

Deterministic Rails for Probabilistic AI Agents in Payments

The frameworks the industry is building, what they miss, and a hybrid model for agent key custody and delegated authority, built from the card industry's own playbook.

Rasagnya Ryali, October 2026

Contents

Introduction	2
Executive summary	2
The core tension: probabilistic agents, deterministic payments.....	3
Anatomy of an agent-initiated card transaction.....	3
The determinism map: where certainty is non-negotiable	5
Visa's Trusted Agent Protocol: the identity and integrity checkpoints	5
Mastercard's Agentic Tokens: credential, consent, and revocation.....	6
Google's AP2: the commitment checkpoint	7
The composition: three signatures, one request.....	7
What the frameworks leave open	8
What the identity world is building, and what payments still needs	8
Borrowing the payments playbook.....	9
A hybrid model: custody and authority	10
A DUKPT-style serial for agents.....	11
Putting the serial under attack	12
How the whole model holds up	13
A day in the life of an agent purchase	15
Open questions	15
Recommendations	16
References	16

Introduction

For as long as card payments have existed, a human has been at the keyboard or the terminal, tapping a card, typing a PIN, clicking buy-now. That assumption is ending. A new class of AI agents is being built to shop, negotiate, and pay on our behalf. Your AI assistant will reorder groceries when the fridge runs low, a travel agent will book the whole trip at the moment prices dip, and a procurement AI-bot will restock a warehouse without anyone signing off on each order.

Visa, Mastercard, Google, and others are already shipping the rails for commerce that will happen at machine speed, with no human in the loop for every purchase. But this raises a question the payments industry has never really had to answer before. When the thing initiating a payment isn't a person, how do you prove it was actually authorized to spend your money?

This paper answers that question by proposing a concrete way to close the gap, drawing on the key-management playbook the card industry has trusted for decades.

Executive summary

Payments has always run on determinism. An AI agent, by design, does not. That is the tension at the heart of this paper. If agents are going to act for us and spend on our behalf, the safe way to put one into a card transaction is not to make the agent more reliable. It is to find the handful of points where a probabilistic system must not be allowed to decide, and to lock them down with the controls the card networks already give you.

That determinism is what makes a card payment defensible after the fact. The same inputs must produce the same auditable, non-repudiable outcome, because every dispute, chargeback, and audit rests on proving exactly who authorized what. Put a probabilistic actor in the middle of that and the proof is the first thing to go, unless every commitment it makes is pinned down and signed. This paper does three things:

- It offers a mental model for the problem, one that stops chasing agent reliability and maps where certainty is non-negotiable instead.
- It walks an agent-initiated card transaction stage by stage, showing where an agent may improvise and where it must not, and how Visa's Trusted Agent Protocol, Mastercard's Agentic Tokens, and Google's AP2 mandates each lock a different checkpoint, as complementary layers rather than competitors. Then it steps back to what they still leave unsolved.
- And it proposes a hybrid model for the checkpoint the industry keeps underweighting. The gap this paper sets out to fill is where an agent's signing key lives and what it is allowed to do. Each half already has a proven answer, DUKPT-style custody that keeps the key in the HSM and never in the agent, and per-hop delegation that narrows authority at every step. The first comes from the card

industry, the second from the identity world, and this paper shows how to bind them together for agents.

Two limits to state up front. The security claims here are properties of the design, checkable by reasoning rather than measured on a running system. And the model secures keys and authority, not the agent's judgment. A purchase that is correctly scoped and correctly signed, for something the agent was allowed to buy, is still valid even when it is the wrong call, and nothing here prevents that.

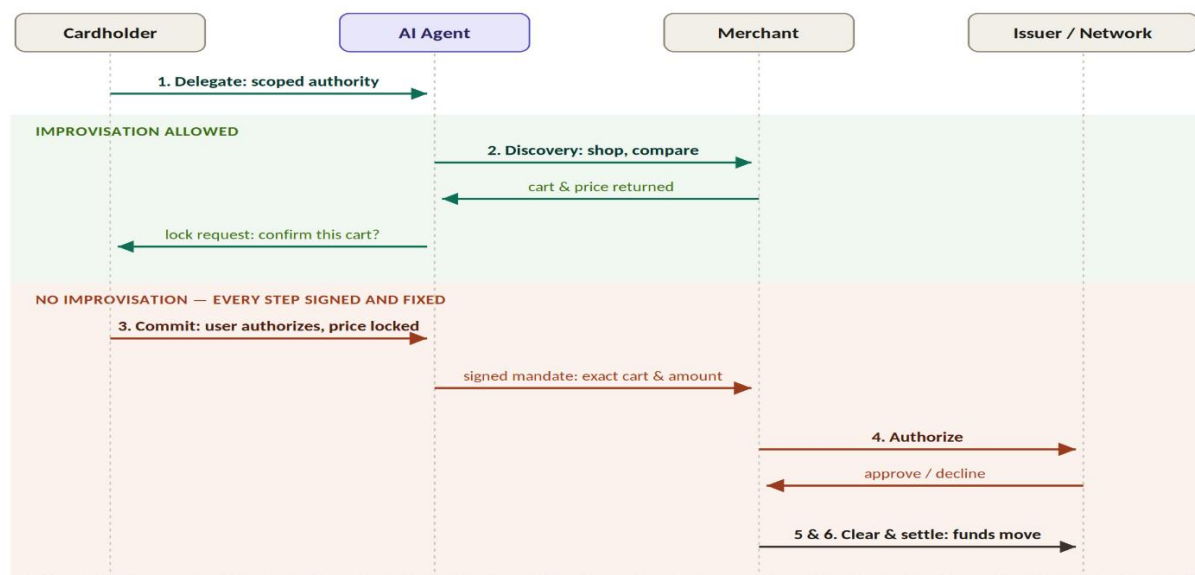
The core tension: probabilistic agents, deterministic payments

Determinism in payments is not a design choice, it is an operating requirement, wired into how the system settles disputes and proves consent. An authorization either maps to a specific, provable consent or it is reversible as unauthorized. There is no third option, and that is what a probabilistic agent collides with the moment it initiates a payment.

The IMF, in its April 2026 note on agentic payments, reaches the same conclusion this paper builds on. Authorization must be governed by deterministic policy rules, even when the intent feeding them came from a probabilistic system [\[1\]](#). The agent proposes structured intent, and that intent still has to clear a deterministic authorization layer before any money moves. The goal was never a perfectly obedient agent. The agent proposes, and signed controls dispose.

Anatomy of an agent-initiated card transaction

An agent-initiated purchase bolts three new stages onto the front of the familiar authorize, clear, settle flow. Same cast, cardholder, merchant, acquirer, network, and issuer, except a non-human initiator now sits between the cardholder and the merchant. The back half of the picture below is just how every card payment already settles. The front half is newer, and no single network has published it as a standard yet. It is the shared shape beneath the emerging frameworks, drawn together here as the reference the rest of the paper measures against and traced end to end.



In step 1, delegation: the cardholder grants the agent scoped authority, a spending mandate with limits rather than a blank cheque.

In step 2, discovery: the agent shops on the cardholder's behalf, comparing options and assembling a cart that the merchant then prices. This is the improvisation-allowed zone, the only part of the flow where the agent is meant to range freely. Everything after it must stay in the signed-and-fixed zone.

In step 3, commitment: this specific cart at this specific price is locked by signature. With the cardholder present, they approve the cart. In an autonomous run, the agent commits on its own, but only within the mandate the cardholder signed in step 1, and the merchant's signature on the cart holds the price. Either way, from that point the amount, the merchant, and the cart are fixed, and the signed mandate carrying the exact cart and amount goes to the merchant.

In step 4, authorization: the merchant sends the transaction to the issuer or network, which approves or declines it.

In steps 5 and 6, clearing and settlement: the funds actually move, exactly as they would for any card payment.

Underneath all of it sits the recovery path. If something goes wrong, the transaction can be attributed to a specific agent, reversed, or the agent's authority revoked, but only because every step from commitment onward was signed. Take those signatures away and there is nothing to attribute or reverse. That is why the boundary matters. The agent improvises in discovery, and from commitment on, every step is signed rather than inferred.

The determinism map: where certainty is non-negotiable

This section makes determinism concrete. The map takes the flow stage by stage and names exactly what an agent may decide and what must be fixed and signed before money moves. The ‘Must be fixed and verifiable’ column is the benchmark for every framework that follows. Each one either locks a row of it or leaves it open.

Table 1

Stage	Probabilistic is fine	Must be fixed and verifiable	Failure if it isn't
Delegation	Which tasks to accept	Scope, spend cap, merchant and category limits, expiry	Agent acts beyond its authority
Discovery	Which products and options to surface	Nothing binding yet	None
Commitment	Which cart to propose	Authority for this cart, traceable to the cardholder's mandate	Price or amount drift
Authorization	Nothing	Agent identity, and that the request maps to the cardholder's authorized consent	Can't tell a real agent from a bot, no attribution
Clearing and settlement	Nothing	Amount, credential, and merchant all match the signed commitment	Mismatch, misrouting, credential exposure
Disputes and revocation	Nothing	Who authorized, which agent acted, ability to revoke one agent	Can't attribute or reverse cleanly

The agent's reasoning remains probabilistic, but every commitment it emits, identity, scope, price, amount, credential, consent, must become a signed, replayable fact before it counts. The next few sections walk the frameworks that sign each one.

Visa's Trusted Agent Protocol: the identity and integrity checkpoints

When an agent arrives to pay, the merchant is the one that must decide whether to trust it, and it has to decide before the transaction goes through. That is a check the web never had to make. Today's tools only tell a merchant whether a request comes from a genuine source, or whether it is a bot or a human. Whether a specific, credentialed agent is the one behind the request is a new question, and it is the one Visa's Trusted Agent Protocol (TAP) answers.

Instead of inferring what the agent is from signals that can be faked, TAP has the agent sign its identity directly into the request, and the merchant checks that signature against a credential Visa vouches for [2]. The question stops being a judgment call and becomes a cryptographic yes or no.

It locks two things:

- Identity (who the agent is) and
- Integrity (that the request wasn't tampered with or replayed).

Identity: The agent signs each request at the HTTP layer using HTTP Message Signatures (RFC 9421) [6], building on the emerging Web Bot Auth standard. The merchant checks that Ed25519 signature against a Visa-operated directory of agent public keys. A signed tag also marks whether the request is to browse or

to pay, so a request to browse cannot be reused to pay, and an optional signed token can name the human behind the agent.

Integrity: Each request is cryptographically pinned to the merchant's domain and the exact operation, with a timestamp, session ID, and key ID. A signature captured in one place can't be reused in another.

Against the determinism map ([Table 1](#)), TAP locks the identity and integrity parts of the Authorization row. Agent identity becomes a deterministic yes or no, the operation is fixed, and the request is tamper-evident. What TAP doesn't do is move money or bind the user's authorization of a specific cart and amount. That's the next layer's job.

Mastercard's Agentic Tokens: credential, consent, and revocation

Where TAP settles who the agent is, Mastercard's Agentic Tokens settle what its credential can do, who may use it, for what, and up to how much. An Agentic Token extends the Mastercard Digital Enablement Service so a tokenized card credential is bound to a specific agent, a specific merchant scope, and a specific consent policy, which means a model can check out without ever touching the raw card number [\[3\]](#).

It locks three things:

- the credential itself,
- the consent attached to it, and
- the ability to revoke it.

Credential and consent: Spend caps, merchant restrictions, and expiration windows are set when authorization is granted, and the network enforces them. None of it depends on the agent behaving. Because the agent's identity is baked into the token, the network can tell an agent-driven charge from a user-driven one and pin a dispute on a specific agent.

Revocation: Pull an agent's authorization and its token is invalidated at the network, so the next attempt fails at authorization rather than after the money is gone, and you can revoke that one agent without killing the card. Strip the agent identifier out and none of this works.

Against the determinism map ([Table 1](#)), Agentic Tokens cover the Delegation limits and the whole Disputes and revocation row (funding-credential scope, spend limits, expiry, per-agent attribution, and revocation).

Two caveats:

- First, the naming invites confusion. Mastercard's Agent Pay with Agentic Tokens is a different thing from its later Agent Pay for Machines work (AP4M), and the two shouldn't be conflated.

- Second, liability is still an open edge. As of this writing, the networks have said far more about how an agent authenticates than about who absorbs the loss when an authenticated agent gets it wrong.

Google's AP2: the commitment checkpoint

Google's AP2 supplies the signed record that ties this exact cart at this exact price back to the user's authorization. That's the one checkpoint that neither network scheme covers on its own, which is why both compose with AP2 rather than compete with it [4]. It locks the commitment, the moment a purchase is pinned to a specific, authorized cart.

AP2 does this with three signed mandates written as W3C Verifiable Credentials [7].

- The Intent Mandate is the user handing over authority within limits.
- The Cart Mandate is approval of a specific cart at a specific price.
- The Payment Mandate is the credential the network actually sees.

Together they are a non-repudiable record of who authorized what, and within what bounds. The clever part is a double signature. The merchant signs the cart, which guarantees the price. The user signs the payment, which authorizes it. The merchant can't overcharge, and the agent can't spend past what it was given. For delayed, human-not-present flows, an Intent Mandate with a time-to-live caps what the agent can do on its own later.

Against the determinism map (Table 1), AP2 owns the Commitment row, the exact cart, price, and amount, and the authorization that ties them back to the user's mandate. It is the layer that connects the other two.

The composition: three signatures, one request

TAP proves the agent, Agentic Tokens scope the credential, and AP2 records the consent that ties both to one specific purchase. Seen together, the three are layers, each signing a different fact inside the same request.

Table 2

Layer	What it signs	Checkpoint it locks	Built on
Visa's TAP	Agent identity and operation, in the HTTP headers	Is this a legitimate agent, for this action	RFC 9421 / Web Bot Auth / Ed25519
Google's AP2	Authorization for the cart, tied to the user, in the body	Is this purchase authorized by the user	W3C Verifiable Credentials / DIDs
Mastercard's Agentic Tokens	The funding credential's scope and limits	What may pay, for whom, up to how much	MDES tokenization (Agentic Tokens)

Three signatures, three facts, all verified before a cent moves. A shared protocol layer such as ACP lets them present one clean interface to the merchant even when the credential leg differs from one network to the next. That stack is the deterministic cage around a probabilistic agent. The agent gets to choose what goes into the request. It doesn't get to forge who it is, what it was authorized to do, or what its credential will allow.

What the frameworks leave open

The three frameworks lock their checkpoints well, but each stops at the same edge. They say how an agent signs and how a verifier checks that signature. What none of them pins down is where the signing key lives, how the agent came to hold it, how it is rotated or revoked across a fleet, or what happens when the agent is compromised. They secure the message and leave the key's custody and lifecycle to whoever builds the agent. The networks may specify more of this in later revisions. Even when they do, the discipline of keeping a signing key in hardware and out of the exposed endpoint has to come from somewhere, regardless of who writes it into a spec, and that is what the rest of this paper supplies. That is a key-management problem at its core, and it hides a handful of assumptions and open edges worth pulling into view.

- **Key custody:** Where do these signing keys actually sit, and how are they protected? The Cloud Security Alliance's AP2 threat model points at hardware-backed key management as the mitigation, but leaves the operational reality, provisioning, custody, rotation, and revocation at fleet scale, wide open [5].
- **A valid grant can still go bad:** Authorization checks a transaction at submission but assumes it was generated in good faith. A compromised agent, through prompt injection or a stolen signing session, can produce transactions that satisfy every signed policy, and because delegated authority is usually standing rather than reissued each time, a grant that was fine when issued can turn unsafe before anyone revokes it. The lag between compromise and cutoff is itself a security parameter.
- **PCI and SCA scope:** How agent tokens and mandates interact with Strong Customer Authentication and 3-D Secure, and where PCI scope falls for agent key custody, remain unsettled [11].

What the identity world is building, and what payments still needs

So, who is working on that key-management problem? The identity world is, but from a different direction. SPIFFE and SPIRE hand out short-lived, attestation-bound workload identities, a run of 2026 IETF drafts extends that model to agents with per-hop delegation and scope attenuation [9][12][13], and a cluster of products already tackles agent key custody directly, using multi-party computation and scoped session credentials so the agent's code never holds the key [14][15]. It is solid work, and payments will borrow the identity and delegation ideas from it.

But it was built for a different kind of failure. Cloud identity assumes the worst case is a leaked token, and a leaked token is something you recover from. You detect it, revoke it, issue a new one, and move on. That works when a stolen credential only lets an attacker read data or call a service, because the next token shuts them out and the damage stops there.

A card payment has no such undo. A signed instruction to move money is a one-way door and once it clears, rotating the key afterward changes nothing, because the money is already gone. A model whose whole safety story is cleaning up after a breach doesn't fit a world where the breach is the irreversible thing.

There is a deeper problem in the identity model most of this work builds on. It hands the workload its credential to hold and use, which for a payment-signing agent means putting the signing key in the agent's own runtime, the one place most exposed to prompt injection and compromise. The MPC custody products are the partial exception, they do keep the raw key out of the agent, but they are built for crypto wallets rather than the card rails, so they borrow the wallet model instead of the containment payments already trusts. Payments worked this out long ago and does the opposite. The key never sits with the exposed endpoint. It stays in hardware, and the endpoint asks it to sign. That is the guarantee the cloud tools don't give you, and the next section rebuilds it for agents.

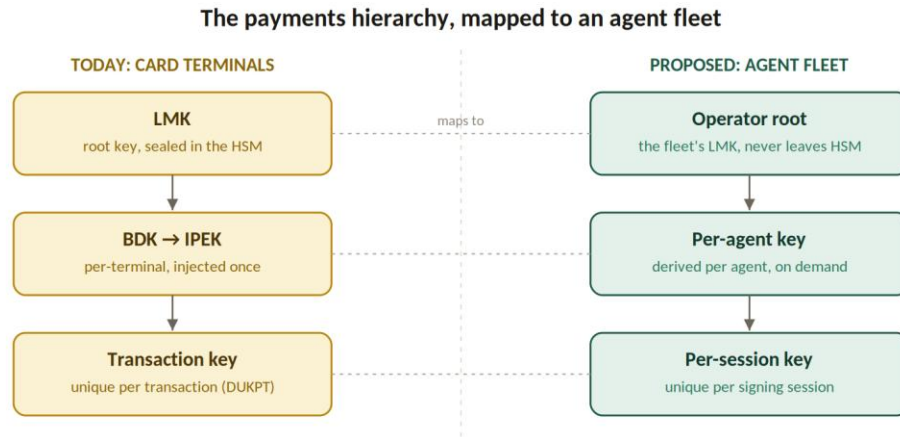
Borrowing the payments playbook

That blueprint is the payments key hierarchy, and it has run in production for decades, in every card terminal and ATM. Behind the simple rule from the last section, keep the key in hardware and let the endpoint ask it to sign, sit two mechanisms an agent fleet can borrow directly. One handles custody. The other handles containment.

The first is the HSM key hierarchy, which handles custody. A root key, the LMK, is sealed in hardware, and every working key below it is wrapped under that root. The agent, the terminal, whatever sits at the edge never handles the sensitive key itself. It asks the hardware to do the cryptography.

The second is DUKPT, which handles containment. Compromise one terminal and the damage stays contained to that terminal. The base key and every other terminal are untouched, and because each transaction uses a key that is derived and then discarded, earlier transactions cannot be recovered. Nothing cascades.

Map those two onto an agent fleet and the shape barely changes. The operator's HSM holds the root, the way an acquirer holds a BDK, and each agent gets a key derived from it, the way each terminal does. The one thing that changes is the endpoint. A terminal is provisioned once in a factory, while an agent asks the HSM to sign on demand. The containment is identical.



But keeping the key safe is only half the gap. Custody settles where the key lives, and containment caps the blast radius of any single compromise. Neither decides what the agent is allowed to do. Even with the key sealed in hardware, a manipulated agent can still ask the HSM to sign a purchase it was never authorized to make, and the signature would be perfectly valid. That second question, authority, needs its own answer, and the next section brings the two together.

A hybrid model: custody and authority

Custody is solved by the hierarchy from the last section. Authority has an equally proven answer, and it is not a payments invention. It is per-hop delegation, the shape the agent-identity drafts are converging on. The hybrid model is simply this. Run the two as parallel tracks and bind them together at the one moment that matters, when the agent signs.

The custody track is the derivation hierarchy, now scoped to an agent. The operator's HSM holds the root, derives a per-agent key, and derives a fresh signing key per session, on demand and never stored. The agent holds no key, only a short-lived credential that lets it ask the HSM to sign. A compromised agent cannot walk off with a key. At worst it can use its own session, for as long as it lasts and within the scope it was granted.

The authority track passes a grant down a chain, from the human to a shopping agent to a payment agent, re-signed and narrowed at every hop, from full wallet to a category budget to a single cart. Each link is cryptographically bound to its parent, so the chain is verifiable and can only ever narrow. No agent can exercise more than it was handed, and any one link can be revoked without collapsing the rest.

The two tracks fit because of where the derivation hierarchy anchors. Delegation changes slowly, only when authority is genuinely re-delegated. Key derivation changes fast, every session, for containment. The hierarchy is anchored per hop but advances per session, and one serial number carries both, a delegation anchor fixed to the agent's place in the chain and a session counter beneath it. That reuses the

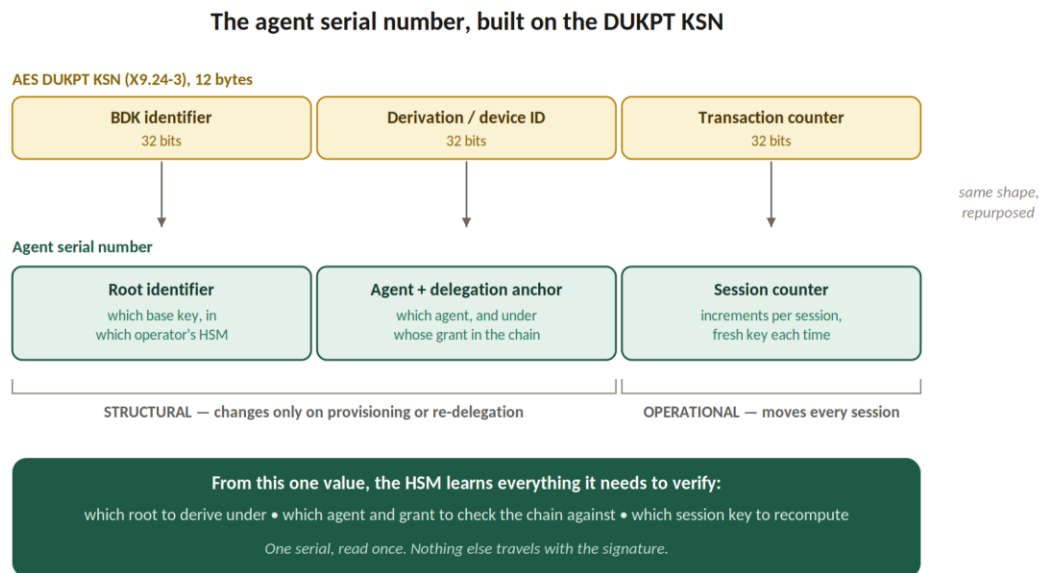
pattern a DUKPT serial number already uses, a device identifier and a transaction counter sitting under a base-key root, rather than inventing something new.

A DUKPT-style serial for agents

It is worth being concrete about the one field everything hangs on, because its structure is the whole design in miniature. AES DUKPT already defines a twelve-byte key serial number in three parts, a base-derivation-key identifier, a device identifier, and a transaction counter. The agent serial keeps that shape and repurposes each part.

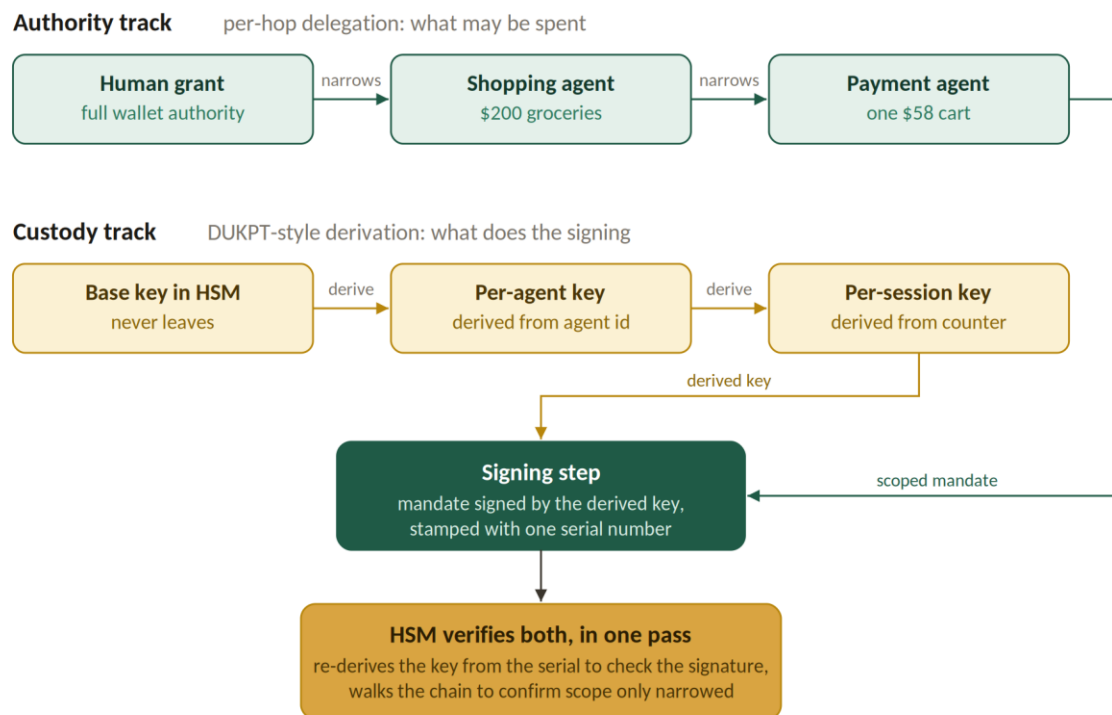
The first part names the root. It identifies which base derivation key, inside which operator's HSM, this signature chains back to, exactly as the BDK identifier does today. The second part identifies the agent, in place of the terminal's device ID, and carries the delegation anchor, a value bound to the agent's position in the authority chain, so the serial names not just which agent signed but under whose grant. The third part is the session counter, the direct analogue of the transaction counter, incrementing once per signing session and driving a fresh derived key each time.

That division is what lets one value do two jobs. The root and agent parts are structural. They change only when an agent is provisioned or re-delegated, and they anchor the key to a place in the hierarchy. The counter is operational. It moves every session and guarantees a key is never reused. A receiving HSM reads the serial once and learns everything it needs, which root to derive under, which agent and grant to check the delegation chain against, and which session key to recompute. Nothing else has to travel with the signature.



The agent serial number, mapped onto the AES DUKPT key serial number.

They meet at the signing step. The agent signs the scoped mandate from the authority track with the derived key from the custody track, stamped with that one serial number. The HSM verifies both in a single pass: it re-derives the key from the serial to check the signature, and it walks the delegation chain to confirm scope only narrowed. Both must pass, or the payment is refused.



The hybrid model: two proven schemes run in parallel and bind at the signing step.

This is the payoff of folding both tracks into one serial. Identity and authorization are normally two separate checks, verified by two different mechanisms at two different points. Here they collapse into a single verification. Re-deriving the key from the serial proves custody, and walking the chain the serial points to proves authority, from the same artifact in the same pass. A design that proves two independent properties with one verification has fewer places to disagree with itself, and fewer seams for an attacker to work between.

Putting the serial under attack

A mechanism this compact earns its place only if it holds up when someone pushes on it. Specifying it to the field, stating what binds it, and then naming the attacks it has to survive shows whether it does. The twelve bytes carry three parts. The first four identify the operator root. The middle four are the delegation anchor, a reference to the scope grant the agent is operating under. The last four are the session counter, incrementing once per signing session. The first eight are the stable identity of the agent and its grant, and the last four move with every transaction.

Two bindings keep the serial from being pulled apart. The serial is the input to key derivation, so any change to it yields a different key and the signature fails. And the signing operation covers the hash of the scope grant, so a signature only verifies against the exact scope it was issued for. The first binding ties the serial to custody. The second ties it to authority.

Assume an attacker sees every serial on the wire, since serials travel with the request, and can replay, increment, forge, swap the scope, or reuse another agent’s serial. Assume the HSM and the operator root stay intact, because once those fall the question is no longer about the serial.

Table 3

Attack	What stops it
Replay a valid serial and signature	The verifier rejects any counter at or below the highest it has seen for that agent, the anti-replay rule DUKPT already enforces
Roll the counter back or reuse it	The same rule. The counter only moves forward, and a repeat is refused
Attach a valid signature to a broader scope	The signature covers the scope hash, so a swapped grant no longer verifies
Use one agent's serial to sign for another	The per-agent bytes derive a different key, and the signature fails
Point two grants at the same anchor	The anchor is a reference, not the binding. The full scope hash in the signed payload is what has to match

That last row is where specifying the serial to the field pays off. The anchor is four bytes, and four bytes cannot carry a collision-resistant commitment to a scope. Collisions would show up after roughly sixty-five thousand grants, far too few for a fleet. So the serial does not try. The anchor is a lookup reference, and the real commitment to scope is the hash the HSM signs, which is full length and does not collide in practice. The serial stays the twelve bytes the networks already carry, and the binding it cannot hold moves to where there is room for it.

The counter is thirty-two bits, roughly four billion sessions before an agent must be re-keyed, ample for any single agent and bounded cleanly when it is reached.

An attacker without the operator root cannot produce a serial, scope, and signature that verifies for a scope broader than the one the anchor references, except by finding a collision in the signed scope hash, which the hash length rules out.

One limit is unchanged from the rest of the paper. None of this stops an agent that stays inside a valid scope but is manipulated into a bad purchase within it. The serial proves the grant is authentic. It does not judge the purchase.

How the whole model holds up

Composed this way, each half covers for the other. The payoff is that most failures now run into two independent defenses instead of one. Compromise a live agent and custody has already confined the

damage to its one expiring session, while the delegation scope caps what that session could authorize anyway. Neither DUKPT nor delegation delivers that doubling on its own, and the table traces its failure by failure.

Because each claim here is a property of the design rather than a measured result, it can be checked by reasoning rather than by benchmark. The following table sets the hybrid against the three alternatives, one threat at a time. Read down the last column, the hybrid is the only one that never leaves a row fully exposed.

Threat	Key in agent environment	DUKPT custody alone	Per-hop delegation alone	Hybrid model
Live agent compromised	Full account drained, nothing contained	One session exposed, fleet and history safe	No custody guarantee, the key was never contained	No key to steal, attacker limited to the live signing session, and only within delegation scope
Forged link that widens authority	No authority model to stop it	No authority model to stop it	Rejected, scope can only narrow	Rejected by the narrowing check, and would still need a key the HSM can re-derive
Prompt injection into the agent	Attacker signs freely with the resident key	Attacker signs, but only for one session	Attackers still need a key to sign with	Attackers get one scoped session at most, no resident key to abuse
Revoke one compromised agent	Not possible without rotating the shared key	Per-agent, via its derivation identity	Per-branch, via its delegation link	One serial number revokes both the key derivation and the authority branch
Compromise of the base secret	N/A	Total, BDK compromise exposes the fleet	N/A (holds no keys)	Total on the custody side, contained by HSM/LMK discipline, same as any payment root
Signature scheme must change (PQC)	Every resident key re-provisioned by hand	Re-key the derivation function fleet-wide	Re-sign the delegation chains	Both tracks migrate, the serial structure and hierarchy are algorithm-agnostic

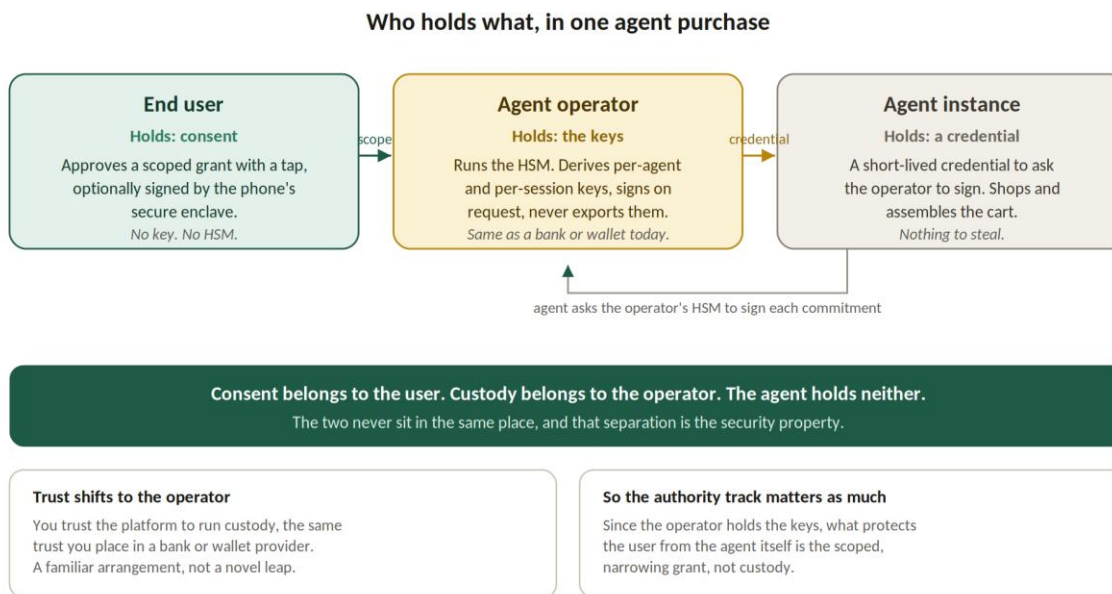
Two limits keep the table from overclaiming. The base-secret row shows the hybrid is not magic. Compromise the root and the custody side falls, exactly as it would for any LMK or BDK, which is precisely why that key is held to hardware discipline and never leaves. And every cell describes a property of the design, not a figure from a running system. Turning these guarantees into measured latency and throughput numbers requires the reference implementation this paper stops short of, and flags as the natural next step.

The model has one real open problem, and everything above rests on it. A card terminal gets its key injected once, slowly, in a controlled facility. Agents hold no key, so nothing is injected, but something harder takes its place. Agents spawn in milliseconds, by the thousands, with no human present, and each time the operator's HSM has to be sure the request genuinely comes from the agent it claims to be, carrying only the scope it was granted. That means attesting a freshly spawned agent and binding it to its serial and scope at spawn speed, cryptographically, not on the agent's own say-so. Today's tools each

cover part of this. Some can attest a new agent quickly, others can bind authority cryptographically, but nothing yet does both together, fast enough. That gap is where this model meets the edge of what exists today, and naming it is part of the proposal.

A day in the life of an agent purchase

The everyday case settles a question the mechanism leaves open. Whose key is this, and who has to run the hardware? Picture a standing instruction. Keep the kitchen stocked, and book my usual barber when a slot opens. For days, an agent is initiating payments on your behalf with nobody watching each one. Three parties share the work.



The three parties in an agent purchase, and what each one holds.

Open questions

The hybrid model has limits beyond the ones already named. Two stand out, because they are where the design meets genuinely unsolved work.

The first is delegation depth. When one agent delegates to another, authority must only ever narrow, and every hop must be verifiable. RFC 8693 records delegation chains in nested act claims but treats the prior actors as informational only [8], and work on multi-hop delegation for AI agents has shown how quickly informational becomes forgeable [16]. Several 2026 drafts are now working on cryptographically verifiable chains [9][12]. A payments-grade version needs attenuation enforced when each grant is minted and again when it is verified, the way a key block binds usage to the key, not carried along as a hint.

The second is post-quantum migration, and here the model has a structural advantage. The agent-facing public-key signatures in these frameworks are Ed25519 or ECDSA. The day those must be replaced, the cost depends entirely on where the keys live. In a model where each agent holds its own resident key,

migration means re-provisioning the entire fleet, every agent re-identified under the new scheme, one at a time. In the hybrid model nothing resident has to change, because agents hold no keys. The derivation function and the signing step are the only places an algorithm appears, and both sit inside the operator's HSM. Swap Ed25519 for a lattice scheme like ML-DSA [10] there, and every future per-agent and per-session key is post-quantum, without touching a single deployed agent. The serial structure and the hierarchy are algorithm-agnostic by construction.

The caveat is signature size. ML-DSA signatures are kilobytes where Ed25519 is sixty-four bytes, and that weight lands on every mandate, every directory entry, and every signature that travels in an HTTP header, some of which are already close to today's size limits. The migration path is clean. The payload budget is the real constraint, and it is one the whole industry shares, not something unique to this model. The point is only that custody done this way turns a fleet-wide re-provisioning into a change at two points inside the HSM, the difference between a migration that is feasible and one that is not.

Recommendations

Treat the agent as an untrusted proposer and the network's signed controls as the deterministic authority. Instrument every checkpoint in the map before switching on agent-initiated flows, and use the hybrid model as the template for the one checkpoint the networks leave to you.

Issuers: Adopt scoped agent tokens with spend caps, merchant limits, expiry, and real-time per-agent revocation. Make agent identity a first-class field so disputes can be attributed.

Acquirers and merchants: Verify the agent's identity signature and require a signed mandate before submitting a transaction for authorization. Separate browse from pay, and turn away anonymous automated traffic on the payment path.

Agent developers: Never let the model decide on a commitment. Bind identity, scope, price, amount, and consent with signatures, and build on hardware-backed key custody from day one rather than retrofitting it.

Standards and security teams: Prioritize agent key lifecycle, revocation latency, and a post-quantum migration path, and settle how mandates interact with SCA and PCI scope.

These are the questions a payments security team has answered for terminals for years, now pointed at agents. Whoever answers them well will define what secure means for this next layer of commerce, and they will do it in the vocabulary of the key hierarchy payments already trusts, not by importing a web-wallet model wholesale. The winners in agentic payments won't be whoever fields the most autonomous agents. They'll be whoever draws the deterministic boundary in the right place, and can prove, cryptographically, that the agent never crossed it.

References

- [1] International Monetary Fund. How Agentic AI Will Reshape Payments. IMF Notes, Vol. 2026, Issue 004, April 2026. <https://www.elibrary.imf.org/view/journals/068/2026/004/article-A001-en.xml>
- [2] Visa. Trusted Agent Protocol (TAP), repository and specifications. October 2025. <https://github.com/visa/trusted-agent-protocol> and <https://developer.visa.com/capabilities/trusted-agent-protocol/trusted-agent-protocol-specifications>

- [3] Mastercard. Mastercard Agent Pay. April 2025.
<https://www.mastercard.com/global/en/business/artificial-intelligence/mastercard-agent-pay.html>
- [4] Google Cloud. Announcing the Agent Payments Protocol (AP2). September 2025.
<https://cloud.google.com/blog/products/ai-machine-learning/announcing-agents-to-payments-ap2-protocol>
- [5] Cloud Security Alliance. Secure Use of the Agent Payments Protocol (AP2): A Framework for Trustworthy AI-Driven Transactions. October 2025.
<https://cloudsecurityalliance.org/blog/2025/10/06/secure-use-of-the-agent-payments-protocol-ap2-a-framework-for-trustworthy-ai-driven-transactions>
- [6] Internet Engineering Task Force. RFC 9421: HTTP Message Signatures. February 2024.
<https://www.rfc-editor.org/rfc/rfc9421>
- [7] World Wide Web Consortium. Verifiable Credentials Data Model v2.0. W3C Recommendation, 2024.
<https://www.w3.org/TR/vc-data-model-2.0/>
- [8] Internet Engineering Task Force. RFC 8693: OAuth 2.0 Token Exchange. January 2020.
<https://www.rfc-editor.org/rfc/rfc8693>
- [9] Prasad, A., Krishnan, R., Lopez, D., and Addepalli, S. Cryptographically Verifiable Actor Chains for OAuth 2.0 Token Exchange. Work in Progress, Internet-Draft, draft-mw-oauth-actor-chain (replaces draft-mw-spice-actor-chain), 2026. <https://datatracker.ietf.org/doc/draft-mw-oauth-actor-chain/>
- [10] National Institute of Standards and Technology. Post-Quantum Cryptography Standardization. 2024.
<https://csrc.nist.gov/projects/post-quantum-cryptography>
- [11] European Banking Authority. Strong Customer Authentication under PSD2. 2019.
https://finance.ec.europa.eu/publications/strong-customer-authentication-requirement-psd2-comes-force_en
- [12] Rampalli, K. PEDIGREE: Verifiable Delegation Identity for Agentic AI Systems. Work in Progress, Internet-Draft, draft-rampalli-pedigree, 2026. <https://datatracker.ietf.org/doc/draft-rampalli-pedigree/>
- [13] Sweeney. Credential Delegation Protocol for AI Agents in Multi-System Environments. Work in Progress, Internet-Draft, draft-sweeney-wimse-credential-delegation-00, July 2026. Listed at <https://datatracker.ietf.org/doc/draft-sweeney-wimse-credential-delegation/>
- [14] Halborn. AI Agent Wallet Key Management Best Practices. Halborn blog, 2026.
<https://www.halborn.com/blog/post/ai-agent-wallet-key-management-best-practices>
- [15] Openfort. AI Agent Wallet Compliance: Custody and Control Requirements. Openfort blog, September 2026. <https://www.openfort.io/blog/ai-agent-wallet-compliance-custody-controls>
- [16] WorkOS. AI agents and the multi-hop delegation problem. WorkOS blog, 2026.
<https://workos.com/blog/oauth-multi-hop-delegation-ai-agents>