MinMandate: Task-Hiding Payment Authorization for Web Agents

Ge Gao
Harbin Institute of Technology
China

Haining Yu*
yuhaining@hit.edu.cn
Harbin Institute of Technology
China

Zhichao Liu
Harbin Institute of Technology
China

Dongyang Zhan
Harbin Institute of Technology
China

Yuanxiao Zhu
Harbin Institute of Technology
China

Zhongyun Hua
Harbin Institute of Technology
China

ABSTRACT

AI agents increasingly pay for Web services on behalf of their users to complete complex tasks. Such a task needs many payments, and the user usually must approve each one, forcing the agent to stop at every payment. Mechanisms such as the Agent Payments Protocol (AP2) therefore enable the user to sign one payment mandate in advance that bounds the agent's payments for the task. However, whoever checks the mandate can learn the task, which is often sensitive. Specifically, (1) the user's wallet can see the full mandate when it checks each payment, so it can trace every payment to the task; and (2) although each merchant sees only the payment it receives, it verifies that payment under a signing key that repeats across the task, so colluding merchants can link the payments and reconstruct the task. We present MinMandate, a task-hiding payment authorization system for AI agents in which the user approves a task-bounded payment mandate once. In MinMandate, the wallet checks each payment through a constant-size keyed-verification policy credential, which reveals neither what the mandate allows nor which mandate the payment comes from. Each merchant then accepts the wallet's one-time ticket as proof of payment and never sees the mandate or its signing key. Formally, we prove authorization soundness and unlinkability in our threat model. Our experiments show that the overhead of MinMandate is practical, with a payment taking 197 ms and under half of AP2's bytes. Notably, MinMandate cuts wallet-side tracing from every payment under AP2 to the level of random guessing. Moreover, from their own payment-request records alone, colluding merchants can reconstruct the user's task in 47% of the cases under AP2 but in only 8% under MinMandate.

CCS CONCEPTS

• **Security and privacy** → **Privacy-preserving protocols**; **Authorization**; *Pseudonymity, anonymity and untraceability*; • **Applied computing** → *Secure online transactions.*

KEYWORDS

AI agents, delegated payment authorization, anonymous credentials, unlinkability

1 INTRODUCTION

In agentic commerce, AI agents increasingly complete user tasks automatically by calling paid Web services, such as the company-data platform Clay [9], through the Model Context Protocol (MCP). A typical paid call takes four steps [32, 46]: the agent sends a service request, the merchant quotes a price, the payment is approved, and the user's wallet pays the merchant. When facing a complex task, the agent usually breaks it down and pays to call several services. For example, to screen a confidential list of acquisition targets for its user, an agent pays web-search merchants for news about each company and data merchants for its financial records. However, if the user had to approve every payment, the agent would stall at each call until the user responded. Per-payment approval thus slows the task and defeats the purpose of delegating it to an agent.

Existing schemes such as the Agent Payments Protocol (AP2) [17] and Stripe Shared Payment Tokens (SPT) [39] have been proposed specifically for agent payments. Instead of approving each payment, they enable the user to set a mandate in advance that bounds the agent's payments and describes what the task may buy. We group them into two classes by who checks the authorized mandate. In *wallet-enforced* designs, such as SPT, the wallet holds the user's funds and the mandate. Before paying a merchant, it checks the payment against the mandate's merchant and amount limits. In *merchant-enforced* designs, such as AP2, the user signs the mandate. For each payment, the agent closes it into a payment-specific mandate that discloses only the merchant being paid. The merchant and the wallet receive and verify this mandate.

However, in both classes, whoever checks the mandate also sees it. As a result, these designs leak the task, which is often sensitive, especially in finance and healthcare. First, both classes allow *wallet-side tracing*: the wallet can see the full mandate when it checks each payment, so it can trace every payment to the task. Second, merchant-enforced designs also allow *merchant-side linking*: each merchant sees only its own payment, but the mandates that all merchants verify are signed under the same key, so colluding merchants can link a task's payments and reconstruct the task from their requests. Such collusion among merchants is realistic, as when third-party trackers on shopping sites linked a user's purchases across sites and traced them to the user's real identity [16]. Both forms of leakage can do real harm to the user. In the screening task, for example, a leaked list of acquisition targets can be exploited for large profits, as when traders who stole merger plans from two law firms made almost $3 million by trading ahead of the deals [42]. Thus, our goal is a mandate that still bounds every payment, while each payment reveals only the call it pays for, not the task it serves.

To achieve this goal, we present **MinMandate**, a task-hiding payment authorization system for AI agents in which the user approves a task-bounded mandate once. Our key observation is that the wallet both issues and verifies the payment credentials, so it can check them with its own secret key, as in keyed-verification

*Corresponding author.

anonymous credentials [7]. The wallet can thus check each payment against the mandate's hidden fine-grained rules without learning which mandate the payment comes from. This removes wallet-side tracing through the mandate. Because the wallet performs this check, merchants never need to see the mandate. Each merchant instead accepts a payment through a one-time wallet ticket bound to its own request and quote. This stops merchant-side linking through a shared mandate. In summary, we make three contributions.

- ◇ We identify two task leaks in delegated payment authorization for AI agents, wallet-side tracing and merchant-side linking. In our experiments with AP2, the wallet traces every payment, and colluding merchants alone reconstruct 41–47% of the tasks.
- ◇ We introduce MinMandate, a payment authorization system that hides the task from both the wallet and the merchants, while the agent still chooses merchants at call time. Its core component is a *keyed-verification policy credential* (KVPC), whose verification cost does not grow with the mandate's rules.
- ◇ We prove the security of KVPC and the authorization soundness and unlinkability of MinMandate. In our experiments, MinMandate cuts wallet-side tracing from every payment under AP2 to the level of random guessing and removes the merchant-side linking that AP2 adds, so merchants reconstruct only 8.3%, close to anonymous tokens, at a practical cost.

2 BACKGROUND AND PROBLEM SETTING

2.1 Related Work

Table 1 compares MinMandate with existing mechanisms, grouped by who checks the mandate: the wallet, the merchants, or no one, when only generic limits apply. Two columns ask whether a mechanism prevents wallet-side tracing and merchant-side linking. Three cover its authorization: *Adapt.*, new merchants need no new approval; *Policy*, each payment follows the mandate's merchant and amount rules; *Binding*, each payment is bound to its request and quote and cannot be replayed or moved to another call.

Wallet-enforced designs. In these designs, the wallet checks each payment against the mandate and thus sees it. Later designs made the rules finer and then hid them, but they still either trace payments or need proofs that grow with the rules. Card spending controls [26, 38, 44] check each payment against coarse merchant categories and amounts. Stripe SPT [39] restricts each token to one merchant, so a new merchant needs a new token. TIVA (Trustless Intent Verification for Autonomous Agents) [1] proves limits in zero knowledge but publishes each payment on-chain, so anyone can trace and link its payments. A wallet verifying anonymous credentials such as BBS [25] or Coconut [36] removes this tracing through presentations unlinkable to any verifier. Used as specified, however, such credentials only disclose or hide each attribute. Disclosing a mandate's rule lets the wallet enforce it but repeats it across the task's payments, while hiding it leaves the rule unchecked. Zero-knowledge proofs of the rule close this gap, but a direct proof grows with the rule (Section 3.1). Keyed-verification anonymous credentials [7] offer a cheaper route, since the issuer verifies with its own secret key. Mirzamohammadi et al. [28] make this check cheap through a polynomial quotient, but only for attributes the holder discloses. Over hidden attributes, keyed-verification credentials still fall back to generic proofs that grow with the rule [30].

Table 1: Mechanism comparison; ● met, ◑ partly, ○ not.

	Prevents		Authorization		
Mechanism	Tracing	Linking	Adapt.	Policy	Binding
Card spending controls [26, 38, 44]	○	○	●	◑	◑
Stripe SPT [39]	○	◑	◑	●	◑
TIVA [1]	○	○	◑	●	●
AP2 mandates [17]	○	○	◑	●	●
Visa TAP [43]	○	○	●	○	◑
Anonymous tokens [6, 11, 34]	●	●	◑	○	◑
E-cash [4, 8, 33]	●	●	●	○	◑
k-TAA [3, 30, 40]	●	●	●	○	◑
MinMandate	●	●	●	●	●

Merchant-enforced designs. Here the merchants check the mandate as well, so one mandate can cover several merchants: in AP2 [17], the user signs a mandate that the wallet and every merchant verify. The wallet sees the full mandate and traces every payment to it. Each merchant sees less. In human-not-present mode (AP2-HNP), the user signs in advance one open mandate pair per allowed call, and the agent closes one pair for each call into a payment-specific mandate that discloses only the merchant being paid [18]; in human-present mode (AP2-HP), the user signs every call. In both modes, every merchant verifies its mandate under a key that repeats across the task's calls, the agent's key under AP2-HNP and the user's signing key under AP2-HP, so colluding merchants link every call, as they can under Visa's Trusted Agent Protocol [43], where each merchant verifies the agent's requests under one published key. Batch issuance in SD-JWT [14] would stop merchants from linking presentations, but the wallet, which checks every presentation, would still see the mandate.

Generic-limit mechanisms. Here no one checks a mandate: these mechanisms hide which approval a payment comes from, but their verifier checks only a count or a balance. E-cash [4, 8, 33] bounds what an anonymous payer spends, and k-times anonymous authentication (k-TAA) [3, 40] bounds how often a credential is shown. Privacy Pass standardized unlinkable tokens for HTTP [6, 11], and Anonymous Credit Tokens [34] add a hidden balance. None of these enforces the mandate's merchant and amount rules, or fully binds a payment to its request and quote. Policy-compliant signatures [2], policy-based tokens [13, 24], and public Privacy Pass metadata [20] attach a readable policy, which partitions payments by their rules and brings back part of the tracing that tokens remove [13].

2.2 System and Security Model

System model. User U approves a mandate that bounds the agent's payments for a task. The mandate sets a total budget and an expiry. It consists of single-use permits, and each permit allows one payment. A permit names a service class, a rule for which merchants it accepts, and a cap on the amount. **Agent** A carries out the task for the user. It holds one credential for each permit. For each paid call, it gets a quote from a merchant and proves to the wallet that a permit allows the payment. **Merchant** M, with identifier μ, sells

a paid service, such as web search or company data. It answers each request with a quote and acknowledges the call. It returns the result only after it receives the wallet's ticket for that call. **Wallet** W holds the user's funds and pays merchants for the agent. Its issuing part $\mathsf{W}_{\mathsf{iss}}$ reserves the budget and issues the permit credentials. Its redeeming part $\mathsf{W}_{\mathsf{red}}$ checks that a permit allows each payment, debits it from a reserve pooled across users, and sends the merchant a signed ticket.

Security goals. MinMandate has two security goals. *Authorization soundness.* The wallet pays only for calls that the user's mandate allows and never beyond its budget, even if the agent and merchants collude. Each payment is bound to the one call it pays for. *Unlinkability.* MinMandate defends against both task leaks, except for the leakage needed to complete each paid call, such as its amount, payee, and time (Definition 4.4). Against wallet-side tracing, the wallet cannot trace a payment to the mandate it comes from or to the user who approved that mandate, even with colluding merchants. Against merchant-side linking, merchants cannot link a task's payments to each other, even when they collude.

Threat model. User. The user and the user's signing device are trusted. The device signs only the mandate that the user reviews, so a malicious agent cannot alter what the user approves. **Agent.** Following recent agent-security work [22], we view the agent as a potential confused deputy: injected tool content or manipulated merchant metadata, such as a tool description that names an attacker's merchant, can steer it into attempting payments that the user did not allow [12, 45]. For authorization soundness, we therefore treat the agent as malicious; it may collude with merchants and send any payment request. For unlinkability, the agent follows the protocol. **Wallet.** The wallet is honest-but-curious: it follows the protocol and keeps all its keys and records, but attempts to link payments to each other or to the approving user to infer the user's task, alone or with any set of merchants. It follows the prescribed admission rules when it certifies merchants. **Merchants.** Merchants are malicious and may collude with each other. To obtain payments beyond the mandate, they may collude with the agent. To link calls across merchants and to the user, and thereby recover the user's task, they pool the requests, payments, and connection metadata they see, and the wallet's records when it colludes.

We assume that the agent sends its requests through an egress shared by all tasks, such as an Oblivious HTTP relay [41], so network identifiers do not separate tasks. Payment withholding by the wallet, denial of service, and the quality of the delivered service are outside our security goals.

2.3 Preliminaries

Notation. Let λ be the security parameter, and let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ be type-3 groups of prime order $p \geq 2^\lambda$ with pairing $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. We write groups additively, encode identifiers, amounts, and times canonically as scalars in $\mathbb{Z}_p$, and let $\mathbb{Z}_p^* = \mathbb{Z}_p \setminus \{0\}$. $\mathbf{H} : \{\mathbf{0},\mathbf{1}\}^* \to \mathbb{Z}_p$ is a hash function that maps messages, such as requests, to scalars.

Assumptions. All adversaries run in probabilistic polynomial time. The external Diffie–Hellman (XDH) assumption holds, that is, decisional Diffie–Hellman is hard in $\mathbb{G}_1$, and the discrete-logarithm problem is hard in $\mathbb{G}_1$. The hash H is collision resistant. Fiat–Shamir challenges and the derivation of generators from fixed labels are modeled as random oracles. KVPC unforgeability additionally uses a restricted generic bilinear-group model.

Signatures. A signature scheme Sig = (KeyGen, Sign, Verify) lets a key holder sign messages that anyone can check with the public key. We use Schnorr signatures [35] under independent keys, which are existentially unforgeable under chosen-message attacks (EUF-CMA) in the random-oracle model.

Commitments. A commitment scheme CS = (Setup, Commit, Open) lets a party commit to a vector $\mathbf{v}$ with randomness r and lets anyone check the opening. We use Pedersen commitments [31], $\mathbf{Com}(\mathbf{v}; \boldsymbol{r}) = \sum_j v_j G_j + rG_\eta$, whose bases are transparent generators of $\mathbb{G}_1$ derived by hashing fixed labels, so no party knows their relative discrete logarithms. They are perfectly hiding, computationally binding, and additively homomorphic.

Zero-knowledge proofs. A non-interactive zero-knowledge proof system NIZK = (Setup, Prove, Verify) lets a prover convince a verifier that a statement x has a witness w with $(x, w) \in \mathcal{R}$, without revealing w; we write $\mathbf{NIZK\{}(\boldsymbol{w}) : \boldsymbol{\mathcal{R}}\mathbf{\}}$. We use Fiat–Shamir Schnorr proofs [15, 35] for linear relations over $\mathbb{G}_1$, and OR proofs [10] that prove one of several statements without revealing which. They are complete, knowledge sound, and zero knowledge.

Keyed-verification credentials. A keyed-verification anonymous credential (KVAC) [7] scheme KVAC = (KeyGen, Issue, Show, Verify) lets an issuer issue credentials on committed attributes without seeing them and verify their presentations with its own secret key. A presentation discloses some attributes and proves a predicate on the others. Presentations cannot be forged, and they are unlinkable to their issuance and to each other beyond the disclosed attributes.

3 MINMANDATE

3.1 Technical Overview

MinMandate has two design goals: the wallet issues and checks permits without seeing them, and each payment is bound to the call it pays for, with no identifier that links a task's calls. Figure 1 follows the screening task through the four phases of MinMandate.

How to check a permit without seeing it. Anonymous tokens and e-cash already hide which approval a payment comes from, but they enforce only a count or a balance, so a steered agent can drain them at any merchant. An anonymous credential can instead carry the merchant rule and the cap as hidden attributes, and each payment proves in zero knowledge that its merchant and amount satisfy them. This enforces the permit, but a direct construction uses OR proofs over up to ℓ merchants and $K + 1$ amount ranks, so each presentation and its verification take $O(\ell + K)$, or $O(\log K)$ with membership and range proofs. Two observations remove this cost. First, both conditions are memberships in a hidden set, which one polynomial identity can check without a proof branch per element. Second, the wallet both issues and verifies the credentials, so it can hold the secret points at which the identity is checked. We therefore introduce the keyed-verification policy credential (KVPC), which encodes the merchant rule and the cap as polynomials whose roots are the accepted merchants and amount ranks, so the wallet checks each payment with one equation at its secret points in $O(1)$ (Section 3.2). The wallet also issues these credentials blindly, so it never sees the permits it later checks (Section 3.3). Hiding the

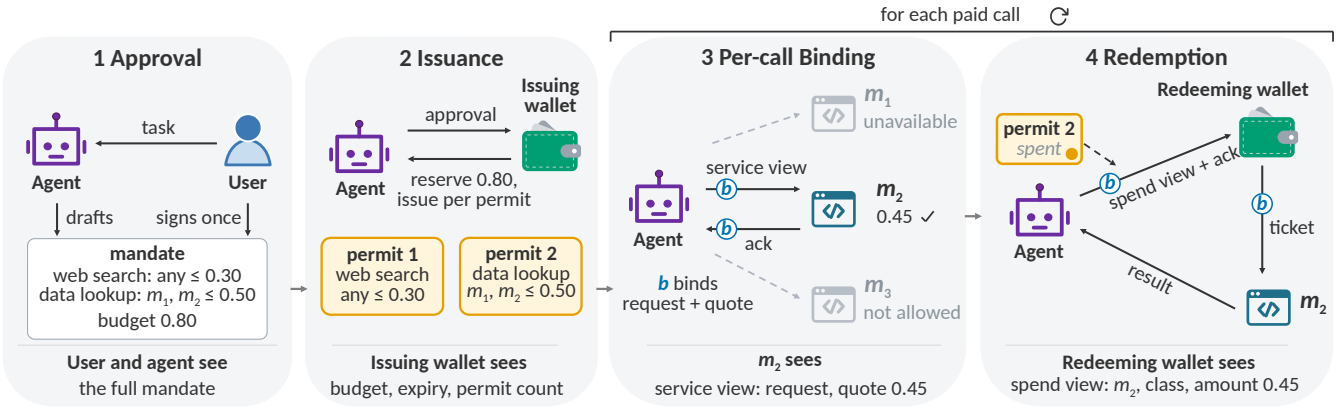

Figure 1: MinMandate workflow on the screening task of Section 1; amounts are examples. Phases 3–4 repeat for each paid call; m_1–m_3 are merchants, and ack is an acknowledgement. Each call carries a binding b of its request and quote: the spend view spends permit 2 on b at the wallet, and the service view opens b to the merchant. The bottom row lists what each party sees.

permits does not stop the user from auditing the spending later, because the agent's records tie each payment to its permit.

How to bind a payment without linking calls. Once the wallet enforces the mandate, merchants no longer need to see it. Each payment must still be bound to one call, so that no quote is paid twice, no payment is applied to another request, and no result is released unpaid. A shared identifier, such as the mandate, a credential, or the agent's key, would bind the payment, but it repeats across a task's calls, so colluding merchants can link every one of them, as under AP2. A fresh identifier per call does not repeat, but it does not say which request and quote the payment is for. One observation resolves this conflict: the two sides only need to agree on the call itself, and every call is different. We therefore give each call one fresh identifier that carries its content, a commitment b to that call's request and quote (Figure 1). The merchant checks that b opens to its request and quote, the wallet checks the quote part and binds the payment and the ticket to b, and since b is new for every call, it links nothing (Section 3.4). The merchant thus never sees the permit, and the wallet never sees the request, as the bottom row of Figure 1 shows. An x402 facilitator, in contrast, learns each request's resource URL and description [37].

3.2 Keyed-Verification Policy Credentials

A *keyed-verification policy credential* (KVPC) lets the wallet check a disclosed merchant and amount against a hidden rule and cap with one constant-size presentation. It extends the polynomial-quotient check of Mirzamohammadi et al. [28] in two ways: (i) it turns the cap into a set of accepted amount ranks, and (ii) it hides whether the merchant rule is an allow-list or admits every merchant.

A policy $\theta = (\alpha, \psi)$ describes one permit. Its public attributes $\alpha = (\gamma, u, \text{expiry})$ are a service class, a fresh serial, and an expiry, and its hidden constraints $\psi = (\varphi, c)$ are a merchant rule and a cap, the largest amount the permit may pay. A presentation discloses the public attributes, which the wallet uses to check the class, double spending, and expiry. The rule φ is either allow$(\mathcal{S})$ for a nonempty set $\mathcal{S}$ of at most ℓ merchants, with ℓ a public bound, or adm, which admits every merchant. The cap c comes from a public denomination table $\mathcal{D} = \{0 = d_0 < \cdots < d_K\}$, and the policy accepts merchant μ and amount a if φ admits μ and $0 < a \le c$. We compare amounts by rank. The cap $c = d_r$ has rank r, and an amount $0 < a \le d_K$ has rank $j = \min\{h \ge 1 : a \le d_h\}$. Because the cap is itself a denomination, $a \le c$ exactly when $j \le r$, that is, when j lies in the set $\{1, \ldots, r\}$ that the polynomial Q_r encodes below.

We encode the rule and the cap as polynomials. For allow$(\mathcal{S})$ and a cap of rank r, we use $f(X) = \prod_{\mu^\star \in \mathcal{S}} (X - \mu^\star)$ and $Q_r(X) = \prod_{h=1}^{r} (X - h)$, e.g., $f(X) = (X - m_1)(X - m_2)$ for permit 2, and the rule adm uses $f \equiv 0$. The policy then accepts merchant μ and an amount of rank j exactly when $f(\mu) = Q_r(j) = 0$. As in KZG [23], the agent evaluates the quotients $f/(X - \mu)$ and $Q_r/(X - j)$ on the public powers, and the wallet checks them at its secret points s and t, as in [28]. The NIZK in Figure 2 is a Schnorr proof. KVPC is unforgeable, and the key holder learns only the disclosed attributes, merchants, and amounts (Section 4).

3.3 One-Time Approval and Issuance

Before any payment, in lines 0–8 of Figure 3, the agent drafts a typed mandate, the user signs it once, and the wallet issues one credential per permit without seeing any permit.

Initialization. The user, each merchant, and the wallet generate signing key pairs $(\text{sk}_\text{U}, \text{pk}_\text{U})$, $(\text{sk}_\mu, \text{pk}_\mu)$, and $(\text{sk}_\text{W}, \text{pk}_\text{W})$. The wallet publishes pk_W and records pk_U when the user opens an account. For each key epoch ep (e.g., one day), the wallet publishes one KVPC key for all users in an append-only log, and agents check the key they receive against it (line 0). When a merchant joins, the wallet operator, as the merchant certification authority, admits it for one service class γ_μ, much like a merchant category code [21]. It signs the certificate $\text{Cert}_\mu = (\mu, \gamma_\mu, \text{pk}_\mu, \sigma_\mu)$ with $\sigma_\mu \leftarrow \text{Sign}_{\text{sk}_\text{W}}(\mu, \gamma_\mu, \text{pk}_\mu)$. All signatures and proofs bind ep, pp, and a label for their type.

Typed mandate. The user states the task in natural language, and the agent compiles it into a typed mandate $\text{mandate} = (\text{mandate}^{\text{pub}}, \text{mandate}^{\text{hid}})$ (line 1). The wallet sees only the public part, the budget B and the expiry. The hidden part lists n permits, each a service class γ_k with hidden constraints $\psi_k = (\varphi_k, c_k)$. A merchant is eligible for a permit only if its certificate names the permit's class, so adm accepts any admitted merchant of the class, including merchants admitted after the

Setup$(1^\lambda) \to$ pp: Choose type-3 groups of prime order p with generators $G \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$. Derive the transparent generators $H_0, H_u, H_\gamma, H_T, H_P, G_\zeta, G_c, G_d, G_\eta \in \mathbb{G}_1$ and the coordinate bases of Com, and fix the denominations $\mathcal{D} = \{0 = d_0 < \cdots < d_K\}$ and the allow-list bound ℓ. Output all of them as pp.

KeyGen$(\text{pp}) \to (\text{sk}, \text{ipk})$: Sample $\xi \leftarrow\$\ \mathbb{Z}_p^*$, $s \leftarrow\$\ \mathbb{Z}_p$, and $t \leftarrow\$\ \mathbb{Z}_p \setminus \{1, \ldots, K\}$. Output $\text{sk} = (\xi, s, t)$ and $\text{ipk} = (\xi g_2, \{S_i = s^i G\}_{i=1}^{\ell}, T = tG, \{t^h H_P\}_{h=1}^{K})$, where $S_0 = G$ and $\mathbf{S} = (S_0, \ldots, S_\ell)$.

Commit$(\text{ipk}, \theta; \zeta) \to (M, L)$ or $\perp$: Take a policy $\theta = (\alpha, \psi)$ with public attributes $\alpha = (\gamma, u, \text{expiry})$ and hidden constraints $\psi = (\varphi, c)$, where $c = d_r$ and $f(X) = \sum_{i=0}^{\ell} f_i X^i$ with $\mathbf{f} = (f_0, \ldots, f_\ell)$, and a blinding $\zeta \leftarrow\$\ \mathbb{Z}_p^*$. Let $H_\alpha = H_0 + \gamma H_\gamma + u H_u + \text{expiry}\, H_T$ and compute

$$M = H_\alpha + f(s)G + \zeta Q_r(t) H_P, \qquad L = \zeta G_\zeta + c G_c \tag{1}$$

from $f(s)G = \langle \mathbf{f}, \mathbf{S} \rangle$ and $t^h H_P$. Output $\perp$ if $M = 0$ or $L \in \{d_h G_c\}_{h=0}^{K}$.

Issue$(\text{sk}, M) \to$ cred: Sample $e \leftarrow\$\ \mathbb{Z}_p \setminus \{-\xi\}$ and output $\text{cred} = (V, e)$ with $V = (\xi + e)^{-1} M$.

Check$(\text{ipk}, M, \text{cred}) \to b$: Parse $\text{cred} = (V, e)$ and output $\mathrm{e}(V, \xi g_2 + e g_2) \stackrel{?}{=} \mathrm{e}(M, g_2)$.

Show$(\text{ipk}, \text{ctx}, (\theta, \zeta, \text{cred}), (d, \eta)) \to \tau$: Take α, μ, a, and the linked point P from ctx, and let j be the rank of a. Require $P = dG_d + \eta G_\eta$ and $f(\mu) = Q_r(j) = 0$, and parse $\text{cred} = (V, e)$. Evaluate the quotients $f_\mu(X) = f(X)/(X - \mu)$ and $Q_{r,j}(X) = Q_r(X)/(X - j)$ on the public powers. Sample $\rho \leftarrow\$\ \mathbb{Z}_p^*$ and $\kappa \leftarrow\$\ \mathbb{Z}_p$, and set $\widehat{V} = \rho V$,

$$W_1 = \rho f_\mu(s) G + \kappa(T - jG), \quad W_2 = \rho \zeta Q_{r,j}(t) H_P - \kappa(S_1 - \mu G), \tag{2}$$

and $Y = \rho H_\alpha - e\widehat{V}$. Output $\tau = (\widehat{V}, W_1, W_2, \pi)$ with

$$\pi \leftarrow \mathsf{NIZK}\{(\rho, e, d, \eta) : Y = \rho H_\alpha - e\widehat{V} \wedge P = dG_d + \eta G_\eta\} \tag{3}$$

for the statement $x = (\text{ipk}, \text{ctx}, \widehat{V}, W_1, W_2, Y)$.

Verify$(\text{sk}, \text{ctx}, \tau) \to b$: Take α, μ, a, and P from ctx, and let j be the rank of a. Output $b = 0$ unless ctx is well formed and $0 < a \le d_K$. Otherwise compute $Y = \xi\widehat{V} - (s - \mu)W_1 - (t - j)W_2$ and output $b = 1$ iff $\widehat{V} \neq 0$ and $\mathsf{NIZK.Verify}(x, \pi) = 1$ for $x = (\text{ipk}, \text{ctx}, \widehat{V}, W_1, W_2, Y)$.

Figure 2: KVPC algorithms. The wallet runs KeyGen**, then** Issue **and** Verify **with its secret key; the agent runs** Commit, Check**, and** Show **with the public key.**

Public: permit count n, denominations $\mathcal{D} = \{d_h\}_{h=0}^{K}$, and verification keys pk_U, pk_μ, and pk_W of U, M, and W

Setup, approval, and issuance

0	W_{iss}	run $\text{pp} \leftarrow \mathsf{KVPC.Setup}(1^\lambda)$ once; per epoch, run $(\text{sk}_{\text{ep}}, \text{ipk}_{\text{ep}}) \leftarrow \mathsf{KVPC.KeyGen}(\text{pp})$ and publish ipk_{ep} in a log
1	A	draft $\text{mandate}^{\text{pub}} = (B, \text{expiry})$ and $\text{mandate}^{\text{hid}} = ((\gamma_k, \psi_k))_{k \in [n]}$ for the user's task
2	A	for $k \in [n]$: sample $u_k, \zeta_k \leftarrow\$\ \mathbb{Z}_p^*$; set $\alpha_k = (\gamma_k, u_k, \text{expiry})$ and $\theta_k = (\alpha_k, \psi_k)$; commit $D_k = (M_k, L_k) \leftarrow \mathsf{KVPC.Commit}(\text{ipk}_{\text{ep}}, \theta_k; \zeta_k)$, resampling ζ_k on $\perp$, and keep (θ_k, ζ_k)
3	$\mathsf{A} \to \mathsf{U}$	send mandate, $\mathbf{D} = (D_k)_{k \in [n]}$, openings $(\theta_k, \zeta_k)_{k \in [n]}$
4	U	check that each D_k commits to permit k; sign $\sigma_\mathsf{U} \leftarrow \mathsf{Sign}_{\text{sk}_\mathsf{U}}(\text{ipk}_{\text{ep}}, \mathbf{D}, \text{mandate}^{\text{pub}})$ to approve
5	$\mathsf{A} \to \mathsf{W}_{\text{iss}}$	send $\mathbf{D}$, $\text{mandate}^{\text{pub}}$, σ_U, π^{iss} of Equation (4)
6	W_{iss}	verify σ_U and π^{iss}; if σ_U was seen before, resend its credentials; check $M_k \neq 0$, $L_k \notin \{d_h G_c\}_{h=0}^{K}$; reserve B; issue $\text{cred}_k \leftarrow \mathsf{KVPC.Issue}(\text{sk}_{\text{ep}}, M_k)$ for $k \in [n]$
7	$\mathsf{W}_{\text{iss}} \to \mathsf{A}$	send $(\text{cred}_k)_{k \in [n]}$
8	A	keep cred_k if $\mathsf{KVPC.Check}(\text{ipk}_{\text{ep}}, M_k, \text{cred}_k) = 1$

Per-call payment *call i to merchant μ_i*

9	$\mathsf{M}_i \to \mathsf{A}$	send $q_i = (\mu_i, a_i, \gamma_i, \text{nonce}_i)$ and Cert_{μ_i} in reply to req_i
10	A	pick an unused k_i whose θ_{k_i} accepts (μ_i, a_i); sample $\eta_i \leftarrow\$\ \mathbb{Z}_p$; set $d_i = \mathsf{H}(\text{req}_i)$, $C_i = \mathsf{Com}(d_i, q_i; \eta_i)$, and $p_i = (\mu_i, a_i, \text{nonce}_i)$
11	$\mathsf{A} \to \mathsf{M}_i$	send req_i again with $\mathcal{V}_i^S = (C_i, p_i, d_i, q_i, \eta_i)$ to pay for it
12	M_i	check that (d_i, q_i, η_i) opens C_i, $d_i = \mathsf{H}(\text{req}_i)$, q_i is fresh, and p_i matches q_i; sign $\text{ack}_i \leftarrow \mathsf{Sign}_{\text{sk}_{\mu_i}}(C_i, p_i)$ for the call it will serve
13	$\mathsf{M}_i \to \mathsf{A}$	send ack_i
14	A	compute $\text{ctx}_i = (\alpha_{k_i}, C_i, p_i, \text{Cert}_{\mu_i})$ and $\pi_i \leftarrow \mathsf{KVPC.Show}(\text{ipk}_{\text{ep}}, \text{ctx}_i, (\theta_{k_i}, \zeta_{k_i}, \text{cred}_{k_i}), (d_i, \eta_i))$ to show that θ_{k_i} accepts (μ_i, a_i)
15	$\mathsf{A} \to \mathsf{W}_{\text{red}}$	send $\mathcal{V}_i^P = (\text{ctx}_i, \pi_i)$, ack_i
16	W_{red}	verify Cert_{μ_i} and ack_i; check expiry, ep, $\gamma_i = \gamma_{\mu_i}$, and $\mathsf{KVPC.Verify}(\text{sk}_{\text{ep}}, \text{ctx}_i, \pi_i) = 1$ on the point linked to C_i
17	W_{red}	return the cached result for an exact retry; else, if u_{k_i} and (μ_i, nonce_i) are unused, atomically mark both used, debit a_i, and sign $\text{tkt}_i \leftarrow \mathsf{Sign}_{\text{sk}_\mathsf{W}}(C_i, p_i)$; otherwise reject
18	$\mathsf{W}_{\text{red}} \to \mathsf{M}_i$	send tkt_i
19	M_i	verify tkt_i on (C_i, p_i) and only then serve req_i to the agent

Figure 3: MinMandate protocol. Line 0 is setup, lines 1–8 run once per approval, and lines 9–19 once per paid call.

approval. For example, permit 1 is (web search, (adm, 0.30 USD)), permit 2 is (data lookup, (allow($\{m_1, m_2\}$), 0.50 USD)), and B is 0.80 USD. Within these rules, the agent chooses which merchant to call. The agent fills every mandate up to the public number n of permits with zero-cap permits, which accept no payment, and picks the expiry from a public grid with step Δ.

Committed approval. The agent turns each permit into a KVPC policy and commits to it (line 2). The user opens the commitments, checks them against the mandate, and signs them together with the mandate's public part, its budget and expiry (lines 3–4).

Blind issuance. Before it reserves funds, the wallet must know that the hidden permits are well formed and fit the budget. The agent sends W_{iss} the signed approval with the proof π^{iss} below (line 5), which, as Figure 5 details, shows that the caps fit the budget over the integers. The wallet verifies it, reserves B, and issues the credentials blindly (lines 6–8), where

$$\pi^{\text{iss}} \leftarrow \mathsf{NIZK}\left\{\begin{array}{l}(\gamma_k, u_k, \mathbf{f}_k, r_k, \zeta_k)_{k \in [n]} : \sum_k d_{r_k} \le B \wedge \\ \forall k : r_k \in \{0, \ldots, K\} \wedge L_k = \zeta_k G_\zeta + d_{r_k} G_c \wedge \\ M_k = H_{\alpha_k} + \langle \mathbf{f}_k, \mathbf{S} \rangle + \zeta_k Q_{r_k}(t) H_P\end{array}\right\}. \tag{4}$$

3.4 Unlinkable Per-Call Views

For each paid call, the wallet receives a spend view and the merchant a service view, linked by a fresh per-call binding (lines 9–19).

Per-call binding. For the ith paid call, merchant μ_i quotes an amount a_i with a one-time nonce for the agent's request req_i (line 9). The agent picks a permit k_i that accepts (μ_i, a_i) and binds the call to the pair (C_i, p_i), where C_i commits to the request and the quote and p_i copies the quote's payment fields (line 10). In the

example, m_1 is unavailable and m_2 quotes 0.45 USD, so the agent uses permit 2, which admits m_2 for up to 0.50 USD.

Service and spend views. The merchant receives the opening of C_i and acknowledges the pair (lines 11–13). The agent then proves with a KVPC presentation that permit k_i accepts the payment and sends the presentation with the acknowledgement to $\mathsf{W}_{\mathsf{red}}$ (lines 14–15). For KVPC.Verify, $\mathsf{W}_{\mathsf{red}}$ rebuilds the quote part of C_i from p_i and the class, and subtracts it to obtain the linked point (line 16). In the example, the wallet does not learn that permit 2 accepts only m_1 and m_2 or that its cap is 0.50 USD.

Redemption. $\mathsf{W}_{\mathsf{red}}$ consumes the serial and the quote atomically, debits a_i from a reserve pooled across users, and returns a ticket that the merchant checks before it serves the request (lines 17–19). A permit cannot pass its unused cap to another permit, and Appendix B discusses how to return unused budget and how to hold a payment until the merchant releases its result.

4 SECURITY ANALYSIS

4.1 KVPC Security

KVPC is unforgeable in a restricted generic group model and hides each permit's rule, cap, and credential from the key holder.

Theorem 4.1 (KVPC unforgeability; Appendix A.1). *In the restricted generic type-3 bilinear-group model of Appendix A.1 with a random oracle, an adversary with verification oracles and issuing oracles that check each commitment's opening gets a presentation for (α, μ, a) accepted with negligible probability unless an issued credential of that epoch has attributes α and a policy accepting (μ, a).*

Proof sketch. For a hash input with two accepting nonzero challenges, special soundness turns acceptance into $(\xi + e)\widehat{V} = \rho H_\alpha + (s-\mu)W_1 + (t-j)W_2$, and taking residues in the generic group forces $\widehat{V} = \rho V_k$ and $e = e_k$ for one earlier credential with matching public attributes. Reducing its verification identity modulo $(s-\mu, t-j)$ leaves $f(\mu)$ and $Q_r(j)$, so both vanish and the policy accepts (μ, a). Adding guessed or exceptional challenges, degenerate or colliding issuance exponents, and accidental evaluations gives

$$\begin{aligned}\mathsf{Adv}^{\mathsf{uf}} \le \frac{Q_A}{p-1} + \frac{Q_A+Q_V}{p} + \frac{N_I}{p} + \sum_{\nu} \frac{n_\nu(n_\nu-1)}{2(p-1)} \\ + \frac{N_{\mathsf{test}}(N_I + \max\{K,\ell\}+2)}{p-K-1} \le \mathsf{negl}(\lambda),\end{aligned} \tag{5}$$

where Q_A counts presentation hash queries, Q_V verifications on unqueried inputs, N_I issued credentials (n_ν per epoch), and N_{test} potential collisions and tests; counts and parameters are polynomial and $p \ge 2^\lambda$. □

Theorem 4.2 (KVPC key-holder privacy; Appendix A.1). *Under XDH, with the Fiat–Shamir hash and generator derivation as random oracles, an honest-but-curious key holder with all its keys, coins, and records distinguishes two admissible schedules (Definition A.2) with equal leakage at every prefix with negligible advantage.*

Proof sketch. Starting from the real execution Hyb_0, Hyb_1 simulates each fresh presentation proof by programming the random oracle. Hyb_2 charges the rare blindings that Commit rejects to a bad event. Hyb_3 and Hyb_4 decouple each commitment's L from M and each presentation from its credential, one DDH step each. Hyb_5 samples serials along the leaked equality pattern and depends only on the leakage, so summing the hop costs gives

$$\begin{aligned}\mathsf{Adv}^{\mathsf{kh}} \le (N+N_\tau)\varepsilon_{\mathsf{XDH}} + \frac{N_\tau Q_H + N_\tau(N_\tau-1)/2}{p} \\ + \frac{2N(K+2)}{p} + \frac{N(N-1)}{2(p-1)} \le \mathsf{negl}(\lambda),\end{aligned} \tag{6}$$

where N counts honest permits, N_τ fresh presentations, and Q_H other random-oracle queries; the DDH gap $\varepsilon_{\mathsf{XDH}}$ is negligible. □

4.2 Authorization Soundness

Authorization soundness follows from KVPC unforgeability, commitment binding, the issuance proof, and signature unforgeability.

Theorem 4.3 (Authorization soundness; Appendix A.2). *In the model of Theorem 4.1, with EUF-CMA signatures on type-labelled messages, a collision-resistant hash, budgets and caps within the integer ranges of Figure 5, an honest user and wallet, and a malicious agent colluding with merchants, except with negligible probability, each fresh accepted payment spends a distinct, unexpired approved permit whose class matches the merchant's certificate and whose policy accepts the merchant and amount, pays no quote twice, matches its quote and, for an honest merchant, the acknowledged request, and keeps its approval within budget.*

Proof sketch. Abort on signature forgeries, hash collisions, and issuance proofs accepted for a false statement. Outside the generic bad event in $\varepsilon_{\mathsf{uf}}$, accepted batches commit to the user's checked openings and satisfy $\sum_k d_{r_k} \le B$ over the integers, so the wallet acts as the issuing oracle of Theorem 4.1. Its tracing statement (Appendix A.1) maps each accepted call to an approved permit with matching epoch and attributes whose policy accepts (μ_i, a_i). The linked-point proof opens C_i to the quote, and commitment binding matches this opening with the honest merchant's acknowledged opening. The wallet's expiry, class, serial, and quote checks give distinct permits, one payment per quote, and $\sum_i a_i \le \sum_k d_{r_k} \le B$ per approval. Summing these costs gives

$$\mathsf{Adv}^{\mathsf{auth}} \le \varepsilon_{\mathsf{outer}} + \frac{Q_I}{p} + \varepsilon_{\mathsf{uf}} \le \mathsf{negl}(\lambda), \tag{7}$$

where the forgery and collision advantages $\varepsilon_{\mathsf{outer}}$ are negligible, Q_I counts issuance-proof hash queries, and $\varepsilon_{\mathsf{uf}}$ is the right side of Equation (5) over the whole protocol. □

4.3 Unlinkability

Unlinkability holds up to each paid call's declared fields.

Definition 4.4 (Leakage and views). A workflow $\mathcal{P}$ is a history of approvals and paid calls, including failed and repeated attempts, together with the permit that pays each call. An observer's *leakage* is what we allow it to learn; its *view* is what it receives. Merchants may learn the request, quote, and epoch of each call attempt $i \in I_{\mathsf{M}}$, and the wallet may also learn, per issuance j and redemption r:

$$\mathcal{L}_{\mathsf{M}}(\mathcal{P}) = \big(\mathsf{pub},\ (\mathsf{req}_i,\ q_i,\ \mathsf{ep}_i)_{i\in I_{\mathsf{M}}}\big), \tag{8}$$

$$\begin{aligned}\mathcal{L}_{\mathsf{W}}(\mathcal{P}) = \big(\mathcal{L}_{\mathsf{M}}(\mathcal{P}),\ (\mathsf{acct}_j, \mathsf{ep}_j, B_j, \mathsf{expiry}_j)_{j\in I_{\mathsf{iss}}}, \\ (\mathsf{ep}_r, \mathsf{expiry}_r, \gamma_r, p_r)_{r\in I_{\mathsf{red}}}, \mathsf{EQ}\big),\end{aligned} \tag{9}$$

where pub holds the public parameters, configuration, and keys, acct_j is the approving account, and $\mathsf{EQ} = \{(r, r') : u_r = u_{r'}\}$ marks

reused serials. Message sizes and timing are also leaked, but no network identifier, since the egress is shared (Appendix A.2). A merchant's view adds the service views, acknowledgements, and tickets of its own calls. The wallet's view adds every merchant's view, since it may collude with them, and its own state st_W: its keys, coins, records, and every message it receives:

$$\mathsf{View}_M(\mathcal{P}) = \mathcal{L}_M(\mathcal{P}) \cup \bigcup_{i \in I_M} (\mathcal{V}_i^S \cup \{\mathsf{ack}_i, \mathsf{tkt}_i\}), \quad (10)$$

$$\mathsf{View}_W(\mathcal{P}) = \mathsf{View}_M(\mathcal{P}) \cup \mathcal{L}_W(\mathcal{P}) \cup \mathsf{st}_W(\mathcal{P}). \quad (11)$$

Equal-leakage workflows may still differ in hidden rules, caps, and the approval and permit that pay each call; Section 5.2 measures how much the leaked fields still link.

Theorem 4.5 (Unlinkability against the wallet; Appendix A.2). *Under Theorem 4.2's assumptions, with an honest user and agent, an honest-but-curious wallet colluding with merchants distinguishes the* View_W *of two admissible workflows (Definition A.5) with equal* $\mathcal{L}_W$ *at every prefix with negligible advantage.*

Proof sketch. Hyb_1 simulates all issuance and call proofs by programming one random oracle. User signatures and credential issuance need no openings, so Hyb_2 to Hyb_5 repeat the blinding, commitment, presentation, and serial hops of Theorem 4.2 over the whole history, after which the view is simulatable from $\mathcal{L}_W$ alone, giving

$$\mathsf{Adv}^W \le (N + N_\tau)\varepsilon_{\mathsf{XDH}} + \frac{N_\pi Q_H + N_\pi(N_\pi - 1)/2}{p} + \frac{2N(K+2)}{p} + \frac{N(N-1)}{2(p-1)} \le \mathsf{negl}(\lambda), \quad (12)$$

where N_π counts fresh proofs and the rest is as in Equation (6). □

Theorem 4.6 (Unlinkability against merchants; Appendix A.2). *With an honest user, agent, and wallet, unbounded colluding merchants have identically distributed* View_M *of two admissible workflows (Definition A.4) with equal* $\mathcal{L}_M$ *at every prefix.*

Proof sketch. Merchants see only service views, their own acknowledgements, and tickets. Coupling the merchants' coins and fresh commitment and ticket randomness across the workflows, and replaying exact retries, gives identical views, so $\mathsf{Adv}^M = 0$. □

5 EVALUATION

Our experiments show that MinMandate is practical and private: tasks continue to run when merchants fail, payments reveal about as much as under anonymous tokens, the hidden mandate is enforced as well as AP2's, and a payment takes 197 ms.

Workloads. All experiments use one cohort of agent tasks, which we built on the AgentDojo sandbox [12]. The cohort has 120 target tasks from 24 templates and 1,590 background tasks. Each task states a goal in the user's words and needs 3 to 6 paid calls, 4.25 on average; the overhead task has four. Paid calls go to sandboxed versions of 35 real services from 29 companies, such as Amazon SES for mail, at per-call prices derived from their published tariffs, and each service is unavailable with probability p_{unavail}.

Models and environment. Language models play three roles. *Planners* carry out the tasks, an *attacker* reconstructs task goals from the merchants' linked records, and *judges* score these reconstructions. The planners are three open-weight models of increasing size, Qwen 2.5 14B (Q14), Qwen 2.5 32B (Q32) [47], and GPT-OSS 120B [29], so that no utility result relies on one model; each runs every target task under three seeds. The attacker links payments that carry the same value, scores other pairs with a logistic model trained on the background tasks, and clusters them by average linkage; GPT-OSS reconstructs each task's goal from the linked payments. The primary judge is Llama 3.3 70B [27], from a third model family, and Qwen3 32B [48] is the second judge; a judge marks a task recovered when the reconstruction matches the goal's main fields. All experiments run on an Intel Xeon Gold 6133 server with NVIDIA A800 80 GB GPUs. We implement MinMandate in Rust over BLS12-381, which targets 128-bit security, with 29 denominations ($K = 28$) and allow-list bound $\ell = 1$. Code and experiment scripts are at https://anonymous.4open.science/r/minmandate-review-33ED.

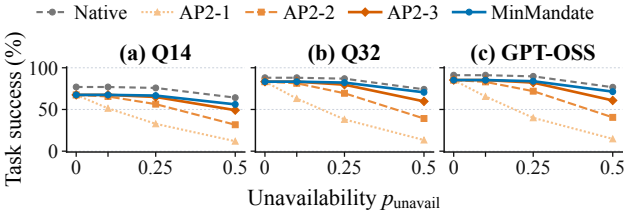

Figure 4: Task utility under merchant unavailability. The percentage of the 120 tasks that each mechanism completes (↑), for each planner, against p_{unavail}.

Metrics. We measure four quantities. *Utility* is the task success rate when merchants may be unavailable. *Privacy* is measured by three attacks. Against wallet-side tracing, *attribution* is the percentage of payments traced to their mandate. Against merchant-side linking (Section 2.1), *linkage* is the average precision of deciding whether two payments come from the same mandate, and *task recovery* is the fraction of tasks whose goal the attacker reconstructs, as rated by a judge. *Enforcement* is the percentage of injected payments that the wallet pays when a prompt injection steers the agent: an admitted attacker merchant of the task's class (*Same*) or of another class (*Cross*) asks to be paid in its tool description or in a call's result, or a merchant raises its quote above the matching permit's cap (*Raise*); we replay every injected payment that the agent makes without a wallet. *Overhead* is latency and bytes per issuance, payment, and task. Reported differences are significant at the 95% level under a paired bootstrap over tasks.

Baselines. The closest baseline is AP2 v0.2 in both of its modes (Section 2.1). *Native*, which lets the agent pay for every call without authorization, gives the highest success; *Policy-only* checks the same mandate in plaintext, with no credentials or proofs. For utility, we compare with *AP2-k*, an AP2-HNP mandate that lists the user's first $k \in \{1, 2, 3, \text{all}\}$ preferred merchants per service. For the other three quantities, we compare with AP2, *Privacy Pass* [6], and *divisible e-cash* [33]. We also test *public-policy tokens*, an ablation that removes KVPC from MinMandate and makes the policy public [20].

Table 2: Privacy and enforcement under each mechanism; –: not applicable. (a) Each attack's score on everything the party sees, at $N_{\text{act}} = 128$. (b) The percentage of payments that the wallet traces to the right mandate when N_{act} tasks are active. (c) The percentage of injected payments that each wallet pays, by kind of injection. All panels use Q32 as the planner.

(a) Attack scores (%, ↓)

	Adversary	Metric	AP2-HP	AP2-HNP	Privacy Pass	E-cash	Public policy	**MinMandate**
$p_{\text{unavail}} = 0$	Wallet	Attribution	–	100.0	–	–	4.3	0.8
		Linkage	100.0	100.0	0.6	0.7	1.8	1.0
	Merchants	Linkage	100.0	100.0	2.3	2.3	2.4	2.3
		Task recovery (Llama)	40.7	47.2	6.6	8.0	7.0	8.3
		Task recovery (Qwen3)	40.7	41.1	5.8	5.8	6.7	5.8
	Wallet with merchants	Attribution	–	100.0	–	–	3.3	0.8
		Linkage	100.0	100.0	9.5	9.4	10.3	11.8
		Task recovery (Llama)	45.6	47.4	11.3	9.2	10.6	9.8
		Task recovery (Qwen3)	27.8	38.2	8.6	6.1	7.0	7.6
$p_{\text{unavail}} = 0.25$	Wallet	Attribution	–	100.0	–	–	9.4	0.4
		Linkage	100.0	100.0	0.6	0.8	2.0	0.9
	Merchants	Linkage	100.0	100.0	2.2	2.2	2.2	2.2
		Task recovery (Llama)	48.9	48.2	5.8	8.3	7.0	8.6
		Task recovery (Qwen3)	41.9	40.4	6.1	6.4	5.2	5.5
	Wallet with merchants	Attribution	–	100.0	–	–	10.4	0.5
		Linkage	100.0	100.0	9.9	11.2	10.1	10.9
		Task recovery (Llama)	43.7	43.9	10.4	8.6	9.5	9.8
		Task recovery (Qwen3)	30.0	40.5	6.4	6.1	6.7	6.4

(b) Wallet tracing under N_{act} tasks (%, ↓)

Mechanism	N_{act}		
	32	128	256
AP2-HNP	100.00	100.00	100.00
Public policy	16.30	4.26	1.44
Random guess	2.08	0.51	0.25
MinMandate	2.50	0.80	0.06

(c) Injected payments paid (%, ↓)

Mechanism	Same	Cross	Raise
AP2-HNP	39.6	0.0	32.0
Privacy Pass	43.9	99.2	96.3
E-cash	80.2	68.0	60.3
Public policy	29.1	0.0	9.1
MinMandate	29.1	0.0	9.1

Table 3: Overhead of one task with four paid calls (↓); –: not applicable. Median latency and bytes per issuance, payment, and task at a 50 ms RTT, and user approvals per task.

	Latency (ms)			Bytes (KiB)		
Mechanism	Issue	Call	Task	Issue	Call	Approvals
AP2-HP	–	185.8	730.4	–	9.13	4
AP2-HNP	56.2	138.2	606.6	15.82	8.78	1
Privacy Pass	144.8	151.3	750.0	0.77	0.14	1
E-cash	133.9	605.9	2558.8	0.54	2.70	1
MinMandate	371.4	197.0	1166.9	35.32	4.08	1

5.1 Task Utility

As Figure 4 shows, hiding the mandate does not reduce utility: MinMandate succeeds on exactly the executions that Policy-only does. Without outages, every mandate, MinMandate's included, completes 6.7 points fewer tasks than Native, because the agent makes paid calls beyond the mandate's permits. As outages grow, MinMandate keeps tasks running without a new approval and never falls below AP2-k, whose k fixed merchants may all be unavailable; the gap to AP2-1 grows to 52.7 points at $p_{\text{unavail}} = 0.5$. AP2-all, which lists every merchant of each service, matches MinMandate's success, but its agent key still links the task's payments.

5.2 Privacy and Enforcement

As Table 2 shows, unlike AP2, MinMandate's scores stay within three points of Privacy Pass and e-cash, which carry no mandate, and it pays the same injected payments as a public policy.

Wallet-side tracing. MinMandate's wallet traces payments not significantly better than random guessing, while AP2-HNP's traces every payment. AP2's wallet checks each payment against the task's signed mandate, whereas MinMandate's sees only public fields, which fit many active mandates. It attributes 0.8% of the payments at $N_{\text{act}} = 128$, where a uniform guess among the unexpired mandates that fit each payment scores 0.51% (Table 2b). KVPC hides the mandates' rules: a public policy that names its merchants raises attribution to 3–10% (Table 2a). **Merchant-side linking.** MinMandate removes the link that AP2's keys give colluding merchants: every merchant sees the same key in each call's mandate, such as the agent key in the `cnf` claim [19], so they link a task's calls and rebuild the task in 41–47% of the cases. Under MinMandate they rebuild 8.3%, close to Privacy Pass and e-cash ($p_{\text{unavail}} = 0$, Llama judge). **Agent-side steering.** MinMandate's permits block the payments that leave the mandate: its wallet pays none of the Cross payments and 9.1% of the Raise ones, while Privacy Pass pays 99.2% and 96.3% (Table 2c). It pays no more than AP2-HNP for any kind of injection, and the payments it accepts stay within the class and cap of another unused permit.

5.3 System Overhead

Table 3 shows that MinMandate's cost is practical for a paid call. A payment takes 1.4 times AP2-HNP's latency and about half of AP2's bytes. The largest single cost is issuance, paid once per task: its joint proof has one branch per denomination per permit, so it grows linearly with n and K. As the allow-list bound ℓ grows from 1 to 16, a KVPC presentation stays at 304 bytes and 5 ms of verification, while a credential that proves the same rule and cap with OR proofs grows from 4.1 to 8.0 KiB and from 7.9 to 20.0 ms. Privacy Pass is faster and divisible e-cash slower per call, but neither enforces a merchant rule or a per-permit cap.

6 CONCLUSION

We presented MinMandate, a task-hiding payment authorization system for AI agents in which the user approves a task-bounded mandate once. The wallet checks each payment through a keyed-verification policy credential, which reveals neither what the mandate allows nor which mandate the payment comes from, and each merchant accepts the payment through a one-time ticket. The wallet verifies a hidden merchant rule and cap with one equation at its secret points, so the proof stays constant-size instead of growing with the allowed merchants. Formal analysis and experiments show that its payments reveal about as much as anonymous tokens while the mandate is enforced as well as AP2's, at a practical cost.

REFERENCES

[1] Vivek Acharya. 2025. Secure Autonomous Agent Payments: Verifying Authenticity and Intent in a Trustless Environment. arXiv:2511.15712.

[2] Christian Badertscher, Mahdi Sedaghat, and Hendrik Waldner. 2024. Unlinkable Policy-Compliant Signatures for Compliant and Decentralized Anonymous Payments. *Proc. Priv. Enhancing Technol.* 2024, 4 (2024), 226–267.

[3] Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. 2006. How to Win the Clone Wars: Efficient Periodic n-Times Anonymous Authentication. In *ACM CCS*. 201–210.

[4] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. 2005. Compact E-Cash. In *EUROCRYPT*. 302–321.

[5] Matteo Campanelli, Rosario Gennaro, Steven Goldfeder, and Luca Nizzardo. 2017. Zero-Knowledge Contingent Payments Revisited: Attacks and Payments for Services. In *ACM CCS*. 229–243.

[6] Sofia Celi, Alex Davidson, Steven Valdez, and Christopher A. Wood. 2024. Privacy Pass Issuance Protocols. RFC 9578, IETF.

[7] Melissa Chase, Sarah Meiklejohn, and Greg Zaverucha. 2014. Algebraic MACs and Keyed-Verification Anonymous Credentials. In *ACM CCS*. 1205–1216.

[8] David Chaum, Amos Fiat, and Moni Naor. 1988. Untraceable Electronic Cash. In *CRYPTO*. 319–327.

[9] Clay. 2026. MCP in Clay. Retrieved 2026-10-25 from https://university.clay.com/docs/mcp-settings

[10] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. 1994. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In *CRYPTO*. 174–187.

[11] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. 2018. Privacy Pass: Bypassing Internet Challenges Anonymously. *Proc. Priv. Enhancing Technol.* 2018, 3 (2018), 164–180.

[12] Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. 2024. AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents. In *NeurIPS*. 82895–82920.

[13] Dennis Faut, Julia Hesse, Lisa Kohl, and Andy Rupp. 2025. Scalable and Fine-Tuned Privacy Pass from Group Verifiable Random Functions. In *IEEE EuroS&P*. 38–59.

[14] Daniel Fett, Kristina Yasuda, and Brian Campbell. 2025. Selective Disclosure for JSON Web Tokens. RFC 9901, IETF.

[15] Amos Fiat and Adi Shamir. 1986. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *CRYPTO*. 186–194.

[16] Steven Goldfeder, Harry Kalodner, Dillon Reisman, and Arvind Narayanan. 2018. When the Cookie Meets the Blockchain: Privacy Risks of Web Payments via Cryptocurrencies. *Proc. Priv. Enhancing Technol.* 2018, 4 (2018), 179–199.

[17] Google. 2026. Agent Payments Protocol (AP2) Specification, v0.2. Retrieved 2026-10-25 from https://ap2-protocol.org/ap2/specification/

[18] Google. 2026. AP2 Documentation: Security and Privacy Considerations. Retrieved 2026-10-25 from https://ap2-protocol.org/ap2/security_and_privacy_considerations/

[19] Google. 2026. AP2 v0.2 Reference Shopping Agent, commit b4587ac. Retrieved 2026-10-25 from https://github.com/google-agentic-commerce/AP2/tree/b4587ac

[20] Scott Hendrickson and Christopher A. Wood. 2026. Privacy Pass Issuance Protocols with Public Metadata. Internet-Draft draft-ietf-privacypass-public-metadata-issuance-03, IETF.

[21] ISO. 2023. Retail Financial Services – Merchant Category Codes. ISO 18245:2023.

[22] Rishi Jha, Harold Triedman, Justin Wagle, and Vitaly Shmatikov. 2026. Breaking and Fixing Defenses Against Control Flow Hijacking in Multi-Agent Systems. In *ICLR*.

[23] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. 2010. Constant-Size Commitments to Polynomials and Their Applications. In *ASIACRYPT*. 177–194.

[24] David Kretzler, Yong Li, and Codrin Ogreanu. 2026. Completing Policy-based Anonymous Tokens: Private Bits, Public Metadata and more.... In *ACM AsiaCCS*. 661–675.

[25] Tobias Looker, Vasilis Kalos, Andrew Whitehead, Mike Lodder, and Greg M. Bernstein. 2026. The BBS Signature Scheme. Internet-Draft draft-irtf-cfrg-bbs-signatures-12, IRTF CFRG.

[26] Marqeta. 2026. Controlling Spending. Retrieved 2026-10-25 from https://www.marqeta.com/docs/developer-guides/controlling-spending

[27] Meta. 2024. Llama 3.3 Model Card. Retrieved 2026-10-25 from https://github.com/meta-llama/llama-models/blob/main/models/llama3_3/MODEL_CARD.md

[28] Omid Mirzamohammadi, Jan Bobolz, Mahdi Sedaghat, Emad Heydari Beni, Aysajan Abidin, Dave Singelée, and Bart Preneel. 2025. Keyed-Verification Anonymous Credentials with Highly Efficient Partial Disclosure. *IACR Commun. Cryptol.* 2, 3 (2025), 1–50.

[29] OpenAI. 2025. gpt-oss-120b & gpt-oss-20b Model Card. arXiv:2508.10925.

[30] Michele Orrù. 2025. Revisiting Keyed-Verification Anonymous Credentials. In *ACM CCS*. 1188–1199.

[31] Torben Pryds Pedersen. 1991. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *CRYPTO*. 129–140.

[32] Erik Reppel, Ronnie Caspers, Kevin Leffew, Danny Organ, Dan Kim, and Nemil Dalal. 2025. x402: An Open Standard for Internet-Native Payments. Retrieved 2026-10-25 from https://www.x402.org/x402-whitepaper.pdf

[33] Alfredo Rial and Ania M. Piotrowska. 2023. Compact and Divisible E-Cash with Threshold Issuance. *Proc. Priv. Enhancing Technol.* 2023, 4 (2023), 381–415.

[34] Samuel Schlesinger and Jonathan Katz. 2026. Anonymous Credit Tokens. Internet-Draft draft-schlesinger-cfrg-act-01, IETF.

[35] Claus-Peter Schnorr. 1991. Efficient Signature Generation by Smart Cards. *J. Cryptol.* 4, 3 (1991), 161–174.

[36] Alberto Sonnino, Mustafa Al-Bassam, Shehar Bano, Sarah Meiklejohn, and George Danezis. 2019. Coconut: Threshold Issuance Selective Disclosure Credentials with Applications to Distributed Ledgers. In *NDSS*.

[37] Vladimir Stantchev. 2026. Hardening x402: PII-Safe Agentic Payments via Pre-Execution Metadata Filtering. arXiv:2604.11430.

[38] Stripe. 2026. Issuing Spending Controls. Retrieved 2026-10-25 from https://docs.stripe.com/issuing/controls/spending-controls

[39] Stripe. 2026. Shared Payment Tokens. Retrieved 2026-10-25 from https://docs.stripe.com/agentic-commerce/concepts/shared-payment-tokens

[40] Isamu Teranishi, Jun Furukawa, and Kazue Sako. 2004. k-Times Anonymous Authentication (Extended Abstract). In *ASIACRYPT*. 308–322.

[41] Martin Thomson and Christopher A. Wood. 2024. Oblivious HTTP. RFC 9458, IETF.

[42] U.S. Securities and Exchange Commission. 2016. Chinese Traders Charged With Trading on Hacked Nonpublic Information Stolen From Two Law Firms. Press release 2016-280. Retrieved 2026-10-25 from https://www.sec.gov/newsroom/press-releases/2016-280

[43] Visa. 2025. Visa Introduces Trusted Agent Protocol: An Ecosystem-Led Framework for AI Commerce. Retrieved 2026-10-25 from https://usa.visa.com/about-visa/newsroom/press-releases.releaseId.21716.html

[44] Visa. 2026. Getting Started with Visa Transaction Controls. Retrieved 2026-10-25 from https://developer.visa.com/docs/vctc

[45] Zihan Wang, Rui Zhang, Yu Liu, Wenshu Fan, Wenbo Jiang, Qingchuan Zhao, Hongwei Li, and Guowen Xu. 2026. MPMA: Preference Manipulation Attack Against Model Context Protocol. In *AAAI*. 35838–35846.

[46] Jeff Weinstein and Steve Kaliski. 2026. Introducing the Machine Payments Protocol. Retrieved 2026-10-25 from https://stripe.com/blog/machine-payments-protocol

[47] An Yang et al. 2024. Qwen2.5 Technical Report. arXiv:2412.15115.

[48] An Yang et al. 2025. Qwen3 Technical Report. arXiv:2505.09388.

AI USE STATEMENT

OpenAI Codex and Anthropic Claude assisted with manuscript revision, figure layout, adaptation to the ACM LaTeX template, experiment scripts, and compilation and layout checks.

A SECURITY DEFINITIONS AND PROOFS

All theorems share one setting. Setup is honest and transparent (Figure 2): hash-to-group generators are random oracles programmable at setup, and epoch secrets are independent. Approvals and proofs bind the full parameter record, and state updates are linearizable. The privacy theorems assume fresh agent randomness per permit. We assume $p > K+1$, $1 \le \ell < p$, $0 \le B < 2^w$, and $nd_K + 2^w - 1 < p$,

with polynomial K and ℓ. Each bound is $\mathrm{negl}(\lambda)$ for $p \geq 2^{\lambda}$, polynomially many queries, and negligible primitive advantages.

A.1 KVPC Security

Definition A.1 (KVPC unforgeability). The adversary gets the public keys and the group and random oracles. Across epochs ν, it adaptively queries $O_{\mathsf{Issue}}(\nu, M, L; \theta, \zeta)$, which checks that θ is a policy of Section 3.2 with its canonical merchant polynomial and $(M, L) = \mathsf{KVPC.Commit}(\mathsf{ipk}_{\nu}, \theta; \zeta) \neq \perp$ (a *legal* commitment), returns a credential, and records the opening in Issued (the issuer sees only M); and $O_{\mathsf{Verify}}(\nu, \mathsf{ctx}, \tau)$, returning one bit. It wins, with probability $\mathsf{Adv}^{\mathsf{uf}}$, if an accepted presentation has no record in its epoch with the disclosed α and a policy accepting the (μ, a) of ctx; repeats of valid presentations do not count.

Definition A.2 (KVPC key-holder privacy). The adversary sees every key, coin, message, and record of the honest key holder. It adaptively supplies two issuance and presentation schedules, and the challenger runs schedule b for a uniform bit b. The schedules have equal leakage at every prefix: the public parameters; the issuance fields, shapes, timing, and epochs; the disclosed presentation fields (attributes, merchant, amount, linked point, and verifying epoch) and outcomes; and which serials are equal or retried. Every presentation uses an issued policy that accepts its call and an honestly opened linked point. Policies are fixed before their blindings, and calls before their presentation coins. Fresh serials are sampled honestly and named by permit labels. The holder's openings, ζ, ρ, κ, and proof masks stay private, though merchants may reveal linked openings. An exact retry resends the same bytes; a re-randomized attempt counts as fresh. The advantage is $\mathsf{Adv}^{\mathsf{kh}} = |\Pr[b' = b] - 1/2|$.

Concrete bound for Theorem 4.1. This restricted generic type-3 bilinear-group model with a random oracle exposes no ξG, no $\mathbb{G}_2$ image of a transparent base, and no secret multiple of a chosen point of unknown representation, and random-oracle inputs contain only handles that were already returned to the querying party. In Equation (5), Q_A counts the distinct presentation hash inputs that the adversary queries, Q_V the verification attempts on inputs it had not queried, and $N_{\mathsf{test}} \leq N_{\mathsf{lab}}(N_{\mathsf{lab}}-1)/2+Q_{\mathsf{eq}}$ the collisions among all N_{lab} group labels and the explicit equality tests over the whole history. *Tracing.* If each issued numerator is a legal commitment, outside the same events every accepted presentation has an earlier issued record (ν, k) with attributes α, accepting (μ, a), and a formal generic representation of P over G_d and G_η only.

Proof details for Theorem 4.1. G_0*: the interface and the game.* The issuance oracle applies $(\xi_\nu + e)^{-1}$ only to commitments whose openings it checks, and verification returns one bit, so the interface never scales a point of unknown representation by a secret. Let G_0 be the experiment of Definition A.1.

G_1*: issuance exponents.* G_1 samples every $e_{\nu,k}$ up front and aborts if $\xi_\nu + e_{\nu,k} = 0$ or two values collide in an epoch, at cost $N_I/p + \sum_\nu n_\nu(n_\nu - 1)/(2(p-1))$.

G_2*: formal evaluation.* G_2 keeps one global ledger of rational functions of the hidden (ξ_ν, s_ν, t_ν) with denominator $\mathsf{Den} = \prod_{\nu,k}(\xi_\nu + e_{\nu,k})$ and answers group operations and equality tests formally. Let $\mathsf{Bad}_{\mathsf{eval}}$ be the event that a label collision or equality test anywhere in the history holds at the sampled point but not formally. After clearing Den, every tested identity has degree at most $N_I + \max\{K, \ell\} + 2$, so by Schwartz–Zippel $\Pr[\mathsf{Bad}_{\mathsf{eval}}]$ is at most the N_{test} term of Equation (5), which also covers $s_\nu \in \mathcal{S}$.

G_3*: unqueried inputs and tracing.* G_3 rejects verification on Fiat–Shamir inputs that the adversary had not queried, at cost Q_V/p by deferred decisions. Let a queried input have two distinct accepting nonzero challenges. At its first query, every handle Z, including $\widehat{V}, W_1, W_2, P$ and the Schnorr first messages, has the form $Z = Z_0 + \sum_{\omega,k} z_{\omega,k} V_{\omega,k}$ with known scalars, where Z_0 lies in the known-scalar span of the transparent bases, $s^i_\omega G$, $t_\omega G$, $t^h_\omega H_P$, and fresh chosen handles; this holds by induction over group operations and issuance, since pairing has no map back to $\mathbb{G}_1$ and the interface supplies no other secret multiple. Subtracting the two transcripts gives scalars ρ, e, d, η with

$$(\xi_\nu+e)\widehat{V} = \rho H_\alpha + (s_\nu - \mu)W_1 + (t_\nu - j)W_2, \qquad P = dG_d + \eta G_\eta. \quad (13)$$

Let $v_{\omega,k}, w_{1,\omega,k}, w_{2,\omega,k}$ be the coefficients of $V_{\omega,k}$ in $\widehat{V}, W_1, W_2$. Each $M_{\omega,k}$ has H_0-coefficient one and each $V_{\omega,k}$ its own pole, so the residue of Equation (13) at $\xi_\omega = -e_{\omega,k}$ gives $\big((\xi_\nu + e)v_{\omega,k} - (s_\nu - \mu)w_{1,\omega,k} - (t_\nu - j)w_{2,\omega,k}\big)\big|_{\xi_\omega = -e_{\omega,k}} = 0$. For $\omega \neq \nu$, the independence of ξ_ν, s_ν, t_ν makes all three coefficients zero. For $\omega = \nu$, it gives $w_{1,\nu,k} = w_{2,\nu,k} = 0$ and $(e - e_{\nu,k})v_{\nu,k} = 0$, and distinct issuance exponents leave at most one source k. The monomials $\xi_\nu \widehat{V}_0$ then have no counterpart, so $\widehat{V}_0 = 0$ and $\widehat{V} = \varpi V_{\nu,k}$ with $\varpi \neq 0$, since $\widehat{V} \neq 0$. Reducing modulo $(s_\nu - \mu, t_\nu - j)$ leaves $\varpi(H_{\alpha_k} + f_k(\mu)G + \zeta_k Q_{r_k}(j)H_P) = \rho H_\alpha$, so independent bases force $\rho = \varpi, \alpha = \alpha_k$, and $f_k(\mu) = Q_{r_k}(j) = 0$ as $\zeta_k \neq 0$: the source policy accepts (μ, a), and P has the stated representation. This is the tracing statement. Every other input accepts at most one nonzero challenge, and a zero challenge is a separate event, so the adversary wins G_3 with probability at most $Q_A/(p-1) + Q_A/p$. Summing gives Equation (5). □

Concrete bound for Theorem 4.2. Under XDH in the random-oracle model, Equation (6) bounds the key-holder advantage and the gap between each execution and its simulation from the leakage. N, N_τ, and Q_H are counted over all epochs, excluding exact retries, and $\varepsilon_{\mathsf{XDH}}$ is the DDH gap for exponents in $\mathbb{Z}_p$ or $\mathbb{Z}_p^*$.

Proof details for Theorem 4.2. Hyb_1*: simulated proofs.* Hyb_1 samples χ and independent responses for each fresh presentation and programs the random oracle, which fails only on an input already defined, at cost $(N_\tau Q_H + N_\tau(N_\tau - 1)/2)/p$.

Hyb_2*: excluded blindings.* Hyb_2 replaces the prover's resampling of excluded blindings with a bad event, at cost $2N(K+2)/p$.

Hyb_3*: commitments.* In $\mathrm{Hyb}_{3,i}$, $i \leq N$, the reduction $\mathcal{B}_i$ receives $(G, \widetilde{G} = \delta G, E = yG, Z)$, sets $G_\zeta = \widetilde{G}$ and $H_P = kG$ for $k \leftarrow\$\ \mathbb{Z}_p^*$, and outputs $M = H_\alpha + f(s_\nu)G + Q_r(t_\nu)E$ and $L = k^{-1}Z + d_r G_c$. For $Z = \delta y G$ this is the real commitment with $\zeta = y/k$; for uniform Z, L is independent of M. $\mathcal{B}_i$ needs neither y nor ζ: later presentations of this permit use $(\rho V, \rho(M - H_\alpha))$ with a known ρ, and proofs are simulated. After these hops each M_i is uniform, since $Q_{r_i}(t_\nu) \neq 0$, so we generate it as $M_i = \varpi_i G$ with a private ϖ_i.

Hyb_4*: presentations.* In $\mathrm{Hyb}_{4,m}$, $m \leq N_\tau$, $\mathcal{B}'_m$ sets $H_0 = \widetilde{G}$ and the other attribute bases as known multiples of G, so $H_\alpha = \widetilde{G} + vG$ for a computable v, and outputs $\widehat{V} = \varpi_i(\xi_\nu + e_i)^{-1}E$ and $F = (\varpi_i - v)E - Z$. A Diffie–Hellman tuple gives the real presentation with $\rho = y$,

Public input. ep, pp, $\mathsf{ipk}_{\mathsf{ep}}$, $\mathbf{D}$, $\mathsf{mandate}^{\mathsf{pub}} = (B, \mathsf{expiry})$, σ_{U}, and slack commitments $\Lambda_0, \ldots, \Lambda_{w-1}$ for a public bit width w. Put $H_{\mathsf{pub}} = H_0 + \mathsf{expiry}\, H_T$, $K_h = Q_h(t)H_P$, and $f_k(X) = \sum_{i=0}^{\ell} f_{k,i} X^i$.

1. **Permit rows.** For each permit k, an OR proof over the ranks $h \in \{0, \ldots, K\}$, with branch challenges $c_{k,h}$, and a compilation row share the branch responses $z_{k,h}$:

$$R^L_{k,h} = z_{k,h} G_\zeta - c_{k,h}(L_k - d_h G_c), \qquad \textstyle\sum_h c_{k,h} = \chi, \tag{14}$$
$$R^M_k = z_{k,u} H_u + z_{k,\gamma} H_\gamma + \textstyle\sum_i z_{k,f_i} S_i + \sum_h z_{k,h} K_h - \chi(M_k - H_{\mathsf{pub}}).$$

   For the true rank $r = r_k$, the prover samples every other branch's $(c_{k,h}, z_{k,h})$ and masks $\tau_k, \tau_{k,v} \leftarrow\$\ \mathbb{Z}_p$, puts $c_F = \sum_{h \neq r} c_{k,h}$, uses $R^L_{k,r} = \tau_k G_\zeta$, and places $(\tau_k - c_F \zeta_k) K_r + \sum_{h \neq r} z_{k,h} K_h$ in R^M_k. After χ, it sets $c_{k,r} = \chi - c_F$, $z_{k,r} = \tau_k + c_{k,r}\zeta_k$, and $z_{k,v} = \tau_{k,v} + \chi v_k$ for $v \in \{u, \gamma, f_0, \ldots, f_\ell\}$.

2. **Budget row.** For $\delta = B - \sum_k d_{r_k} = \sum_{h<w} 2^h \beta_h$, the prover commits $\Lambda_h = \beta_h G_c + \omega_h G_\zeta$ with $\omega_h \leftarrow\$\ \mathbb{Z}_p$, proves $\beta_h \in \{0, 1\}$ with two-branch OR proofs, and proves $\sum_k L_k - B G_c + \sum_h 2^h \Lambda_h = \eta_B G_\zeta$ for $\eta_B = \sum_k \zeta_k + \sum_h 2^h \omega_h$; with $B < 2^w$ and $n d_K + 2^w - 1 < p$, the budget bound holds over the integers.

3. **Output.** π^{iss} is $\chi = \mathsf{H}(\text{public input, first messages } R^L_{k,h}, R^M_k, \ldots)$ with all challenges and responses; $\mathsf{W}_{\mathsf{iss}}$ recomputes the first messages and checks χ.

Figure 5: The joint issuance proof π^{iss} of Equation (4).

where $F = (s_\nu - \mu)W_1 + (t_\nu - j)W_2$ is the point that KVPC.Verify reconstructs; a random Z removes the source. $\mathcal{B}'_m$ samples W_1 uniformly and sets $W_2 = (t_\nu - j)^{-1}(F - (s_\nu - \mu)W_1)$, the exact distribution under the fresh kernel mask κ. Both reductions know every epoch secret, $\mathcal{B}'_m$ also knows every ϖ_i, and $\mathcal{B}_i$ needs no ϖ of its target, so neither guesses, and each of the $N+N_\tau$ hops costs $\varepsilon_{\mathsf{XDH}}$.

Hyb_5*: serials.* Hyb_5 samples fresh serials lazily along the leaked equality pattern, at cost $N(N-1)/(2(p-1))$, and depends only on the leakage. If a distinguisher outputs 1 with probability p_b on schedule b and p_S on Hyb_5, then $\mathsf{Adv}^{\mathsf{kh}} = \frac{1}{2}|p_1 - p_0| \leq \max_b |p_b - p_S|$, and summing the hop costs gives Equation (6). □

A.2 MinMandate Security

Views and leakage. In this appendix, the leakage of Definition 4.4 also includes, per interaction x, an event record δ^O_x for each observer O, and the wallet's funding records. The event record δ^O_x lists what observer O sees of interaction x: message occurrences and encoded lengths, local results (accept, reject, or a cached answer with the request it repeats), the order and times of these events, settlement events, and references to reused messages. Settlement pays the payee from the pooled reserve and shows the merchant its amount, payee, time, status, and references, but no funding account or approval. Message lengths depend only on pub. Commitments, openings, and signatures are protocol objects; in the views of Equations (10) and (11), each appears only at the stages that δ^O_x records.

Definition A.3 (Authorization soundness). The adversary controls the agent and any admitted merchants, with their keys, and has four honest-party oracles. *Approval* $O_{\mathsf{Appr}}(\mathsf{mandate}, (\theta_k, \zeta_k)_{k\in[n]})$ checks each policy ($1 \leq |\mathcal{S}_k| \leq \ell$, distinct canonical encodings, and the complete monic-product vector, or the zero vector for adm) and that the canonical openings of the ordered points match the reviewed mandate and parameter record, returns σ_{U}, and stores $(j, \mathsf{ep}, \mathbf{D}_j, \mathsf{mandate}^{\mathsf{pub}}_j, (\theta_{j,k}, \zeta_{j,k})_k)$. *Issuance* O_{Iss} (lines 5–8 of Figure 3) accepts a stored approval with a valid π^{iss}, reserves B_j, returns the credentials, and appends $(j, k, \mathsf{ep}, \alpha_{j,k}, \theta_{j,k}, \mathsf{cred}_k)_{k\in[n]}$ to Issued. *Merchant* O_{Mer} (lines 12–13) acknowledges (C, p) only after the checks in Section 3.4 and logs the pair. *Redemption* O_{Red} (lines 15–18) uses the atomic transition of Section 3.4, returns accept or reject, and sends an accepted ticket to the payee. A redemption $(\mathsf{ep}, \alpha, \mu, a)$ *matches* a record of Issued with epoch ep and attributes α whose policy $\theta_{j,k}$ accepts (μ, a). The adversary wins if, at any time, the fresh accepted redemptions admit no assignment to matching records that gives distinct redemptions distinct serials and assigns payments totaling at most B_j to each batch j; an accepted redemption's payment fields differ from the projection of the quote committed in C; it carries an honest merchant's acknowledgement that O_{Mer} never issued for (C, p); a fresh acceptance repeats one (μ, nonce); or a ticket bypasses the atomic transition. $\mathsf{Adv}^{\mathsf{auth}}$ is the winning probability.

Definition A.4 (Unlinkability against merchants). With honest user, agent, and wallet, colluding merchants see the events in their records δ^{M} and run non-challenge sessions. They submit two workflows $\mathcal{P}_0, \mathcal{P}_1$ with equal $\mathcal{L}_{\mathsf{M}}$ at every prefix, under the rules of Definition A.2, receive $\mathsf{View}_{\mathsf{M}}(\mathcal{P}_b)$ for a uniform bit b, and output b'; $\mathsf{Adv}^{\mathsf{M}} = |\Pr[b' = b] - \frac{1}{2}|$.

Concrete bound for Theorem 4.3. Let Q_I bound the distinct issuance-proof preimages that any party evaluates. The counts in Equation (5) cover the whole protocol, including outer signatures, issuance proofs, pairings, the agent's credential checks, and wallet-internal points. Then Equation (7) holds with $\varepsilon_{\mathsf{outer}} = \sum_\iota n_\iota \mathsf{Adv}^{\mathsf{euf}}_{\mathcal{B}_\iota} + \mathsf{Adv}^{\mathsf{cr}}_{\mathsf{H},\mathcal{B}_{\mathsf{H}}}$, for efficient forgers $\mathcal{B}_\iota$ and a collision finder $\mathcal{B}_{\mathsf{H}}$, where ι ranges over approval, certificate, acknowledgement, and ticket signatures and n_ι counts the honest keys of type ι.

Proof of Theorem 4.3. Let G_0 be the experiment of Definition A.3. We reason outside the global event $\mathsf{Bad}_{\mathsf{eval}}$ of Appendix A.1 and charge it once at the end. G_1*: outer signatures.* G_1 aborts on a valid signature under an honest key on a message never submitted for signing under that key, or a collision of H on request digests. All signatures are Schnorr signatures on type-labelled messages, and only certificates and tickets may share a key. $\mathcal{B}_\iota$ guesses one of the n_ι keys, embeds its challenge key as a $\mathbb{G}_1$ label, generates everything else, answers every query to the challenge key with its signing oracle, and forwards all random-oracle queries. Programming collisions with proof inputs lie inside its advantage, so $|\Pr[\mathrm{G}_0 = 1] - \Pr[\mathrm{G}_1 = 1]| \leq \varepsilon_{\mathsf{outer}}$. Signature handles $R = zG - cP_{\mathsf{sig}}$ add no epoch-secret pole and enter N_{test}.

G_2*: issuance.* G_2 aborts if the wallet accepts a batch input with at most one accepting challenge; the first messages of π^{iss} are computed from public handles and all rows share one challenge, so this costs Q_I/p. For an accepted batch, two accepting challenges in the cap rows of Figure 5 give $\Delta z_h G_\zeta = \Delta c_h (L - d_h G_c)$. As G_ζ, G_c are independent, at most one branch r changes and $\sum_h \Delta c_h = \Delta\chi \neq 0$, so

$$\Delta c_r = \Delta\chi, \qquad \Delta z_r = \zeta \Delta\chi, \qquad L = \zeta G_\zeta + d_r G_c. \tag{15}$$

Issuance rejects $M = 0$ and $L \in \{d_h G_c\}_{h \le K}$, so $\zeta \neq 0$. The compilation row reuses the z_h, and subtraction gives

$$M - H_{\text{pub}} = uH_u + \gamma H_\gamma + \textstyle\sum_{i=0}^{\ell} f_i S_i + \zeta K_r, \tag{16}$$

with $u = \Delta z_u / \Delta\chi$ and likewise for the other coefficients. The response bases are denominator-free, so M has no credential component and shares the rank and blinding of L. The bit ORs give $\Lambda_h = \beta_h G_c + \omega_h G_\zeta$ with $\beta_h \in \{0, 1\}$, and the G_c coefficient of the budget row gives $\sum_k d_{r_k} + \sum_h 2^h \beta_h \equiv B \pmod p$, an integer equality under the ranges of the preamble, so $\sum_k d_{r_k} \le B$.

The approval signature binds the checked ordered points. Comparing openings fixes (ζ, r), attributes, and expiry, while the formal identity $\sum_{i=0}^{\ell} (f_i - f_{k,i}) s^i = 0$ fixes the approved coefficient vector. After batch verification, the wallet reserves the signed budget against the approval key's account, samples e_k, and issues. Sessions linearize there, and retries repeat neither reservation nor issuance. By induction, each credential has a legal numerator, a recorded opening and approval index, and an earlier reservation, so the wallet acts as O_{Issue} of Definition A.1: the game supplies (θ_k, ζ_k) from the approval record, and the issuer sees only M_k.

G_3*: calls.* G_3 aborts if an accepted redemption violates the tracing statement of Appendix A.1. The issuance step establishes its hypothesis, and the outer and proof handles are known-scalar combinations of public points and credentials, so each accepted call traces to an issued permit with matching serial, class, and expiry whose policy accepts (μ_i, a_i). For $P_i = C_i - \text{Com}(0, q_i; 0)$, two accepting challenges of the linked row give $P_i = (\Delta z_d / \Delta\chi) G_d + (\Delta z_\eta / \Delta\chi) G_\eta$, an opening of C_i whose quote part is fixed by p_i and the class. The linked and authentication rows share one challenge, so either failure falls in the $Q_A/(p-1) + Q_A/p$ event for a queried input and in the Q_V/p event otherwise. The hop costs the terms of Equation (5) with global counts, including Bad_{eval} once.

An honest merchant acknowledges (C_i, p_i) only after opening C_i to its request digest and quote. In G_3, commitment binding equates this opening with the extracted one. A second opening falls in Bad_{eval}, and a forged acknowledgement or digest collision falls in G_1. Expiry, class, and atomic serial and quote checks give the remaining claims and $\sum_i a_i \le \sum_k d_{r_k} \le B$. A ticket is emitted only by the atomic transition that records the serial, quote, debit, and ticket; rejected attempts change no state, and exact retries return cached results, so the invariant survives failures and concurrency. The three hop costs give Equation (7). □

Definition A.5 (Unlinkability against the wallet). This extends Definition A.2 to honest user, agent, and wallet executions. The observer receives all epoch keys, wallet coins, messages, issued objects, parameter and funding records, tickets, caches, and logs. It may collude with merchants, obtain their request openings, and run non-challenge sessions; the user and agent keep their openings, blindings, randomizers, and proof masks private. The two workflows may arrive online, obey the rules of Definition A.2, and have equal $\mathcal{L}_{\mathsf{W}}$ at every prefix. They may differ in unused policies and in the assignment of calls to compatible permits; outcomes of background traffic and merchant deviations stay in the leakage. The experiment generates serials and derived fields and returns the view of workflow b for a uniform bit b; $\text{Adv}^{\mathsf{W}} = |\Pr[b' = b] - \frac{1}{2}|$.

Concrete bound for Theorems 4.5 and 4.6. Take the assumptions, N, N_τ, and ε_{XDH} of Appendix A.1. N_π counts the new issuance and call proofs over all epochs (one per batch or fresh presentation, none for exact retries), and Q_H all other relevant observer, wallet-internal, and outer-signing oracle queries. Each workflow has a simulation from $\mathcal{L}_{\mathsf{W}}$ whose gap is at most the right side of Equation (12).

Proof of Theorems 4.5 and 4.6. Hyb_1*: simulated proofs.* The simulator generates the honest outer keys, signs each hybrid's objects, and samples the wallet view of Definition A.5, keeping user signing keys and approval openings private. Hyb_1 simulates every fresh issuance and call proof. For issuance, it samples χ and responses uniformly, subject to each OR's branch challenges summing to χ, and computes the first messages from Figure 5. The real responses are bijective translations of independent masks, including all $\ell + 1$ coefficient masks; shared responses keep the correlation between the cap and compilation rows, and bit ORs are simulated alike. Given the statement, χ, and the other responses, the budget row's independent response for η_B makes its first message uniform (Figure 5). With the call-proof simulation of Appendix A.1, each programmed input hits an earlier one with probability at most $1/p$; on a hit the programming fails and the earlier answer stands. The cost $\varepsilon_{\text{FS}} = (N_\pi Q_H + N_\pi (N_\pi - 1)/2)/p$ replaces the primitive's term.

Hyb_2–Hyb_4*: domain, commitments, presentations.* Hyb_2 to Hyb_4 apply the domain, commitment, and presentation hops of Appendix A.1 throughout the interleaved history, with simulated proofs and no further programming. Target approvals and issuer responses need neither y nor ζ, later presentations reuse a known ρ, and uniform slack commitments with simulated proofs allow $M_i = \varpi_i G$ with a private ϖ_i. Merchants that know (d_i, η_i) learn no authentication masks, the numbers of merchant-polynomial powers and coefficient masks depend only on ℓ, and other message sizes are public.

Hyb_5*: serials and derived fields.* Hyb_5 applies the serial hop and generates identifiers, derived fields, cached bytes, and signing coins only from the event records of $\mathcal{L}_{\mathsf{W}}$. Each hop costs as in Equation (6). Comparing each workflow with Hyb_5, as at the end of the proof for Theorem 4.2, yields Equation (12).

Colluding merchants. Couple the parameters, wallet keys, adversary coins, and randomness. At a common prefix, equal $\mathcal{L}_{\mathsf{M}}$ fixes the request, quote, epoch, and outcome, so service views, acknowledgements, and tickets agree; reuses replay cached bytes at their recorded stages, and hidden permits and funding records yield no external object. By induction the views agree and $\text{Adv}^{\mathsf{M}} = 0$. □

B EXTENSIONS

Returning unused budget. The wallet reserves B but debits only the paid amounts, and returning the rest per approval would link the approval to its payments. Anonymous change, as in Anonymous Credit Tokens [34], avoids this link: the agent keeps one change credential whose hidden balance grows through unlinkable updates. Issuance adds the slack $B - \sum_k c_k$, the bit commitments of π^{iss} in Figure 5 already authenticate it; each redemption adds $c_{k_i} - a_i$; after expiry, each unused permit adds its cap through a presentation at the smallest denomination that pays no merchant. A serial is consumed by either its payment or its recovery, so no cap is credited twice, and at withdrawal the wallet learns only the account's unused budget. One further proof binds each increment to the presented

permit's hidden cap; since KVPC.Verify already enforces $a_i \le c_{k_i}$, the increment needs no range proof.

Conditional key release. In the base protocol, the merchant is paid before it returns its result, in line 19 of Figure 3. The wallet can instead escrow the payment, like a hash-time-locked contract. The merchant sends the agent its result encrypted under a fresh key dk_i and signs $h_i = \mathsf{H}(\mathsf{dk}_i)$ into ack_i, and redemption places a hold. The merchant then reveals dk_i to the wallet, which pays and forwards the key if it matches h_i before the quote's payment deadline dl_i, and otherwise releases the hold. The hold consumes the serial like any redemption, so a cancelled call never reuses the serial, and the held amount returns through the change credential above. The wallet also learns each hold's outcome and timing; h_i and dk_i are random for an honest merchant and already known to a colluding one. The hold guarantees the key, not a correct result; a predicate on the result needs a zero-knowledge contingent payment [5].