

Design for Secure Revenue Projection

Security is a component, not a review. Send, hold or refuse at every hop a revenue signal takes, with a record of each decision.

For: marketing and growth leads, analytics and data engineers, designers who build the consent

and account surfaces, and the compliance reviewer who signs off what leaves the store. **Genre:** design specification.

A revenue signal is only as trustworthy as the least-checked hop it passed through. Design the checks as a component, with states and tests, the same way you design a button.

DOCUMENT CONTROL

Status	Published; the component is partly built, and each gap is a failing test
Version	1.0
Owner	Platform security lead, with the marketing data owner
Evidence basis	<code>persephonepunch/ucp-purchase-agent</code> at <code>b753d6c</code> (main); the production Worker at <code>282d947</code>
Measured	24 October 2026
Review cycle	Quarterly, and whenever a channel or a vendor is added

Vocabulary

The same words are used loosely in marketing, engineering and compliance. Here they mean one thing each:

TERM	MEANS HERE	DOES NOT MEAN
Revenue projection	Sending a revenue signal about customers from the store's own record to a channel that acts on it: a conversion value for bidding, a predicted value, an audience list, a product feed price	A financial forecast. Forecasting (predicted lifetime value) is one <i>input</i> to a projection
Revenue signal	A conversion value, a predicted lifetime value (pLTV) score, an audience membership, or a price in a feed	Raw orders or customer records. Those stay in the store's own systems
Channel	The system that receives the signal and acts: Google Ads, Customer Match, Merchant Center, GA4, Klaviyo, Meta	The plugin that carries it. The plugin is transport
Projection target	A channel's adapter: the code that shapes a signal for one channel	The record. The record has one home; targets get copies
Component	A designed part with inputs, states, guarantees and tests, reused at every place it applies	A review, a policy document, or a one-off check in one integration
Security component	The part every projection passes through that decides <i>send</i> , <i>hold</i> or <i>refuse</i> , and leaves a record of the decision	Encryption alone, or TLS alone. Both are inside it, and neither is all of it
Purpose	Why a record exists: <code>marketing</code> , <code>analytics</code> , <code>support</code> , <code>fraud_prevention</code> . It decides which channels may receive it	A tag for reporting

The question this answers

Every store already projects revenue. The Google channel sends order values. A Klaviyo flow sends spend. A pLTV model sends predicted values. The question

is not whether to do it, but **who can prove, afterwards, that each signal was allowed to leave, and was the signal it claimed to be.**

Most stacks answer with a review: a privacy assessment before launch, a penetration test once a year. A review checks the design once. Revenue signals leave every few seconds, through plugins that change monthly. The answer that holds up is a component that runs on every signal.

The components of a revenue projection

A projection is built from six components. Five are familiar. The sixth, security, is usually left out of the design and added as a review, which is the gap this document closes.

COMPONENT	JOB	LIVES IN	WHAT BREAKS WITHOUT IT
Identity	One pseudonymous ID per customer, linked to each vendor's ID	Xano, behind Cloudflare	Each plugin's mapping table becomes the identity record, and it leaves with the plugin
Consent	The customer's current choice for each signal (<code>ad_user_data</code> , <code>ad_personalization</code> , <code>analytics_storage</code>)	Xano, written when the customer makes a choice	Consent is inferred from a marketing opt-in, and nobody can show it was granted
Value	The number sent: order total, or a pLTV score	BigQuery ML scores, returned to the Worker	Bidding learns from first orders only
Purpose	Which uses a record is allowed	On every record on the bus	A fraud score or a support ticket reaches an ad audience
Channel adapter	Shapes the signal for one channel	Worker, one adapter per channel	Each channel's quirks leak into the record
Security	Decides <i>send</i> , <i>hold</i> or <i>refuse</i> at every hop, and records it	Worker and Xano, on every projection	Every other component can be bypassed, and no one can tell

Identity, consent, value and purpose describe the signal. The adapter shapes it. **Security is the only component that checks the other five actually held.**

The security component

Contract

The component takes one signal and returns one decision:

```
input:  signal (value, pseudonymous ID, destination, purpose)
        + the consent state for that ID, read now, not cached from earlier
        + the sender's proof (a webhook signature, a service key)

output: send | hold | refuse
        + reason
        + a record: what, to where, under which consent and purpose, when
```

It runs at every hop, not once at the edge. A check that only runs at the edge is bypassed by the first internal job that calls a channel directly.

States

STATE	WHEN	WHAT HAPPENS TO THE SIGNAL	WHAT THE RECORD SAYS
send	Sender verified, purpose allows the destination, consent granted for this signal	Shaped by the adapter and delivered	Sent, with the consent state and purpose
hold	Consent unknown, or the consent service didn't answer	Queued; nothing leaves	Held, with the reason. A held signal is retried, never assumed granted
refuse	Sender not verified, purpose doesn't allow the destination, consent denied, or the tenant's own gate isn't set up	Dropped	Refused, with the reason
degraded	The decision was made, but the record couldn't be signed (signing key expired or unavailable)	The decision stands	Written unsigned and marked, so a later job can sign it. The record is never lost to save the signature

The rule across all four: **every failure fails toward less exposure**. A gate that can't reach the consent record holds; it doesn't send.

Checks at each hop

HOP	SECURITY CHECK	WHY IT IS HERE, NOT ELSEWHERE
Order or event arrives (Shopify, Stripe, a vendor webhook)	Verify the sender's signature in constant time before reading the body	An unsigned or forged event would become a revenue signal
Record joins the bus	Parse into a declared shape; refuse unknown fields and oversized bodies	Injection happens at transfer points, in whatever code reads the payload
Identity is attached	Use the pseudonymous ID; never put the email or phone on the bus	A leak of the bus then leaks no contact details
Value is computed	The model only reads and scores; it never decides access	A model that decides who may be used can't be audited the way a rule can
Gate before a channel	Ask the release gate per person, per destination: purpose, then consent, then the tenant's own gate	Consent is per person. A batch stamped "granted" is the most common silent failure
Adapter sends	One Google account per tenant; hashed contacts only; keys held as Worker secrets	A shared or mistyped account ID sends one store's customers to another's audience
Channel acknowledges	Record the result on the hash-chained ledger, signed with a key the merchant holds	The record must outlive the plugin, the vendor and the vendor's logs
Domain and transport	TLS everywhere; CAA names the CAs that may issue for the domain	A distrusted CA lets someone impersonate an endpoint the channel calls back
Erasure	Fan out from the identity spine to every channel copy, each with its own record	Erasure that reaches the app it was sent to, and not the audiences, isn't erasure

Failure modes, as they look from outside

Each of these looks exactly like a working system until someone checks the record:

- **A 200 that sent the wrong thing.** The audience upload succeeds, but every member was stamped as consented because the batch was. The fix is a decision per person, recorded per person.
- **A gate that opens late.** Consent is checked after the upload is queued. From the dashboard, a late gate and a working gate look the same.
- **An erasure that erased the original.** The redact webhook returns 200 and deletes the store's copy. The Customer Match list and the pLTV score still hold the person.
- **A fork in the record.** Two writers append to the same ledger at once and both succeed, leaving two histories. Only a constraint on the previous hash makes the second write fail.
- **A reporting address no one receives.** The CAA reporting record exists, but its value starts with a space, so a strict certificate authority ignores it.

The two shapes the component checks

A security component can only check what it can read. Two shapes carry nearly every revenue signal: the **product shape** Google Merchant reads, and the **customer event shape** on the user tags bus. Both are fixed shapes, not free-form payloads, which is what lets the component, and an AI caller, rely on them.

The Google Merchant data layer shape

A product's price, availability and identifiers are revenue signals too: they decide what an ad or an AI shopping agent offers, and at what price. CRM Sync composes them in the Merchant API `products_v1` shape, from one composer that feeds three surfaces: the public catalog at `/merchant/products`, three read-only Merchant tools on the MCP endpoint (`merchant_get_product_by_name`,

`merchant_list_products`, `merchant_list_aggregate_product_statuses`), and a JSON block inside each product page. One composer means the three can't disagree.

One product, as served today:

```
{
  "name": "accounts/example.myshopify.com/products/en~US~<offerId>",
  "offerId": "<offerId>",
  "contentLanguage": "en",
  "feedLabel": "US",
  "productAttributes": {
    "title": "...", "description": "...", "link": "https://...", "imageLink": "https://...",
    "price": { "amountMicros": "129000000", "currencyCode": "USD" },
    "availability": "IN_STOCK", "condition": "NEW", "brand": "...",
    "identifierExists": false,
    "productTypes": ["..."],
    "shipping": [{ "country": "US" }],
    "shoppingAdsExcludedCountries": ["KP", "IR", "SY"]
  },
  "productStatus": { "itemLevelIssues": [
    { "code": "missing_identifier", "severity": "warning", "attribute": "gtins",
      "description": "Neither GTIN nor MPN present" } ] }
}
```

FIELD	SECURITY RULE	WHAT GOES WRONG WITHOUT IT
<code>name</code> , <code>offerId</code>	The offer ID is part of the resource name and never changes or gets reused	A reused ID moves one product's history, reviews and bids onto another
<code>price.amountMicros</code> + <code>currencyCode</code>	Read from the live store, in the market's own currency, never converted	An exchange-rate move changes what an ad promises
<code>availability</code> , <code>condition</code>	Enum values (<code>IN_STOCK</code> , <code>NEW</code>), not free text	A typo is ignored by Google, and the offer silently stops serving
<code>gtins</code> or <code>identifierExists: false</code>	A real identifier, or a plain statement that there isn't one. Never an invented one	A made-up GTIN is a false claim to Google and to every agent that trusts it
<code>shipping</code> , <code>shoppingAdsExcludedCountries</code>	Shipping countries come from the store's country rules; sanctioned countries are excluded	An ad reaches a market the store may not sell to
<code>productStatus.itemLevelIssues</code>	Missing image, price, identifier or description reported, not hidden	A broken offer looks healthy until it is disapproved
The JSON block in the page	<code><</code> is escaped, and the block is stripped and rewritten as a unit	A product description becomes a script on the page

The walk from a flat product CSV to this shape, including why a missing [Handle](#) makes the same file read as 4 products or 169, is in [From product.csv to Consented Conversions](#).

The user tags event bus

The customer side moves as events, not as copies of the customer record. Three tables carry it:

TABLE	HOLDS	SECURITY PROPERTY
Tag catalog	Each tag's key, kind, label, scope, whether it emits events, its event topic, and whether it is consent-linked	The catalog has no personal data. It is mirrored to Shopify as <code>user_tag</code> metaobjects, so a theme can read tag definitions and never membership
Tag membership	Which user holds which tag, with status, a version number, timestamps, and what fired the change	Withdrawing consent ends consent-linked memberships, bumps the version and records <code>fired_by: consent_withdrawn</code>
Event bus (<code>ucp_funnel_history</code>)	One row per event: an event ID, an idempotency key, the user, the domain, the event type, status and version, a payload, the consent basis, the region, what fired it, its source, and the channels it may reach	A redelivered event with the same idempotency key is a no-op

Seven domains write to the bus today: `tag` , `consent` , `order` , `return` , `agentic` , `omnibus_discount` and `audience` . An order event carries a **conversion record** in its payload, built for revenue projection rather than as a log line:

CONVERSION RECORD FIELD	WHY IT IS IN THE RECORD
<code>value_minor</code> + <code>currency</code> + <code>market</code>	Value in minor units in the market's currency, never converted, so a past conversion's worth can't drift
Identity per scope	The person's account ID is recorded separately from a browser's cookie ID, so one customer on two devices isn't trained as two
<code>consent</code> + <code>consent_version</code> , as at capture	Consent is a fact about a moment. Exporting under today's consent would claim a basis that didn't exist when the data was collected
<code>pltv_score</code> , <code>pltv_model_version</code> , <code>pltv_scored_at</code>	The prediction as it stood, with the model that made it, so it can be checked against what actually happened
<code>realized_value</code> , <code>realized_at</code>	What the customer actually spent, recorded later, so the prediction can be marked right or wrong

What the bus doesn't do yet. It is an event log, not yet an evidence chain:

- **Writes are best-effort.** A failed write is logged and the caller carries on, so an event can be missed without anyone being told. An outbox with retries would close it.
- **Rows aren't chained or signed.** Consent and erasure events are also written to the hash-chained, Ed25519-signed ledger; the bus rows themselves are not.
- **The idempotency check reads, then writes.** A unique index on the key is what actually stops a duplicate when two deliveries race.

Setting up an AI caller against both shapes

An AI agent, a chat assistant or an automated job reads these shapes to decide what to offer and what a customer has agreed to. The security component treats it like any other caller:

RULE	STATE
The agent reads products through the Merchant shape only: the public catalog or the three read-only Merchant tools, never the store admin API	Built
The agent's actions are written to the bus under the <code>agentic</code> domain, each with an idempotency key	Built
Orders an agent places pass through the same release gate before any conversion or audience is sent	Built for the destinations listed above
The agent spends only inside a signed mandate, and revoking the mandate stops it	Built (mandate enforcement has its own test suite)
The agent reads a customer's tags only through a tool scoped by its mandate, and can't set a consent-linked tag	Pattern, not built
Bus events an agent relies on are chained and signed, so it can prove what it acted on	Not built

What is built, and how it is tested

The component is partly built. Each part is pinned by a test in the CRM Sync test harness: green tests guard what is built, and red tests fail on purpose until a gap is built.

CHECK	STATE	EVIDENCE
Webhook signatures compared in constant time	Built	Green test: no plain <code>===</code> on the signature
Release gate asked per person before Customer Match, GA4 purchase and user properties, and Merchant reviews	Built	Green tests in the release-gate suite; the gate fails closed when a tenant's own gate isn't provisioned
One Google account per tenant	Built	Green tests in the Google-bindings suite: an account already bound to another store is refused
Consent and erasure records chained by <code>seq</code> and <code>prev_hash</code> , signed with Ed25519, kept when signing fails	Built	Green tests: chain constraint, signing, degraded path
Erasure fan-out to Shopify, Klaviyo, Salesforce, Adobe, Google Customer Match, the pLTV score and agent mandates	Built	Green test; the Google removal is written to the chain
CAA names the permitted CAs, with a valid reporting address; no emSign in any served chain	Built	Green tests, run live against DNS and the served certificates
No private key in the source; the public key list carries public keys only	Built	Green tests
Purpose tags that keep fraud and support data out of ad destinations	Not built	Red test
Erasure to helpdesk, fraud and consent-platform vendors (Gorgias, Sprinklr, Signifyd, OneTrust)	Not built	Red test
Arrival time recorded and clock skew flagged on inbound vendor events	Not built	Red test
A Rust/WebAssembly shape gate on inbound vendor payloads	Not built	Red test
One table linking every vendor's customer ID to the identity spine	Not built	Red test

CHECK	STATE	EVIDENCE
The per-destination consent mapper called on every send	Written and tested, not yet called; the release gate decides today	—
One Merchant composer for the catalog, the Merchant tools and the page block, in the <code>products_v1</code> shape	Built	Live: <code>/merchant/products</code> returns <code>productAttributes</code> , enum values and the <code>en~US~<offerId></code> name
Bus events idempotent per key, consent-linked memberships ended on withdrawal	Built	Code; no harness test yet
Bus writes retried through an outbox	Not built	No test yet
Bus rows chained and signed	Not built	No test yet

The vendor-by-vendor detail behind the red rows is in The plugin in the middle, and which channel data may feed conversions, audiences, pLTV or model training is in Google audiences and conversions with Vertex AI, and where LoRA fits. The CAA steps, and a script that tests a domain red then green, are in CAA records.

Designing the visible part

Most of the component is invisible. Three parts reach a screen, and they are designed like any other component, with states:

SURFACE	STATES TO DESIGN	WHAT IT MUST NEVER DO
Consent banner and preferences	Not yet asked; granted per signal; denied per signal; withdrawn	Offer one "accept" that grants ad signals the customer never saw named
Account page ("what we share")	Shared with each channel; held; erased, with the date of each	Show a channel as connected after an erasure was sent but not confirmed
Merchant dashboard (projection health)	Sent, held and refused counts per channel, with reasons	Show only <i>sent</i> . A dashboard without <i>held</i> and <i>refused</i> hides the gate

The account and dashboard states come straight from the component's records, so the screen can't disagree with what was sent.

What it costs

- **Fewer signals reach the channel.** Holding until consent is known, and refusing on purpose, means smaller audiences and fewer conversion values than a stack that sends everything. Bidding learns from less data. That is the trade, and it should be stated in the plan, not discovered.
- **Latency.** A per-person gate adds a lookup before each send, and a batch is gated member by member. It is never free.
- **One more system to own.** The identity spine, the ledger and the gate run in Xano and Cloudflare. If those stop, projection stops, by design. That is a dependency the team takes on.
- **Vendor coverage lags.** Each new channel needs an adapter, an erasure call and a test before it is trusted. Until then it is a red row.
- **The tests prove the code, not the vendor.** A green erasure test proves the call is made; it doesn't prove the vendor deleted anything. Confirmations still have to be recorded and checked.

What to do next

1. **List every revenue signal that leaves the store today**, with the channel, the plugin that carries it, and the purpose of the data. Most teams find signals no one owns.
2. **Write the security component's checks as tests before changing anything.** Each check you can't pass yet is a red test, and the list of red tests is the plan.
3. **Put the gate in front of the highest-volume signal first**, usually the audience upload or the conversion value, and make it decide per person.
4. **Move identity and consent to a record you own**, so the gate reads the current choice, not the plugin's copy.
5. **Add the visible states** to the banner, the account page and the dashboard from the same records.
6. **Turn one red row green at a time**, and add each new channel with its adapter, its erasure call and its test together.

Resources and downloads

This document:

FORMAT	LINK
Web article	design-secure-revenue-projection.html
PDF download	design-secure-revenue-projection.pdf
In the CRM Sync knowledge base	Knowledge base → Design for Secure Revenue Projection
Book chapter	Security → Design for Secure Revenue Projection
The two data shapes	The two shapes the component checks

Companion brief: *AI Security: Challenges, Benefits, and Tools* (24 October 2026, 5 pages). It pairs five AI security risks (memory-unsafe native stacks, prompt injection, model supply chain, data leakage, over-privileged agents) with the defence each one gets from AI, shows a reference architecture for a secure AI data layer, and lists open-source and commercial tools and the frameworks auditors use (OWASP Top 10 for LLM Applications, MITRE ATLAS, NIST AI RMF, Google SAIF).

- [Download the brief \(PDF, 223 KB\)](#)

The brief's architecture is the target design. Its Rust Worker trust boundary corresponds to the Rust shape gate, which is a red row in [What is built, and how it is tested](#): read the brief for where the design is going, and this document for what holds today.

Related sections:

- [The plugin in the middle](#): where data ownership breaks between Shopify, CRM, service, fraud, consent and Google
- [Google audiences and conversions with Vertex AI, and where LoRA fits](#): which channel data may feed conversions, audiences, pLTV or model training
- [CAA records](#): naming the CAs that may issue for a domain, and a red-then-green test script

- From product.csv to Consented Conversions: the walk from a flat catalog file to the Merchant shape