscript. Version 0.6. https://toc.cryptobook.us/book.pdf

[11] Lorenz Breidenbach, Christian Cachin, Benedict Chan, Alex Coventry, Steve Ellis, Ari Juels, Farinaz Koushanfar, Andrew Miller, Brendan Magauran, Daniel Moroz, et al. 2021. Chainlink 2.0: Next Steps in the Evolution of Decentralized Oracle Networks. Chainlink Labs white paper. https://research.chain.link/whitepaper-v2.pdf.

[12] Conduition. 2024. Off-Chain Ticketing for Discreet Log Contracts. Retrieved October 22, 2026 from https://conduition.io/scriptless/ticketed-dlc/

[13] conduition. 2026. musig2: A Rust Implementation of MuSig2 with Adaptor Signatures. Source revision recorded in the evaluation artifact. https://github.com/conduition/musig2

[14] DLC community. 2026. Specification for Discreet Log Contracts. https://github.com/discreetlogcontracts/dlcspecs

[15] Manu Drijvers, Kasra Edalatnejad, Bryan Ford, Eike Kiltz, Julian Loss, Gregory Neven, and Igors Stepanovs. 2019. On the security of two-round multi-signatures. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*. IEEE, 1084–1101. doi:10.1109/SP.2019.00050

[16] Thaddeus Dryja. 2017. Discreet log contracts. MIT Digital Currency Initiative. https://adiabat.github.io/dlc.pdf.

[17] Lloyd Fournier. 2019. One-time verifiably encrypted signatures aka adaptor signatures. Technical report. https://github.com/LLFourn/one-time-VES/blob/master/main.pdf.

[18] Lloyd Fournier. 2023. DLC Verifiable Encryption with BLS Attestations. Author benchmark implementation, ndss-snapshot. https://github.com/LLFourn/dlc-verifiable-encryption-bls/tree/ndss-snapshot

[19] Tommaso Frassetto, Patrick Jauernig, David Koisser, David Kretzler, Benjamin Schlosser, Sebastian Faust, and Ahmad-Reza Sadeghi. 2023. 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*. Internet Society. https://www.ndss-symposium.org/ndss-paper/pose-practical-off-chain-smart-contract-execution/

[20] Paul Gerhart, Dominique Schröder, Pratik Soni, and Sri Aravinda Krishnan Thyagarajan. 2024. Foundations of Adaptor Signatures. In *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part II (Lecture Notes in Computer Science)*. Springer, 161–189. doi:10.1007/978-3-031-58723-8_6

[21] Panpan Han, Zheng Yan, Xueqin Liang, and Laurence T Yang. 2025. Vwe2psTS: Supporting Oracle-Based Conditional Payments from Joint Addresses. In *2025 IEEE International Conference on Blockchain (Blockchain)*. IEEE, 93–100. doi:10.1109/BLOCKCHAIN67634.2025.00022

[22] Yunfeng Ji, Yuting Xiao, Birou Gao, and Rui Zhang. 2024. Threshold/Multi Adaptor Signature and Their Applications in Blockchains. *Electronics* 13, 1 (2024), 76. doi:10.3390/electronics13010076

[23] Xuan Thanh Khuc, Willy Susilo, Dung Hoang Duong, Yannan Li, Partha Sarathi Roy, Kazuhide Fukushima, and Shinsaku Kiyomoto. 2026. Adaptor Multi-Signatures: Definition, Applications and Construction from Cryptographic Group Actions. IEEE EuroS&P. doi:10.1109/EuroSP68448.2026.00015

[24] Thibaut Le Guilly, Nadav Kohen, and Ichiro Kuwahara. 2022. Bitcoin oracle contracts: Discreet log contracts in practice. In *IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2022, Shanghai, China, May 2-5, 2022*. IEEE, 1–8. doi:10.1109/ICBC54727.2022.9805512

[25] Xiangyu Liu, Ioannis Tzannetos, and Vassilis Zikas. 2024. Adaptor Signatures: New Security Definition and a Generic Construction for NP Relations. In *Advances in Cryptology - Asiacrypt 2024, Pt Ii*, Vol. 15485. Springer-Verlag Singapore Pte Ltd, Singapore, 168–193. https://eprint.iacr.org/2024/1051

[26] Varun Madathil, Sri Aravinda Krishnan Thyagarajan, Dimitrios Vasilopoulos, Lloyd Fournier, Giulio Malavolta, and Pedro Moreno-Sanchez. 2023. Cryptographic Oracle-Based Conditional Payments. In *Network and Distributed System Security Symposium (NDSS)*. https://www.ndss-symposium.org/wp-content/uploads/2023-24-paper.pdf

[27] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. 2019. Simple Schnorr Multi-Signatures with Applications to Bitcoin. *Designs, Codes and Cryptography* (2019). https://eprint.iacr.org/2018/068

[28] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. White paper. https://bitcoin.org/bitcoin.pdf.

[29] Sela Navot and Stefano Tessaro. 2024. One-More Unforgeability for Multi- and Threshold Signatures. Cryptology ePrint Archive, Paper 2024/1947. https://eprint.iacr.org/2024/1947

[30] Jonas Nick, Tim Ruffing, and Elliott Jin. 2022. BIP327: MuSig2 for BIP340-compatible Multi-Signatures. https://github.com/bitcoin/bips/blob/master/bip-0327.mediawiki

[31] Jonas Nick, Tim Ruffing, and Yannick Seurin. 2021. MuSig2: Simple Two-Round Schnorr Multi-Signatures. In *Advances in Cryptology – CRYPTO 2021*. https://eprint.iacr.org/2020/1261

[32] p2pderivatives contributors. 2026. rust-dlc: Discreet Log Contract Libraries. dlc crate 0.8.0; revision and locked dependencies recorded in artifact. https://github.com/p2pderivatives/rust-dlc

[33] Giulia Scaffino, Lukas Aumayr, Mahsa Bastankhah, Zeta Avarikioti, and Matteo Maffei. 2025. Alba: The Dawn of Scalable Bridges for Blockchains. In *Proceedings 2025 Network and Distributed System Security Symposium*. Internet Society, San Diego, CA, USA. https://www.ndss-symposium.org/ndss-paper/alba-the-dawn-of-scalable-bridges-for-blockchains/

[34] Victor Shoup. 2006. Sequences of Games: A Tool for Taming Complexity in Security Proofs. Revised January 18, 2006; first version 2004. https://www.shoup.net/papers/games.pdf

[35] Sri Aravinda Krishnan Thyagarajan, Giulio Malavolta, and Pedro Moreno-Sanchez. 2022. Universal Atomic Swaps: Secure Exchange of Coins Across All Blockchains. In *IEEE Symposium on Security and Privacy*. https://eprint.iacr.org/2021/1612

[36] Anqi Tian, Peifang Ni, Yingzi Gao, and Jing Xu. 2025. Horcrux: Synthesize, Split, Shift and Stay Alive; Preventing Channel Depletion via Universal and Enhanced Multi-hop Payments. In *Proceedings 2025 Network and Distributed System Security Symposium*. Internet Society, San Diego, CA, USA. https://www.ndss-symposium.org/ndss-paper/horcrux-synthesize-split-shift-and-stay-alive-preventing-channel-depletion-via-universal-and-enhanced-multi-hop-payments/

[37] Sam Werner, Daniel Perez, Lewis Gudgeon, Ariah Klages-Mundt, Dominik Harz, and William Knottenbelt. 2022. SoK: Decentralized Finance (DeFi). In *Proceedings of the 4th ACM Conference on Advances in Financial Technologies, AFT 2022, Cambridge, MA, USA, September 19-21, 2022*. ACM, 30–46. doi:10.1145/3558535.3559780

[38] Di Wu, Shoupeng Ren, Yuman Bai, Lipeng He, Jian Liu, Wu Wen, Kui Ren, and Chun Chen. 2025. On the atomicity and efficiency of blockchain payment channels. In *34th USENIX Security Symposium (USENIX Security 25)*. USENIX Association, 4053–4072. https://www.usenix.org/conference/usenixsecurity25/presentation/wu-di

[39] Pieter Wuille, Jonas Nick, and Tim Ruffing. 2020. BIP340: Schnorr Signatures for secp256k1. https://bips.dev/340/

[40] Pieter Wuille, Jonas Nick, and Anthony Towns. 2020. BIP341: Taproot: SegWit version 1 spending rules. https://bips.dev/341/

[41] Danlei Xiao, Chuan Zhang, Haotian Deng, Jinwen Liang, Licheng Wang, and Liehuang Zhu. 2025. Parallelizing Universal Atomic Swaps for Multi-Chain Cryptocurrency Exchanges. In *34th USENIX Security Symposium*. USENIX Association, 4073–4092. https://www.usenix.org/conference/usenixsecurity25/presentation/xiao-danlei

[42] Danlei Xiao, Chuan Zhang, Haotian Deng, Jinwen Liang, Licheng Wang, and Liehuang Zhu. 2025. Parallelizing Universal Atomic Swaps for Multi-Chain Cryptocurrency Exchanges. Software artifact (ParaSwap); source archive SHA256

retained in evaluation artifact. doi:10.5281/zenodo.15594022

A Security Analysis of HB-AMS

A.1 Basic Properties and Bitcoin Compatibility

Proof of Proposition 1. (i) Summing honest shares and the tweak gives $\hat{s}G = \delta S + cX' = R' - \delta\beta A + cX'$, hence $\mathsf{PV}_C(\hat{\sigma}) = 1$ on every nonaborting execution. (ii) For any accepted $\hat{\sigma}$, $(\hat{s} + \delta\beta a)G = R' + cX'$, so $\mathsf{V}_m(\mathsf{MAdapt}(C, \hat{\sigma}, a)) = 1$. (iii) Matching nonce coordinates lift to the same even R' and challenge. Therefore $(s - \hat{s})G = \delta\beta A$, and $\delta\beta \neq 0$ yields $\mathsf{MExt}(C, \hat{\sigma}, \sigma) = a$. ORD uses $R' = \delta S$. (iv) In WE_1, outside ϵ_B failures, target registration fixes its unique target-message Prepared tuple: admission excludes ORD on $m^\star$ and any second target Start. Release changes neither this uniqueness nor the registered transcript. By (iii), $W_{\mathsf{WE}_1} = 1$ forces a different final nonce. One target Start and $m^\star \notin \mathcal{F}$ force non-Prepared; Lemma 2 gives $\mathrm{Adv}_{\mathsf{WE}_1}(\mathcal{A}) \le \mathrm{Adv}_{\mathsf{XIDL}}(\mathcal{B}_W) + \epsilon_B$. XIDL-to-DL and Proposition 3 yield (iv) and its binary bound. □

Proposition 2 (BIP340 verification compatibility)**.** In the secp256k1 profile, $\forall C, \hat{\sigma}, a, \sigma := \mathsf{MAdapt}(C, \hat{\sigma}, a) \neq \bot \Rightarrow (|\sigma| = 64 \wedge \mathsf{MVerify}(x(X'), m, \sigma) = 1)$ under unchanged BIP340.

Proof. Canonical 32-byte encodings, nonidentity even X', R', and $\tilde{s}G - cX' = R'$ meet BIP340 verification [39, Verification]. BIP341 SIGHASH_DEFAULT key-path signatures use 64 bytes without a suffix [40, Taproot key path spending signature validation]; transaction validity is separate. □

A.2 Concurrent Security Experiments

Parameters and algebraic interface. Use the group and independent scalar oracles of Section 3, commitment/salt lengths $h_{\mathrm{com}}, \lambda$, and statically honest h. For $\mathcal{H} \neq \emptyset$, independently generate keys and simulate the signing states of $\mathcal{H} \setminus \{h\}$ locally, $h \in \mathcal{H}$. Polynomial B bounds external/internal hashes, point/commitment entries, targets, calls, candidate lists, and outcomes; $1 \le Q_o \le B$ bounds well-formed distinct H_o inputs with prefix (U, X_o), including setup/validation. Unused guesses fail. Final list/output are fixed; earlier queries may use other lists.

Append delivered elements to basis G. For basis $(G, Q_1, \cdots, Q_d)$, coefficients $\rho_P \in \mathbb{Z}_q^{d+1}$ certify $P = \rho_0 G + \sum_i \rho_i Q_i$. Cache first-submission coefficients in prior received elements for every adversarial point, including parsed raw/logical $H_{\mathrm{agg}}, H_{\mathrm{com}}, H_{\mathrm{ad}}, H_o$ query points. Use normalized x-only lifts. Annotations are private, not protocol proofs of knowledge; exact-byte caches never overwrite; malformed inputs get independent ordinary answers.

Setup and admission. Fix h before key generation; sample $(P_h, x_h) \leftarrow \mathsf{MKGen}(pp)$ and independent nonzero x_o, r_o. Given P_h, X_o, U and hash access, $\mathcal{A}$ chooses other keys and Π, placing P_h at h. Reject invalid keys/points, duplicate public keys in $\mathcal{L}$, zero adaptors, repeated challenges/digests, or $\mathcal{D} \cap \mathcal{F} \neq \emptyset$. Define $Q_\Pi = \{(\mathsf{PRE}, m_j, A_j, (E, M_j)) : j \in [N]\} \cup \{(\mathsf{ORD}, m, \bot, \bot) : m \in \mathcal{F}\}$ and $\mathrm{Admit}_\Pi(C) := [(\mathcal{L}_C, X'_C) = (\mathcal{L}, X')] \wedge [(\mathsf{mode}, m, A, \mathsf{app}) \in Q_\Pi]$ for well-formed C; otherwise zero.

Policy and atomic sessions. Freeze Section 2.1's policy before Start; enforce it before and after Release. The manifest may follow arbitrary queries and (U, X_o) publication; validation shares the H_o cache. The whitelist $\mathcal{F}$ contains the refund digest. Manifest digests forbid ORD/alternative adaptors even with a witness; hashes/CUF outputs remain unrestricted.

Initialize Used = Auth = AuthOrd = Released = $\emptyset$ and Sess = $\emptyset$. Use Section 3's contexts/checks/nonce rules, immutable records, and permanent used-*sid* storage:

(1) Start(C) rejects inadmissible contexts/reused identifiers; else reserve *sid*, store the nonce, salt, and commitment in COMMITTED, add m to Auth (also AuthOrd for ORD), and return D_h.
(2) Reveal(*sid*, **D**) requires COMMITTED and a complete signer-indexed vector containing D_h; atomically freeze, monitor, enter REVEALED, and return (R_h, η_h) without malicious openings/aggregate validity.
(3) Respond requires REVEALED; check encodings, frozen openings, and aggregates; durably enter RESPONDED and erase r_h before the one response. Retain public data only.
(4) Failed Reveal/Respond or Abort erases the nonce and enters ABORTED. Nonexistent/terminal sessions return $\bot$ unchanged; rejected Start authorizes nothing. Retry with fresh *sid*/randomness; crashes cannot restore consumed nonces.

Calls interleave with delayed delivery. Crash() aborts active sessions, retaining used identifiers, authorizations, policy, public preparation, and the private monitor's records. Real/simulated games share the interface.

Prepared monitor and wins. Before returning h's opening, resolve frozen malicious digests to unique prior H_{com} preimages $(C, i, \mathrm{enc}(R_i), \eta_i)$ matching context/identity and canonical nonidentity points. If all resolve and the aggregate is nondegenerate, record $T = (x(X'), m, x(R')) \in$ Prepared with openings/coefficients/mode. PRE shares H_{ad}; ORD uses $R' = \delta S$. Otherwise Reveal without recording; invalid Respond openings reject, later valid unqueried openings imply BadCom. Monitoring changes no messages/authorizations/nonce states. Records survive aborts without delivered malicious openings/shares or completed pre-signatures; duplicates suffice outside tuple collisions. Message disjointness separates ORD from ES/PRE tuples.

Use Definitions 1–2 and BIP340 verification. Only the first valid Release(k) atomically returns a_k and sets Released = $\{k\}$; others return $\bot$. Let $E_{\mathrm{pre}}, E_{\mathrm{post}}$ denote ES wins before/after release and PrepHit exact final-tuple membership in Prepared.

Definition 4 (Single-target WE experiment)**.** Use Appendix A.2's setup and interfaces. Fix $j^\star \in [N]$ before Start and $m^\star = m_{j^\star}$; admit at most one Start on $m^\star$. Other messages remain concurrent; Release may select $j^\star$. Split the post-setup attack into registration and final-output stages $\mathcal{A}_{\mathrm{reg}}, \mathcal{A}_{\mathrm{fin}}$, $\mathrm{Adv}_{\mathsf{WE}_1}(\mathcal{A}; \lambda) :=$

$$\Pr\left[\begin{array}{l}(C^\star, \hat{\sigma}^\star, \mathsf{tr}^\star, \mathsf{st}) \leftarrow \mathcal{A}_{\mathrm{reg}}^{O_\Pi}(\Pi, j^\star);\\ \mathsf{TargetOK} \leftarrow [C^\star = \mathsf{Sess}[sid^\star].C] \wedge [m_{C^\star} = m^\star]\\ \quad \wedge [\mathsf{state}(sid^\star) = \mathsf{RESPONDED}]\\ \quad \wedge [\mathsf{CheckTranscript}(\mathsf{tr}^\star) = 1]\\ \wedge [\hat{\sigma}^\star = \mathsf{Aggregate}(\mathsf{tr}^\star)] \wedge \mathsf{PV}_{C^\star}(\hat{\sigma}^\star); \sigma^\star \leftarrow \mathcal{A}_{\mathrm{fin}}^{O_\Pi}(\mathsf{st});\\ W_{\mathsf{WE}_1} \leftarrow \mathsf{TargetOK} \wedge \mathsf{V}_{m^\star}(\sigma^\star)\\ \quad \wedge [\mathsf{MExt}(C^\star, \hat{\sigma}^\star, \sigma^\star) = \bot] :\\ \qquad W_{\mathsf{WE}_1} = 1\end{array}\right].$$

Here $sid^\star$ is the session identifier in $C^\star$. CheckTranscript checks the complete immutable context, commitments, openings, and all

shares, including the honest response. Count public registration checks in B.

A.3 Quantitative Reduction Bound

Theorem 3 (Concurrent security bound)**.** In Appendix A.2's experiment, reductions exist with

$$\mathrm{Adv}^{\mathsf{CUF}}(\mathcal{A}) \le \mathrm{Adv}^{\mathsf{XIDL}}(\mathcal{B}_X^{\mathsf{C}}) + \epsilon_B, \tag{6}$$

$$\mathrm{Adv}^{\mathsf{ES}}(\mathcal{A}) \le \mathrm{Adv}^{\mathsf{XIDL}}(\mathcal{B}_X^{\mathsf{E}}) + \mathrm{Adv}^{\mathsf{DL}}(\mathcal{B}_{\mathrm{pre}}) + Q_o\,\mathrm{Adv}^{\mathsf{DL}}(\mathcal{B}_{\mathrm{post}}) + (2+Q_o)\epsilon_B, \tag{7}$$

where $\epsilon_B = c_*(B^2/q + B^2/2^{h_{\mathrm{com}}} + B^2/2^{\lambda})$, c_* sufficiently large and absolute, bounds simulation/extraction errors before guessing loss, independently of success. Prepared extraction uses the algebraic interface.

A.4 Failure Events and Bounds

Proof of Theorem 3. Couple each simulation until its first failure [34, Lemma 1]. For delayed honest openings and fixed prior history, union-bound B draws against B prior entries without conditioning on final success/selection:

(1) BadCom: distinct complete-input digest collisions, honest-opening guesses before Reveal, programming queried inputs, or first valid openings of frozen malicious digests after Reveal (including Respond). Validation queries/preprogramming cost $O(B^2/2^{h_{\mathrm{com}}} + B^2/2^{\lambda})$; salts remain independent even given nonce points.

(2) BadAd: a resolved PRE base $(C, \mathrm{enc}(S))$ is evaluated before commitment freeze/honest-opening determination, including its first raw counter query. Fresh nonzero R_h hits each prior $S_0 - \sum_{i\neq h} R_i$ with probability at most $1/(q-1)$: total $O(B^2/q)$.

(3) BadPrequery: a resolved nondegenerate tuple is cached before its first monitor/simulator determination, including external/internal LOCAL/XIDL/ORDINARY entries. Outside BadAd, PRE's injective $\beta \mapsto S + \beta A$ has at most two preimages per x-coordinate; ORD has fresh R_h. Exact-input prequeries/session collisions cost $O(B^2/q)$; PRE/ORD messages are disjoint.

(4) BadKeyReg: output conflict with a registered/challenged key or normalized internal-key hit on a prior tweak input. Given list/malicious coefficients, fresh t_h makes $V + t_h X$ uniform up to zero; normalization is two-to-one. Prior hits cost $O(B/q)$ each; else the binary tweak is fresh uniform given range acceptance. Total: $O(B^2/q)$.

(5) BadOracle: prepublication current-prefix queries or equal challenges at distinct current-prefix inputs (including c_0), costing $O(B^2/q)$ by uniform embedded U and fresh answers. Accepted manifests have distinct challenges. BadZero: zero embedded setup points, XIDL key/target coefficients, or simulated one-draw nonces, costing $O(B/q)$; setup rejects fresh zero adaptors.

(6) BadDen: zero extraction denominator in a matching unreleased PRE record (probability at most $1/(q-1)$ or $1/q$ per draw, as shown below). BadUnqueried: an unevaluated final challenge verifies, with probability at most $1/q$ since $X' \neq O$. Union bounds cost $O(B^2/q)$.

Count internal monitor/challenge evaluations in B; ordinary rejections follow real checks without extra loss.

A.5 Non-Prepared Signatures: XIDL Reduction

Imported game and registration. Additive XIDL [29, Lemma 4] gives $X = xG$, $x \in \mathbb{Z}_q$ uniform. NewTarget(V) registers $K = V + tX$ with fresh uniform t and index J; Challenge(J, W) returns fresh uniform c and index I. Output (I, z) wins iff $zG = W + cK$; V, W precede t, c, indices count calls, and no extra oracle is available. For $1 \le b_1, b_2 \le B$ targets/challenges, the DL conversion [7, Theorems 3.2, 3.4] is $\mathrm{Adv}^{\mathsf{XIDL}}(\mathcal{B}_X) \le \left[b_2\left(\sqrt{b_1\,\mathrm{Adv}^{\mathsf{DL}}(\mathcal{D}_X)} + b_1/q\right)\right]^{1/2} + b_2/q$. Zero X, t_h aborts cost $O(B/q)$; other weights are nonzero.

Set $P_h = X$. On the first canonical member query for a valid list of distinct public keys containing X, sample $t_{i\neq h}$, set $V = \sum_{i\neq h} t_i P_i$, obtain $(J_{\mathcal{L}}, t_h)$ from NewTarget, and cache the normalized/tweaked output $x(X'_{\mathcal{L}}) \mapsto (J_{\mathcal{L}}, \gamma_{\mathcal{L}}, \tau_{\mathrm{eff},\mathcal{L}})$. Invalid outputs/tweaks invalidate the list but retain coefficients; prior conflicts imply BadKeyReg. Lists with invalid, identity, or duplicate public keys, lists without X, nonmember queries, and malformed inputs receive ordinary cached answers. Canonical encodings exclude precomputed-list aliases too.

Index exact BIP340 bytes $x(R')\|x(X')\|m$ by T, tagged LOCAL/XIDL/ORDINARY. For fresh valid registered inputs, lift even R' and cache (c, XIDL, I) from Challenge$(J_{\mathcal{L}}, \gamma_{\mathcal{L}} R')$. Unregistered/malformed/unliftable inputs get ORDINARY answers; later registration conflicts imply BadKeyReg. LOCAL never calls Challenge.

Concurrent simulation. Generate the prescribed oracle with known Release scalars. Start returns uniform D_h, deferring its opening; Reveal freezes/resolves malicious commitments. For missing/invalid preimages, sample fresh nonzero R_h and a fresh salt, program to D_h, and Reveal without Prepared/LOCAL. Missing preimages cannot pass Respond outside BadCom; invalid openings reject and ambiguity implies BadCom.

Otherwise sample independent $u, v \leftarrow \mathbb{Z}_q$, set $B_h = \gamma t_h X$, $R_h = uG - vB_h$. Abort at zero $(1/q)$; else the nonce is uniform nonzero. Program its fresh-salt commitment (conflicts: BadCom). Derive S, R'; degenerate aggregates Reveal but fail Respond. For nondegenerate T, bypass the generic challenge handler: atomically check T fresh, install $(\delta v, \mathsf{LOCAL}, sid)$, save $\hat{s}_h = \delta u$, and record Prepared before Reveal, ensuring $\mathsf{LOCAL}(T) \Rightarrow T \in$ Prepared. Cached T implies BadPrequery without overwriting. Respond applies real checks and consumes/returns the share once.

Outside BadCom, opening delay is valid: the Schnorr simulator [10, Theorem 19.4] makes $r_h = u - v\gamma t_h x$ uniform independently of v, also given $r_h \neq 0$. Since δ depends only on the nonce transcript, $c = \delta v$ is uniform and shares have the real distribution. One-draw sampling/atomic programming/caching ensure strict polynomial time for arbitrary interleavings, without rewinding or malicious nonce logarithms.

Extraction. Outside failures, every valid non-Prepared final tuple is XIDL (an unevaluated final challenge costs $1/q$; final verification may register it). Output $z = \gamma(\tilde{s} - c\tau_{\mathrm{eff}})$ at index I: $X' = \gamma K + \tau_{\mathrm{eff}} G$ implies $zG = \gamma R' + cK$, with $W = \gamma R'$ fixed before c.

A.6 Prepared ES: Oracle-DL Reductions

Both reductions know h's key, sign honestly, and embed $Y = yG$. Solve $Y = O$ directly; nonzero-secret conditioning and zero-U rejection cost $O(1/q)$ within ϵ_B.

Pre-release embedding. Sample $x_o \neq 0$, set $X_o = x_o G$ and $U = Y$, answer H_o normally, and stop at valid Release. Every $A_j = c_j X_o + Y$, without guessing queries or branches.

Post-release handler and embedding. Before publishing U, sample hidden $J^* \leftarrow [1, Q_o]$, $c_0, a_0 \leftarrow \mathbb{Z}_q$; set $X_o = Y$, $U = a_0 G - c_0 Y$. Answer distinct well-formed current-prefix H_o input number J^*, $(\mathrm{enc}(U), \mathrm{enc}(X_o), E, M)$, stored as t^*, with c_0; others freshly. Shared-cache counting includes external/setup/validation queries once per input. Unused guesses fail; prepublication conflicts imply BadOracle without overwriting.

Return a_0 on valid Release(k) iff its full input is t^*; otherwise stop. On a match, $c_k = c_0$, $A_k = a_0 G$, and $A_j = a_0 G + (c_j - c_0) Y$ for all $j \neq k$. Setup ensures $c_j - c_0 \neq 0$; trial collisions imply BadOracle. Guess only the released input.

Lemma 1 (Adaptive first-query index)**.** The unconditioned post-release shadow view $\mathcal{V}_{\mathrm{sh}}$ includes transcript/shared oracle tables/original first-submission annotations/Prepared tuples, excluding hidden index/helper labels and (G, Y) expansions, retaining actual hash answers/releases. Then $J^* \perp \mathcal{V}_{\mathrm{sh}}$ and $\Pr[\mathrm{Succ} \wedge J^* = I_o] = \Pr[\mathrm{Succ}]/Q_o$ for $\mathcal{V}_{\mathrm{sh}}$-measurable Succ with existing first-query index I_o. Matching released-input indices couple reduction/shadow until charged failures.

Proof. Run the real experiment with independent unused J^* and genuine Release, continuing on mismatches. Reparameterize $a_0 = r_o + c_0 y$: $U = r_o G = a_0 G - c_0 Y$. For fixed J^*, designated c_0 is fresh uniform independently of r_o, also given $r_o \neq 0$; other answers are fresh. Lazy sampling preserves $\mathcal{V}_{\mathrm{sh}}$, including adaptive annotations/internal queries, until prepublication conflicts [34, Lemma 1]. J^* stays uniform with fewer than Q_o queries. When Release matches the designated query, both runs return a_0. □

Extraction and denominator. Expand first-submission annotations in (G, Y), retaining them after nonce erasure. For pre-release or unreleased post-release branches, $A_j = \alpha_A G + \ell_A Y$ has coefficients $(\alpha_A, \ell_A) = (c_j x_o, 1)$ or $(a_0, c_j - c_0)$, respectively, with $\ell_A \neq 0$. Alternative $\bar{\ell}_A \neq \ell_A$ gives $y = (\bar{\alpha}_A - \alpha_A)/(\ell_A - \bar{\ell}_A)$, including $\bar{\ell}_A = 0$; equal Y coefficients force equal G coefficients.

A matching Prepared record fixes the malicious nonce sum $\alpha_a G + \ell_a Y$ before the honest opening $r_h G$. For the analysis, express $S = (r_h + \alpha_a) G + \ell_a Y$, $X' = \alpha_X G + \ell_X Y$, $R' = \alpha_R G + \ell_R Y$, where $\alpha_R = \delta(r_h + \alpha_a + \beta\alpha_A)$ and $\ell_R = \delta(\ell_a + \beta\ell_A)$. Verification gives $\tilde{s} G = (\alpha_R + c\alpha_X) G + dY$, $d = \ell_R + c\ell_X$. For $d \neq 0$, extract $y = (\tilde{s} - \alpha_R - c\alpha_X) d^{-1}$ using retained coefficients, prior preimages, and final verification, independently of malicious delivery, use of h's opening, or aggregation.

Commitments fix ℓ_a, ℓ_A before fresh β; the list fixes ℓ_X. For $\ell_X = 0$, zero d requires $\beta = -\ell_a/\ell_A$: probability at most $1/(q-1)$ (zero for $\ell_a = 0$); normalization preserves zero. For $\ell_X \neq 0$, fixed S, β, R' leave fresh uniform c outside prequeries/collisions, with one root. Union-bound B unreleased PRE records/queries without final-selection conditioning; exclude ORD/released records. Discrepancies yielding y are successes.

Direct extraction before release and Lemma 1 after release, with a Q_o guessing loss, yield

$$\Pr[E_{\mathrm{pre}} \wedge \mathrm{PrepHit}] \leq \mathrm{Adv}_{\mathrm{DL}}(\mathcal{B}_{\mathrm{pre}}) + \epsilon_B,$$
$$\Pr[E_{\mathrm{post}} \wedge \mathrm{PrepHit}] \leq Q_o \, \mathrm{Adv}_{\mathrm{DL}}(\mathcal{B}_{\mathrm{post}}) + Q_o \epsilon_B.$$

A.7 Game Composition

Prepared messages lie in Auth, hence $W_{\mathsf{CUF}} = 1 \Rightarrow \neg\mathrm{PrepHit}$; Appendix A.5 gives Equation (6). Since $\Pr[W_{\mathsf{ES}} = 1] = \Pr[W_{\mathsf{ES}} = 1 \wedge \neg\mathrm{PrepHit}] + \Pr[E_{\mathrm{pre}} \wedge \mathrm{PrepHit}] + \Pr[E_{\mathrm{post}} \wedge \mathrm{PrepHit}]$, Appendices A.5–A.6 give Equation (7) with error $(2 + Q_o)\epsilon_B$. □

Lemma 2 (Non-Prepared signature bound)**.** In Section 2.2's model, let NP denote a valid signature under X' with $(x(X'), m, x(R')) \notin$ Prepared. Some $\mathcal{B}_N$ satisfies $\Pr[\mathrm{NP}] \leq \mathrm{Adv}_{\mathsf{XIDL}}(\mathcal{B}_N) + \epsilon_B$, including authorized/released messages. Public transcript verification without signing responses counts in B.

Proof. Appendix A.5's simulator knows oracle secrets and answers Release. Non-Prepared signatures solve XIDL via $z = \gamma(\tilde{s} - c\tau_{\mathrm{eff}})$ outside failures, regardless of authorization/release. Public verification shares caches/budget, preserving ϵ_B. □

A.8 Binary Profile and Statistical Transfer

Tagged Secure Hash Algorithm 256 (SHA-256) uses prefix MuCPay/HB-AMS/, suffixes keyagg/v1, adaptor/v1, oracle/v1, commit/v1; retain BIP0340/challenge and TapTweak. Both games use injective, length-delimited, uniquely parseable tuples and canonical compressed points. Contexts are (mode, sid, $\mathcal{L}$, X', m, A, app) with prescribed app and 256-bit commitments/salts.

For $H_{\mathrm{agg}}, H_{\mathrm{ad}}$, hash Encode$(D, j)$ with tuple base D and canonical unsigned counters $j = 0, \cdots, L-1$, accepting the first integer in $[1, q-1]$. Reject $\{0\} \cup [q, 2^{256} - 1]$. At cap $L \geq 1$ (256 for Bitcoin), return $\perp$ and abort before responding. Reduce H_o and BIP340 challenges modulo q, while TapTweak follows BIP341 range rejection without reduction.

Proposition 3 (Transfer to independent binary random oracles)**.** Use independent 256-bit tagged domains with logical/raw access. The bound B counts raw and virtual queries and sampler attempts, while Q_o counts first evaluations of distinct outcome inputs. For $q < 2^{256}$, set $d_q = 2^{256} \bmod q$, $\Delta_q = d_q(q - d_q)/(2^{256} q)$, $p_{\mathrm{acc}} = (q-1)/2^{256}$. Each $Z \in \{\mathsf{CUF}, \mathsf{ES}, \mathsf{WE}_1\}$ and binary-profile $\mathcal{A}$ has scalar-profile $\mathcal{A}'$ satisfying $\mathrm{Adv}^{\mathrm{bin}}_Z(\mathcal{A}) \leq \mathrm{Adv}^{\mathrm{scalar}}_Z(\mathcal{A}') + B\Delta_q + B(1 - p_{\mathrm{acc}})^L$. For $k(\lambda)$-bit profiles, $q < 2^{k(\lambda)}$, replace 256 by $k(\lambda)$; asymptotic transfer requires $B\Delta_q + B(1-p_{\mathrm{acc}})^L \in \mathrm{negl}$. For secp256k1, $2^{256} - q + 1 < 2^{129}$ gives $\Delta_q < 2^{-127}$ and $1 - p_{\mathrm{acc}} < 2^{-127}$; at $L = 256$, the added error is less than $B(2^{-127} + 2^{-32512})$.

Proof. On each base's first raw/logical query, sample $a \leftarrow \mathbb{Z}_q^*$ and independent zero-based geometric k, capped at L. Cache uniform rejections before k, encoded a at $k < L$, and fresh uniform 256-bit values otherwise. Cap failure is $(1 - p_{\mathrm{acc}})^L$. Streams remain independent under out-of-order/post-success queries in strict polynomial time. Uncapped secp256k1 sampling averages < 2 trials.

The first raw queries for candidate lists or adaptors undergo the same registration and freshness checks as their corresponding logical queries. Malformed encodings receive cached ordinary answers without overwriting. Lift ideal H_o and challenge scalars, including LOCAL values, uniformly among their 256-bit modulo preimages. The preimage counts determine the statistical distance Δ_q and yield a hybrid cost of $B\Delta_q$ [10, Theorem 3.11]. Each distinct outcome input is counted only once, and the commitment and TapTweak interfaces remain unchanged in both games. □